1. INTRODUCTION

This chapter describes the current development trends as well as the problems with Java desktop applications and the resulting need for Rich Clients. Moreover it defines the goals of this work. Beside this it gives basic definition of terms and motivation for the research followed by a brief introduction of all containing chapters given as the outline of entire thesis.
1.1 DEVELOPMENT TRENDS
Due to ease of maintenance in terms of deployment and upgrading web applications, allow one to reach a larger audience than software applications which are distributed using common data media. Although, they deny the user experience that a full-fledged desktop application can provide. Therefore, there is a movement among enterprise applications which is a great change in the past few years trend that abandoned client/server applications in favor of interfaces which supports the web.

Rich Client applications, for providing the platform independency, use native Graphical user interface components that can interact with the desktop. By adding the application platform dependent features like drag-drop, copy-paste, it increases the usability in the end. This leads to faster work with fewer human errors. Typically, a Rich Client application will be Graphical user interface concentrated with the retrieval and update of data being the core functional requirement, but it also enables the user to join together with the other applications which are residing at client side such as spreadsheet applications, E-mail, etc.

Moreover, as the application will on the whole physically reside at client side, it takes advantage of the handing out control of industry standard PCs ensuring fast responses to user requests.

Microsoft has already recognizable with this matter and addresses it with its .net framework and with very smart client technology. This technology merges the power of the PC with the reach of the web. By the way, Smart client software can be deployed and updated from a centralized web server [1]. This simple deployment helps to ensure that users have the latest version of the application beside help IT departments to reduce deployment costs.

The Java™ world has to bring its own technology for Rich Clients to gain back power on the desktop. IBM addresses this Thing with the Eclipse technology, which combines the power of true platform independence and the implementation of native components to perk up the user convenience when working with the application. Or we can say that the
Eclipse framework provides a Java alternative to building robust, responsive and great-looking desktop applications.

1.2 SOME PROBLEMS WITH JAVA APPLICATIONS

As Java was always strong on the server-side, its client-side technologies have all suffered from strong criticisms and were never able to accomplish the expectations on the desktop market as predicted. Many people think that the alteration from the common user interface design principles, which were used in Rapid Application Development (RAD) environments like Visual Basic™, Delphi™ or similar, to the completely new Model-View-Controller (MVC) paradigm, which was introduced by Swing, led to the loss of the battle for the desktop. The majority of attempts to use Java for creating desktop applications have failed. The reasons for these failures are two vague rumors about the language:

- Java is faster than other programming languages.
- Swing can be easily used to create native looking applications.

Java programs are interpreted and therefore not as fast as native C or C++ applications. The host machine has to do one more step, the explanation of the byte code by the virtual machine, to perform an action. This problem can be addressed by using Just-In-Time (JIT) compilers and precompiling the source code. But when it comes to user interfaces and therefore Swing, the gap between a native and a Java application cannot be bridged that easy. Swing does not use native libraries and will therefore never be as responsive as a native application [2, 3, and 4].

1.3 OBJECTIVE

The main objective is to answer the question whether the Rich Client Platform eases or supports the development of Java desktop applications. For this purpose, this thesis copes with the development, implementation and analysis of Earth Navigator application using the Eclipse Rich Client Platform and possible problems in this context. For this I have used NASA Worldwind SDK which is an open source. It provides all jar files needed for View of Earth.
The problem which every developer face is the load of interaction in server based programming, which results in poor overall usability, compared both to static web pages and desktop applications. As far as Rich Client Platform is concern it could improve if developers had guidelines on hand – just like traditional web site usability can improve when guidelines are being used. It is much better than Thin Clients.

1.4 BASIC OF MODULAR APPLICATIONS

Modular programming is a technique for breaking down program functions into smaller modules. We can say that it is a software plan method which can be used to increase the extent to software which is collection of separate and similar components. The objective of a modularization is basically a software technique which is used to divide the system into a number of small modules for good system designing [5]. Every of these modules are basically create for fulfilling one function and provides the complete piece of code necessary to accomplish the desired task. Conceptually, modules improve maintainability by using logical boundaries linking components and it also used to correspond to a separation of concerns. By using the interfaces Modules are typically incorporated into the program [6]. An interface of a module needs to utter the elements that should be provided and should be required to accomplish the task of that particular module. The fundamentals which are provided in the interface are also visible to the other modules of that package or system. The implementation of the module which is declared in the interface contains the complete code that should be accomplished by the module. In the concern of a client module we can say that it is a crucial part of the application, which provides the most important features of the application that is the whole representation of the application. Beside this, an application is the collection of components which are related in some concern like pieces of application infrastructure, business logic, application-level services for authenticating users so that they can logged into the application interface. Although all these features sounds difficult to implement but modular programming makes it simple for the developer. Developing, testing, deploying of these applications is not only easy but extending of these applications in future is also
become easier. This is because of independent nature of the modules of each other and this feature is known as loosely coupled.

