Spring 1998

Application of the internet technology and client/ server paradigm for the implementation of REPI

Chetankumar G. Patel
New Jersey Institute of Technology

ABSTRACT

APPLICATION OF
THE INTERNET TECHNOLOGY AND CLIENT/SERVER PARADIGM
FOR THE IMPLEMENTATION OF
REPI

by
Patel Chetankumar G.

There are many problems associated with Requirements Engineering such as defining the system scope, developing understanding among the communities involved in the system to be built, volatility of requirements etc. These problems may lead to poor requirements and therefore cancellation of the system development, or else the development of a system that is unsatisfactory, has high maintenance cost or is unacceptable. By improving Requirements Elicitation, the Requirements Engineering can be improved, leading to a better requirements specification and eventually a better product.

Requirements Elicitation requires effective communication among the team members, as communication is the key factor. Easing communications between stakeholders and developers makes the process of Requirements Elicitation easier. REPI guides team members through the elicitation phase using the SEI's framework. REPI forces stakeholders to explicitly describe the requirements resulting in reduced chances of misunderstood requirements, leading to better requirements specification.

APPLICATION OF
THE INTERNET TECHNOLOGY AND CLIENT/SERVER PARADIGM
FOR THE IMPLEMENTATION OF
REPI

by
Patel Chetankumar G.

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer and Information Science

Department of Computer and Information Science

May 1998

APPROVAL PAGE

APPLICATION OF
THE INTERNET TECHNOLOGY AND CLIENT/SERVER PARADIGM
FOR THE IMPLEMENTATION OF
REPI

Patel Chetankumar G.

_4/27/98_

Dr. Murat M. Tanik, Thesis Advisor   Date
Department of Computer and Information Science
New Jersey Institute of Technology

_4-27-98_

Dr. Franz Kurfess, Committee Member   Date
Department of Computer and Information Science
New Jersey Institute of Technology

_4/27/98_

Dr. Donald H. Sebastian, Committee Member   Date
Department of Industrial and Manufacturing Engineering
New Jersey Institute of Technology

_4/27/98_

Dr. Ali H. Dogru, Committee Member   Date
Department of Computer and Information Science
New Jersey Institute of Technology

APRIL 27, 1998

Dr. Ajaz R. Rana, Committee Member   Date
Department of Computer and Information Science
New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Patel Chetankumar G.

Degree: Master of Science

Date: May 1998

Undergraduate and Graduate Education:

- Master of Science in Computer and Information Science,
  New Jersey Institute of Technology, Newark, NJ, 1998

- Bachelor of Science in Electrical Engineering,
  BVM Engineering College, Vallbh Vidyanagar, INDIA, 1994

Major: Computer and Information Science

To
the Lotus Feet of
Lord Swaminarayan
and
My Spiritual Guru Pramukh Swami Maharaj

ACKNOWLEDGEMENT

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

CHAPTER 1

JAVA - THE INTERNET TECHNOLOGY

1.1 Introduction

The electronic *tsunami* called the Internet swept ahead unstoppably throughout 1997 and looks to advance indefinitely. The Internet is a global network of networks interconnecting very large number of heterogeneous and autonomous computer systems worldwide using a simple standard common addressing system and communications protocols. Many networks are part of the Internet, including federal, regional, educational, social and some foreign networks [RICHMOND 97]. The Federal Networking Council (FNC), in consultation with members of the Internet and intellectual property rights communities, came up with the following definition of the Internet:

> "Internet" refers to the global information system that -- (i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ones; (ii) is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ones, and/or other IP-compatible protocols; and (iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein [LEINER 97].
> Some of the "founding fathers" of the Internet say "the Internet is at once a world-

wide broadcasting capability, a mechanism for information dissemination, and a medium for collaboration and interaction between individuals and their computers without regard for geographic location" [LEINER 97]. Intranet can be thought of as a local area network based on Internet technology. It uses the same technologies used in the Internet; but its servers are limited to company's networks; in effect, a private Internet for a company. Intranets are easier and cheaper than LAN. The use of open Internet standards provides

Intranet users with more choices, easier setup and maintenance, lower cost of application deployment and management, cross platform access to information and applications, easier access to information, and lower training costs [ORACLE 96]. Extranets are Intranets, of various companies, joined together, for mutual integration and collaboration among close business partners. At the application level, the underlying technologies of all these networks are the same. This thesis, For the REPI web site implementation, treats Internet, Intranets and Extranets as the same.