Modular programming is fundamentally a way used by the developer to divide a program into manageable parts with each having a specific purpose and reasonable size. This module may then be further split into smaller sub-modules. Since the program is separated into parts, the mapping of the code is easier to understand when compared to an individual looking at the structure of an entire program with hundreds of coded lines. Beside this, it also provides an added benefit that it reduces the chance of the programmer making a crucial mistake in the middle of the program.

This methodology is sometimes necessary. For example, when it comes to organizations that may require developing large projects this kind of approach should be a concern. Modular programming would allow the project manager to develop each module separately if needed. They can then be tested, debugged and finally skilled to be used throughout the organization to support other projects. Some may agree that this approach is parallel to object-oriented programming and structured programming. Modular programming is also significant because it allows programmers to conduct specific changes if program requirements deviate. The impact would be confined to an individual module and not the gross product. One more important point to convey is that this process allows programmers to focus their expertise to a particular module.

Modular programming may also have some disadvantages. Since each module is processed independently, there must be a link to connect them which can sometimes guide to problems if not tested fully. It is also note that there is no encapsulation so data can be accessed by any code wanting permission. This may bring up security issues unless resolved by the programmer. Finally, when there is a large project each of the team members involved must document their work clearly and coordinate their overall work efforts. Not doing so can have a deep unsafe result on the costs and timeline of the whole project.
1.4.1 Benefits of Building Modular Applications
As modular programming allows the developer to split his application into small and manageable parts, it makes easier to develop and test the application. Besides this, it also makes the application easier to extend in the future also. By providing these kinds of facilities, modular programming approach adds the flexibility and maintainability to development of complex applications. One of the major advantages of this kind of development is its functional independency. Each function or module provides a special and different kind of functionality as they are loosely coupled.

1.4.2 The Need for Modular Applications
1. Applications which are large in size are by default complex in nature. As they broken into smaller pieces, they become easier to develop. After developing these small pieces or modules, these modules assembled together to form the complete application.
2. One of the great advantages of this modularization is that these different components can be developing by different teams or programmers.
3. By developing these components as separate modules, they are not dependent on each other. So this method of developing the application removes the problem of complex dependency.

1.5 TERMINOLOGY USED IN WORK
This section holds the definition of various terms that are important for the understanding of this thesis.

1.5.1 Java
The Java language provides all the features of object-oriented programming language. Java has been developed by James Gosling and other engineers at Sun Microsystems. One of the main goals of this programming language is to achieve platform independence. For this reason, a specific machine language, the so-called Byte code, was developed. This byte code is not built to work on existing hardware architecture, but on the Virtual Machine (VM), which is an abstract machine specification. The advantage of
this architecture is that Java programs work on every platform that provides an implementation of the JVM. This virtual machine has to load compiled Java classes and execute the program code. These tasks are performed by two different components of the virtual machine:

1. Class-Loader
2. Execution-Engine

The class-loader as its name implies, is used to load Java classes into the main memory and to prepare their execution. The class-loaders are nothing just Java classes, but to load at least one of these classes at all, a non-Java class-loader is needed. This one is called the Bootstrap-Classloader, which is an inherent part of this virtual machine. The bootstrap class-loader has only one possibility to load Java classes, whereas the user-defined class-loaders may load Java classes from different sources like networks or databases. After the class-loader loaded a class into memory, the Execution Engine (EE) executes the included byte code. Every user-thread of a java program is equivalent to one instance of the execution engine. The engine executes the byte code step by step. If native calls are required by the Java program, they are delegated to the according dynamic linked library.

1.5.2 Just-In-Time (JIT) Compilers

The main work of Compilers is to generate machine-readable code for a specific platform. The advantage of compiling code before executing it is that the resulting programs tend to be very fast. But the resulting byte code can only be executed on the platform it was compiled for and is therefore not portable.

An interpreter is a program that runs on the target platform and expects source code as input. The source code is interpreted and executed immediately. The advantage of this execution type is that a program runs on every platform as long as an interpreter exists for it. But as the source code has to be parsed and executed on the fly, interpreters tend to be much slower than compilers.
There is an intermediate solution for executing programs, which is called p-code or byte code. Such systems compile the source-code into byte code. This byte code is then executed on different platforms using an interpreter. For the reason that the performance of these systems is not very good, these systems have never been a serious competitor. The Java platform was the first modern platform to approach the byte code solution. The first step is to compile the Java source code into the intermediate language, the so called byte code.