With the growth of the Internet, the most likely question is "When will the Internet do this . . .?" and not "Can the Internet do this . . .?" [DEEPAK 98] Companies and people are using the Internet for many things such as online shopping, chatting, uploading and downloading of newly developed software, gathering ideas and specifications for new software and in near future for live conferences and lectures. The Internet can be used for all types of communication needs, between and among developers, clients and end-users, during the application development process. Electronic commerce over the Internet is flourishing. The on-line brokering division of San Francisco's Charles Schwab & Co. boasts 900,000 active accounts, while the Internet only bookseller Amazon.com reports second-quarter sales of US $27.9 million- a startling 1168 percentage higher than for the same period a year ago. A study media a market analysis firm Cowles/Simba Information Inc., of Stanford, Conn., suggested that the on-line services market, propelled by vigorous growth in the brokerage, marketing, and news segments, will reach $37.5 billion in 2001. "Unlike two years ago, leading companies now believe developing intranet delivered products is absolutely critical for their long-term survival" said Ben de la Cruz [Larry 98]. In short, The Internet is

becoming a necessity for the 21st century human society. In this thesis, the Internet is viewed as a large-scale, open, integrated environment for distributed application development. Along with the detail description of the Java, this chapter summarizes the Internet technologies.

1.2 World Wide Web

The World Wide Web is a huge collection of hypertext documents linked together without any overall organization. The WorldWide Web is composed of thousands of virtual transactions taking place per hour throughout the world, creating a web of information flow. Many people and organizations use web sites for many purposes. Large and small companies use their web sites as a virtual public-relation office or as a virtual product showcases. Individual person and other organizations use web pages to publish their interests and information. Organizations view the Internet as a virtual office and new platform for business applications that can be accessed from variety of locations and/or platforms.

Figure 1.1 shows an overview of the World Wide Web's technology. Web client uses the HyperText Transport Protocol (HTTP) to transmit a request to the web server across the Internet. The web server, of course using HTTP, either returns a static HyperText Markup Language (HTML) document from its local disk or uses Common Gateway Interface (CGI) scripts to communicate with external applications such as database servers.
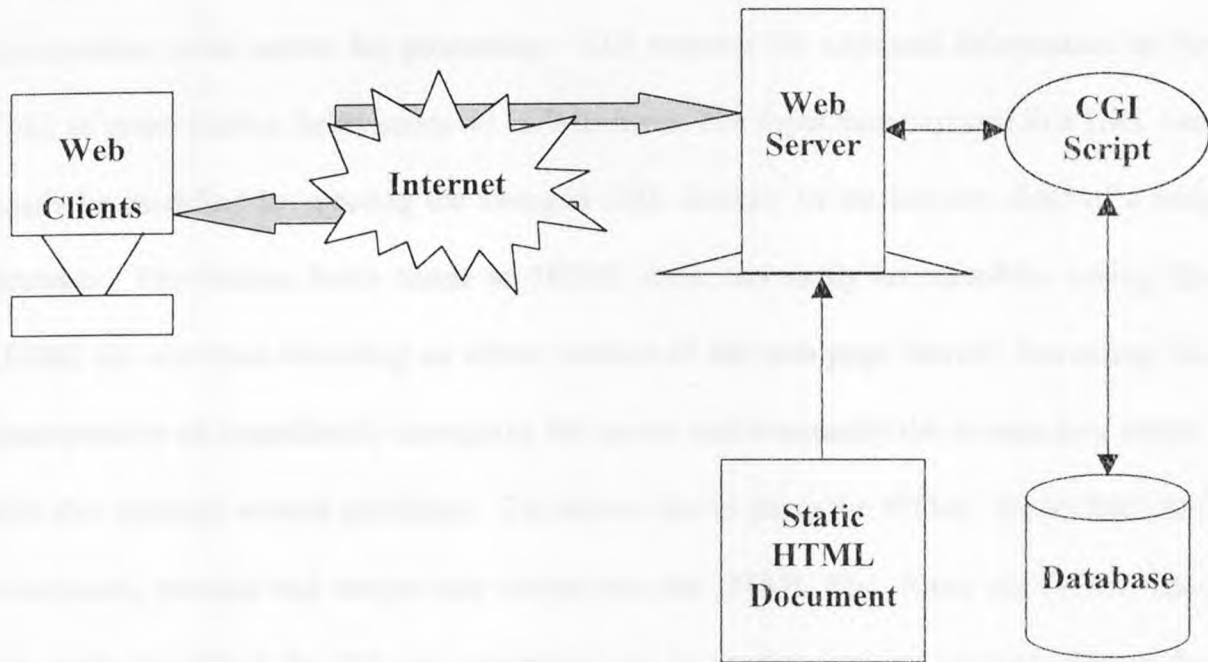
Figure 1.1: Simplified View of the World Wide Web. Source: "Web Access to the Core Business Infrastructure." Report MCS-0260-1. Butterworth, Paul. Forté Software Inc. September 1996

1.3 The Java Language

Today, Java is known as the universal language of the Internet. Static HTML does not provide enough client side intelligence. Powerful applications on the web require a powerful user interface on the client side. Even dynamic HTML has some limitations mainly the security everybody on the Internet worrying about. Client side programming can be of two types: scripts such as JavaScript, VBScript etc. and compiled programs such as C, C++, Delphi, Java, and Visual Basic can be used to create various types of client side programs called as plug-ins, controls, or applets. CGI provides a method for creating dynamic web pages based on input provided by the user and output provided by external applications [CGI]. With the popularity of the Internet security is becoming more and more important. One of the major problems with CGI is that of security

[HERRMANN 96]. CGI has to capture information from the user and pass this information to the server for processing. CGI exposes the captured information in the URL or in the hidden fields inside an HTML form. The input data exposed in a URL can easily be modified by entering the changed URL directly in the location field of a web browser. The hidden fields inside an HTML form can easily be edited by saving the HTML file and then reloading an edited version of the web page thereby increasing the susceptibility of intentionally corrupting the server and eventually the system as a whole. SSI also imposes several problems. The server has to parse the HTML file to find SSI commands, execute and merge their output into the HTML file. Since the HTML files can easily be edited, the SSI *exec* command can be used to execute any program on the server. This imposes an obvious security risk. The second major problem is that since CGI scripts are interpreted, CGI scripts are slow which causes a performance problem and they do not scale well as the number of requests increases. Using Java the security problem that is becoming of paramount importance with the exposure of the Internet can be overcome. Also, Java applets and applications give better performance over CGI or other script. Using Just-In-Time compiler the speed of execution of Java programs can be made as fast as C++ or C programs. Thus Java has both kinds of capabilities: those of scripts and those conventional programming languages. Let us discuss in detail about the content of the Java as a language and environment.

1.3.1 History

```
The Next Stage of the Known,
Or a Completely New Paradigm?

Taiichi Sakaiya--The Knowledge-Value Revolution
```

The team of six top software developers called "Green Team" at SUN, including Naughton, set up to innovate something cool, went into a self-imposed exile, very much like the scientists on the Manhattan Project. The team discussed what they liked and didn't like about the technologies that were out on the market; and took apart countless electronic devices, such as Nintendo Game Boys, TV set-top boxes and remote controls. The reason for this free-form exploration was to find a way for the appliances to talk to each other. The team discovered early on that electronic devices such as VCR's, laser disc players, and stereos were all made with different CPU's. Thus if a manufacturer wanted to add functions or features to a TV or VCR, they were stuck because they were limited by what the hardware and its wired-in programming would allow them to do. This, coupled with the fact that the chips used by many of these devices were limited in program space, suggested a fresh approach to software programming that might be a key to enabling innovation in this product space [ANONYMOUS 1].

The team's efforts kicked off the development of a new object-oriented programming language that Gosling called Oak, after the tree outside his window. Loosely based on C++, the language was stripped down to a bare minimum in order to be compatible with the limited space the chips in hand-held devices would offer, and was designed to allow programmers to more easily support dynamic, changeable hardware. The Green team conducted extensive research into how and why people were attracted to certain video games and how they interacted with various kinds of electronic equipment.