The execution engine can now interpret this byte code step by step. Every byte code instruction represents a sequence of assembly instructions of the underlying operating system. For the cause that the subroutines themselves are already assembly code, which can be directly executed on the specific CPU, the byte code can be replaced with assembly instructions during the interpretation.

1.5.3 Thin / Thick Clients

A thin client mostly executes its most of the processing part on a central server. And it uses as little hardware and software as possible at another side that is the client-side. Ideally the user will have only a screen, keyboard, a pointing device and enough processing power to handle display and communications.

Whereas a Thick Client which is also known as Rich Client does the most of processing at the client-side and passes only the data which is required for communications from client to server and server to client and archival storage to the server.

Thin clients are usually cheaper because the processing and storage of data is handled by the server. The disadvantage is that they require more bandwidth than thick clients, because the display data has to be transferred over the wire. Moreover, thin client servers have to be much more capable than thick client servers, because the clients request information on every user interaction. One advantage of this concept is that with thin clients only the server has to be upgraded to make updates available for all clients.
1.5.4 Rich Client Applications
Rich client User Interfaces not only provides rich user experience but high performance also. This kind of application can be operated in many scenarios like connected, sometimes connected, and disconnected. There are a number of tools available for supporting user interface which is a crucial aspect of any application, like Windows Forms, Microsoft Office Business Application, and Windows Presentation Foundation. These environments and tools help to developers to fast and in a simple way development of applications which works on the technology of rich client.

These technologies are mostly used to develop web based application but can be used to develop stand-alone applications also. Sometimes these applications depend upon some services provided by other layers including logical and physical layers. The main work of these layers is to sending information to other systems, providing the data access to client, prepare backup copies, information retrieval, searching and many other activities which are required by client.

1.5.5 Integrated Development Environment (IDE)
An IDE is computer software that bundles of a source code editor, a visual editor, a compiler, an interpreter, build-automation tools and a debugger. Examples of IDEs are Visual Studio, JBuilder, Frontpage and Dreamweaver. An IDE is implemented for a specific programming language. Sometimes other tool such as a version control system and Graphical user interface builders are integrated as well to simplify the development process.

1.5.6 AWT / Swing
Swing is a graphics library for Java. It sets up on the Java AWT library, which is the Abstract Windowing Toolkit (AWT). It is also a platform-independent windowing provided for java. AWT belongs to the Java Foundation Classes which are abbreviated as JFC. Nearly all of the technologies used for a browser-based solution (JSP, HTML, CSS, Java Script) need run-time testing to see if something is working. This results in far more Edit, Compile, Run, and Test cycles. With a Swing-based solution, problem can be
found earlier either by Eclipse before compilation or during compilation—resulting in fewer time being wasted chasing down bugs [7].

1.5.7 SDK
For development of a good application, a development tool is needed which provide all the facilities so that the final project or application can provide a good user experience. To facilitate the ease of development this development tool must have some features as debugging facility, providing the good interface and other utilities. A Software Development Kit, or SDK for short, is a development tools which provides all these features like allowing a software engineer to create applications based on some certain software package or software framework.

This SDK or development tool can be easily received by the software developer by the target system developer, or it can be downloaded directly using Internet. A number of SDKs can be downloaded free with attached licenses to encourage developers to use the system or language.

1.5.8 Use case
A use case describes a firm behavior that is expected of a system. All use cases together describe the functionality of the whole system. Use cases can be recognized using dissimilar methodologies, as by collecting customer wishes or by analyzing the conceptual formulation. The textual representation of a use case consists of a table that holds information about the scope, level and other use case exact attributes. The description of the interaction between the system and the main actor accounts for the main part [8].

1.5.9 Eclipse
There are a number of development environment available for multiple language. Eclipse is one of them which provide a workspace, framework and an extensible plug-in system for developing an efficient plug-in [9]. It is written mostly in Java. It can be used to build up applications in Java and many other programming languages such as Ada, C, C++, COBOL, FORTRAN, Haskell, Perl, PHP, Python, R, Ruby, and Ruby on Rails, Groovy,
and Scheme. Eclipse is an extensible, open source IDE (integrated development environment) which provides a lot of features for ease of development. The project for eclipse was stated in November 2001. At that time IBM worked with the Websphere Studio Workbench and as the result Eclipse Consortium formed. In this development environment the tools for different languages are

Java - Java development tools (JDT)
C/C++ - CDT
PHP – PDT

The Foundation of Eclipse is a non-profit, member supported corporation. This foundation supports both a network of corresponding services and products as well as Open Source community.