After collecting their research data, the team developed a hand-held, remote-control-like device with a tiny visual interface. The device, dubbed "*7", featured an animated character named "Duke" who helped guide users through the easy-to-use, image rich, graphical interface remote control. Central to the design of the *7 was the conviction that the interface must be engaging and fun to use, and that the device itself must be a small, personal artifact. "Duke," created by Joe Palrang, would go on to become Java's mascot. Sun renamed Oak into Java. With Java in the hands of the Internet community, all that was needed was a way to run Java applets. "WebRunner" was renamed to HotJava browser to run Java Applets. Then, Netscape began supporting Java. Now millions are Java-ready, and Duke has never looked back [ANONYMOUS 1].

So what exactly is Java? The past few years were awful for the developers due to the growth of multiple incompatible hardware architectures such as Unix, IBM compatible PCs, Apple etc., each supporting multiple incompatible operating systems, with each platform operating with one or more incompatible graphical user interfaces. Now developers were supposed to cope with all this and make applications work in a distributed client-server environment. The growth of the Internet, the WorldWide Web, and "electronic commerce" have introduced new dimensions of complexity into the development process. Java is designed to relieve the developers from the burden of dealing with these hurdles in the context of heterogeneous and network-wide distributed environments. Paramount among these challenges is secure applications that can run on any hardware and software platform, and can be extended dynamically. The security of the system as a whole, one of the prime factors when dealing with the network is provided by Java as Java run-time system has built-in protection against viruses.

A simplified way of describing Java is to say that it is "a new programming language, with elements from C, C++ and other languages, and with libraries highly tuned for the Internet environment" [LINDEN 96]. A complicated way of describing Java is to say that it is "a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language" [SUN]. Java can be thought of as a way to make Web pages sexy incorporating stock tickers, sound or video into Web pages. It is becoming computing platform the base upon which software developers can build applications. Developers can build a variety of applications using Java such as traditional spreadsheets and word processors in addition to mission critical applications: accounting, asset management, databases, human resources and sales. Java applications, or applets, are different from other applications in that they reside on the network in centralized servers. The network delivers the applet to the system when a user requests them [ANONYMOUS 1].

From the corporations' point-of-view, Java will simplify the creation and deployment of applications thus saving money. Applications created in Java can be deployed without modification to any computing platform, thus saving the costs associated with developing software for multiple platforms. And because the applications are stored on centralized servers, there is no longer a need to have people insert disks or ship CD's to update software.

1.3.2 Java Platform

The Java Platform is represented by the runtime environment. The diversities of computer platforms such as Microsoft Windows, Macintosh, OS/2, UNIX® and NetWare® have created many problems. Software must be compiled separately to run on each platform. Being binary a machine specific language, an application that runs on one platform cannot run on another. The Java Platform is a new *software*, not hardware *platform* for delivering and running highly interactive, dynamic and secure applications on networked computer systems. The Java Platform sits on top of these other platforms, and compiles software to bytecodes, which are machine instructions for a Java virtual machine. B*ytecode*, also called *J-code,* are "architecture neutral intermediate format designed to transport code efficiently to multiple hardware and software platforms" [GOSLING 95].

The platform has built-in security, exception handling, and automatic garbage collection. Just-In-Time (JIT) compilers are used to speed up execution. From within the Java Language, developers can also write and call native methods in C, C++ or another language. The Java Language is the entry ramp to the parkway of Java Platform. Compiled Java programs run on the Java Platform. The two primary parts of the Java Platform: (1) Java Virtual Machine (JVM), and (2) Java Application Programming Interface (Java API) provide an end-user runtime environment for deploying Internet and intranet based Java applications. These components will be discussed later on in detail. In the figure 1.2, the Java Base Platform consists of Porting Interface, Java Virtual Machine, Java base and standard API. The Porting Interface lies between the Java Virtual Machine and the operating system (OS) or browser. This Porting Interface has a platform

independent part (shown above adapters) and a platform-dependent part, shown as Adapters. The OS and JavaOS provide the window, filing and network functionality. Different machines can be connected by a network, as shown in figure 1.2.
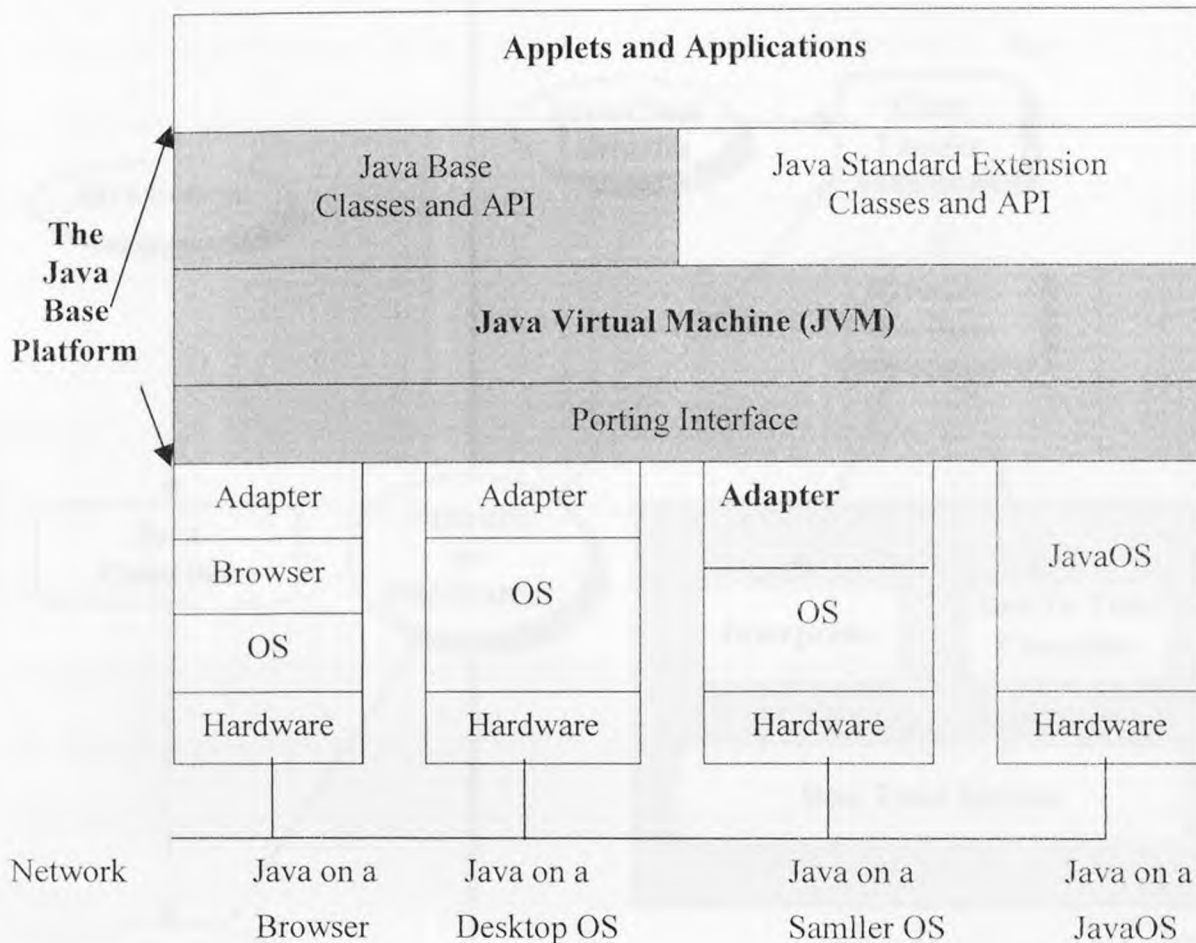
Figure 1.2: The Java Platform   Source : A White Paper, Sun Microsystems Inc. 1996

Figure 1.3 shows the Java Compile Time and Run Time environments. As shown in figure 1.3, Java Source is converted to bytecodes or jcodes by Java compiler. These bytecodes make a move through a network or file system. When the Java byte code is collected at the client machine, Java run time system adds the class libraries if required by the class loader. This code is passed through the bytecode verifier for the security verification. This code can is then passed to run time system either through interpreter or

Just-In-Time (JIT) compiler to convert from the architecture neutral code to machine

dependent code and then executed.