1.5.10 OpenGL
OpenGL is abbreviation of Open Graphics Library. This is a multi-platform, cross-language Application Programming Interface which is used for rendering 2D and 3D computer graphics. OpenGL is the most extensively adopted 2D and 3D graphics application programming interface in the industry. This feature is very useful to bring a number of applications to work with wide range of computer platforms. OpenGL describes all the features of the latest graphics hardware. It is an operating-system and window-system independent as well as network-transparent. As now days there is a big market for Computer Aided Design, energy, entertainment, medical, and virtual reality, game development, manufacturing, content creation, OpenGL helps developers by providing the high-performance, visually compelling graphics.

1.5.11 Plug-in Architecture
The Eclipse Platform, a development framework donated to the open source community by IBM. This architecture allows the developer to build tools that can work properly with the environment as well as other tools. The main feature of eclipse is that it can be used as extensible integrated development environment as it is an open source platform. The basic idea behind the perfect integration of tools with Eclipse is only one that is plug-in. surprisingly everything in Eclipse is a plug-in except a runtime kernel. As in
development with using Eclipse it uses the same technique for creating the plug-ins, all features which is required by plug-in remains equal. So there will not be any problem of creating, maintaining, extending the plug-in if it is done by using the framework of eclipse.

A plug-in must have an extension point which is required to plug into in order to utility. For this purpose, there are two indispensable plug-ins are provided by the Eclipse Platform, the Workbench and the Workspace plug-ins. Most of the plug-ins uses this plug-ins to provide the extension points.

1.6 THE ECLIPSE PROJECT - ECLIPSE.ORG

The first version of Eclipse™ was released in November 2001. It was announced as a 40 million dollar present to the open source community. One can read the statement that “Eclipse is a kind of universal tool platform - an open extensible IDE for anything and nothing in particular” on the website. But we must say here that Eclipse provides much more features than only an Integrated Development Environment. Due to its plug-in architecture, it can change to nearly every application one can imagine. The Java Eclipse-IDE is only one excellent example of an Eclipse plug-in. But it is even more: with JFace and the Standard Widget Toolkit, which is the core of the Eclipse platform, an alternative to the Abstract Window Toolkit (AWT) and Swing is provided. Standard widget toolkit libraries allow the creation of native looking user interfaces by providing small interfaces to the operating system dependent libraries. A Java application, which is based on standard widget toolkit, responds like a native application. Moreover, Eclipse provides features like resource management, editors, views, task management, a help system and wizards which can be used in own applications.

1.7 THE RICH CLIENT PLATFORM

In addition to the small Platform Runtime, the Eclipse platform consists of the team, workbench, workspace and help components. Other tools can be plugged into the stand by extending the platform at certain extension-points. The Eclipse Software Development Kit (SDK) already includes two tools: the Java Development Tooling (JDT), which implements a set of plug-ins for Plug-in Developer Environment and Java development,
which can be used to create and manage plug-ins for the Eclipse Platform. These tools are the basis for the use of the Eclipse Platform as IDE, but also demonstrate how new tools can be integrated into the platform.

1.8 DOCUMENT STRUCTURE

The thesis is divided into eight parts:

The first chapter Introduction describes the current development trends, the problems with Java desktop applications and the resulting need for Rich Clients. Moreover it defines the goals of this work.

The second chapter Review of Literature copes with the terminology that is used in this work, the Eclipse project and its history. Furthermore the main characteristics of the Rich Client Platform and Earth Navigator are illustrated. Beside this, it describes Rich Client Platform Architecture and all its features. To facilitate the comprehension of the concept Literature Review is also provided here.

The third chapter Materials & Methods introduces the Intro Plug-in. It is used to demonstrate how the Model of Eclipse Plug-in works, and how one can make plug in. This chapter also put some light on OSGi. It introduces OpenGL with its basic features and why we need it. Design describes all the code and design of Main project Earth Navigator, with Google Geocoding API.

The fourth chapter Observations, Result & Discussion shows the strengths and weaknesses of Plug-in Architecture which are observed in development of a project i.e. Earth Navigator on Rich Client Platform. It also shows the comparison of Linux and Windows Platform, standard widget toolkit and swing. Problems with Eclipse, with Discussion.

Finally the fifth chapter Conclusion summarizes the main concepts of this thesis and gives an outlook on future extensibility.

The sixth chapter Recommendations explains the features of Eclipse and Rich Client platform describes why it is recommended to use Rich Clients Applications.
The seventh chapter gives the complete Bibliography. A bibliography is a list of the resources (books, magazines, videos, computer programs) which is used in assignment or project. It is important to include where you have found the information.

The eighth chapter Publications gives detailed information about my publications in Journals and Conferences.