Compile Time   **Run Time**

Java Source

Java Class Libraries → Class Loader

Java Compiler

Network or File System

Java Bytecodes

Bytecode Verifier

Interpreter | Just In Time Compiler

Java Virtual Machine

Run Time System

Native Operating System and Hardware
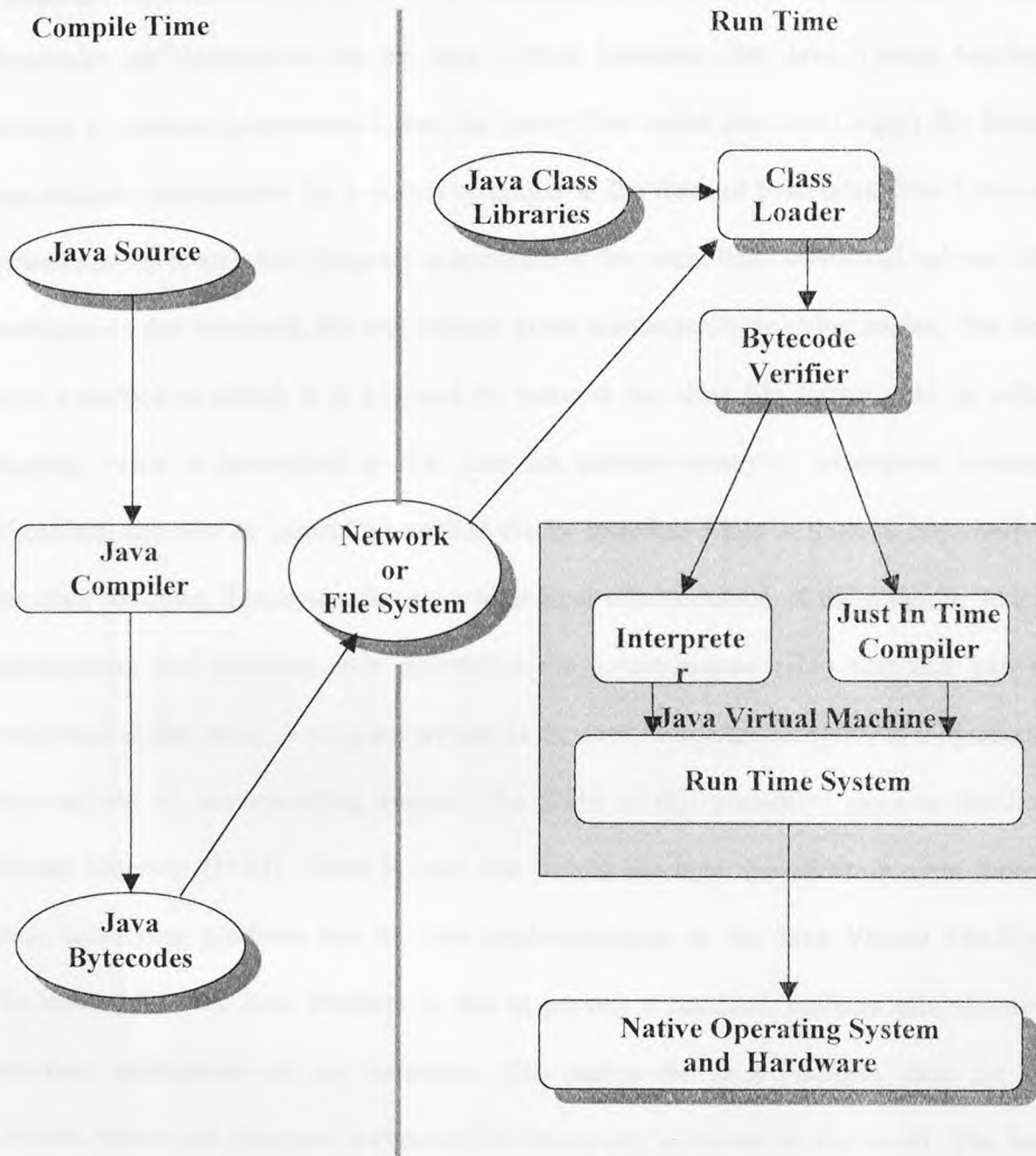
Figure 1.3: Java Environments Source: A white Paper. Douglas Kramer, Sun Microsystems Inc. 1996

1.3.2.1 Java Virtual Machine (JVM)

The Java Virtual Machine is at the core of the Java platform. The developer writes Java Language source code (.java files) and compiles it to bytecodes (.class files). These bytecodes are instructions for the Java Virtual Machine. The Java Virtual Machine defines a machine-independent format for binary files called the class (.class) file format that includes instructions for a virtual computer in the form of bytecodes. The bytecode representation of any Java program is symbolic in the sense that offsets and indexes into methods are not constants, but are, instead, given symbolically as string names. The first time a method is called, it is searched by name in the class file format, and its offset numeric value is determined at that time for quicker access at subsequent lookups. Therefore, any new or overriding method can be introduced late at runtime anywhere in the class structure. The bytecodes are a high-level representation of the program so that optimization and machine code generation via a just-in-time (JIT) compiler can be performed at that level. A program written in the Java Language compiles to a bytecodes that can run on any operating system. The credit of this portability goes to the Java Virtual Machine (JVM). There is only one virtual machine specification even though each underlying platform has its own implementation of the Java Virtual Machine. Because of this, the Java Platform is able to provide a standard, uniform programming interface applications on any hardware. This makes the Java Platform ideal for the Internet, where one program is expected to run on any computer in the world. The Java Platform is designed to provide this "Write Once, Run Anywhere" capability [KRAMER 96]. Developers compile Java-powered applications once to the Java Platform, rather than to the underlying operating system. Being JVM a "soft" computer it can be implemented

in software or hardware. In other words it is an abstract machine that is designed to be implemented on top of existing processors. Due to availability of the porting interface and adapters it is easy to port Java-powered applications to new operating systems without being completely rewritten. Garbage collection can be performed inside the JVM since it holds the stack space for the variables.

1.3.2.2 Java Application Programming Interface (Java API)

The Java API is a standard interface for applets and applications, regardless of the underlying operating system. The Java API is the essential framework for application development. The Java API framework is open and extensible. This API specifies a set of essential interfaces that developers will use to build their Java-powered applications. The API is organized by groups, or sets. Each of the API sets can be implemented as one or more packages (namespaces). Each package groups together a set of classes and interfaces that define a set of related variables, constructors, and methods. We will discuss, in brief, about various Application Programming Interfaces made available in the Java language.

The Java Base Platform is the minimum Java Platform that is able to support Java-powered applications and contains the JVM with a minimal set of API called Java Base API or Core API or Applet API. It includes all the classes in the Java packages: java.lang, java.util, java.io, java.net, java.awt, and java.applet.

The Embedded Java Platform is being designed for consumer devices with fewer resources and more specialized functionality. For example, printers, copiers, and cellular phones etc. have special constraints such as smaller memory, no display, or no

connection to a network are the target of Embedded Java Platform. The API defined for this platform is called the Java Embedded API and is the smallest API a low-function embedded device can have and still run. The Standard Extension API, defined by JavaSoft in consultation with leading industries, is the extension to the base functionality. They can be added to but not changed in a way that calls to them would fail. Over time, Other nonstandard extension APIs can be provided by the applet, application, or underlying operating system. The AWT provides the ability for uniform interface design across different GUIs. It has several input objects such as *Button*, *Checkbox*, *Choice*, and *TextField* [JAMSA 96]. Java supports event driven programming like many other visual languages. The *action* method is used to define the code for a given GUI widget.

Java Security API have been defined to help developers easily implement secure applets and applications. This functionality includes cryptography, with digital signatures, encryption and authentication. Java Security includes an abstract layer that applications can call. That layer, in turn, makes calls to Java Security packages that implement the actual cryptography. This allows third-party developers specializing in providing cryptographic functionality to write packages for Java Security[KRAMER 96].

The Java Media API have been developed to meet the evolving needs of effective and efficient communication technologies. The Java Media API is the set of the multimedia classes that support a wide range of rich, live media on and off the Web, including audio, video, 2D, 3D, animation and telephony. Java includes support for 2-dimensional and 3-dimensional graphics programming; multimedia and telephony applications; network and systems management; electronic commerce; and encryption and authentication [VONDRAK 97].

The Java Media API is composed of several distinct components such as audio, video, 2D, 3D,animation, telephony etc. Java 2D API provides graphics and imaging capabilities beyond those available with the Java Applet API. It allows the creation of high-quality graphics including line, art, text, and images in a single model that uniformly addresses color, spatial transforms and composition. The following figure demonstrates the power of Java 2D API in the field of spatial transformation and composition.

Figure 1.4: Composition: The power of Java 2D API   Source: 2D API. White Paper, Sunsoft, Sun Microsystems Inc., 1996

Java Media Framework API handles time-critical media, such as audio and video. This framework provides a common model for timing, synchronization, and composition, which can be applied to media components to allow them to interoperate. Video API facilitates streaming and stored video sources. It defines basic data formats and control interfaces. Audio API supports sampled and synthesized audio. It includes a specification for 3D spatial audio, and accommodates both streaming and stored audio sources. MIDI API provides support for timed-event streams. It uses the Media Framework for synchronization with other activities, and for an extensibility mechanism for new

synthesizers and effects. Java Animation API supports 2D animation of sprites. It makes use of 2D interfaces for composition and the Media Framework for synchronization, composition and timing. Java Share API provides the basic abstraction for live, two-way, multi-party communication between objects over a variety of networks and transport protocols [KRAMER 96].

Java3D API provides high-performance, interactive, 3D graphics support. It supports VRML. The 3D API simplifies 3D application programming and provides access to lower level interfaces for performance. The 3D API is closely integrated with Audio, Video, MIDI and Animation areas. The key areas of application of 3D API are 3D computer games, Virtual Reality Systems, CAD systems both Web-based CAD applications and stand-alone MCAD and in future VRML, interactive web based Graphic Tools etc. The following figure shows one of the application of Java 3D API.
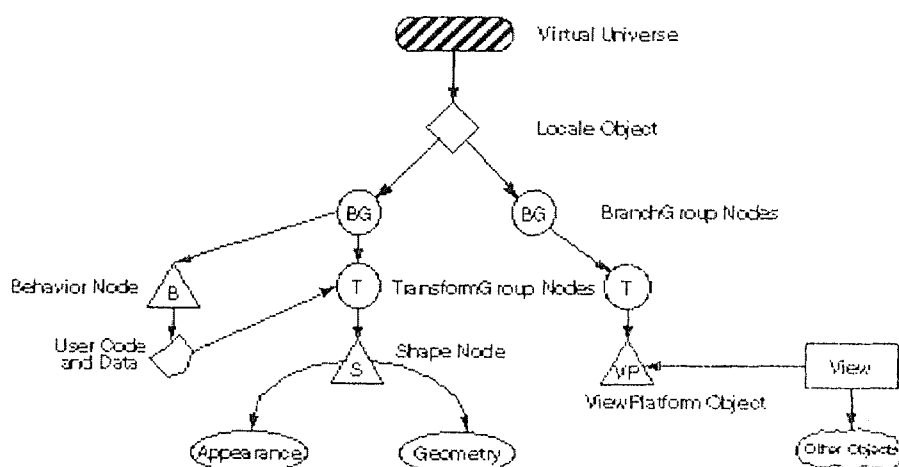
Figure 1.5: Application of Java 3D API Source: Source:3D API. White Paper, Sunsoft, Sun Microsystems Inc.

The Java Media Player is a set of high-level interfaces for reception and playback of arbitrary time-based media such as MPEG-1, MPEG-2, QuickTime, AVI, WAV, AU, MIDI, and real-time streaming audio/video. They can present multimedia data from a

various sources by providing a universal resource locator (URL). Media data can come from reliable sources such a HTTP or FILE, or a streaming source such as video-on-demand (VOD) servers or Real-Time Transport Protocol (RTP). Appendix A.3 shows the example of Java media player.

Java Telephony API unifies computer/telephony integration by providing basic functionality for control of phone calls: 1st-party call control (simple desktop phone), 3rd-party call control (phone call distribution center), teleconferencing, call transfer, caller ID, and DTMF decode/encode.

The Enterprise classes connect Java applications to enterprise information resources. Currently, there are three groups for connectivity: JDBC™ (Java Database Connectivity), Interface Definition Language, and Remote Method Invocation.

JDBC is a standard SQL database access interface that provides Java programmers with a uniform interface to a wide range of relational databases, and also provides a common base on which higher level tools and interfaces can be built. Partners such as Intersolv, Visigenic, and a dozen other database-connectivity vendors are providing JDBC drivers in the next few months for dozens of DBMSs, including Oracle, Sybase, and Informix. These database companies, and leading tool vendors, such as Symantec and Borland, have already endorsed the JDBC API and are developing products using JDBC. The JDBC API defines classes to represent constructs such as database connections, SQL statements, result sets, and database metadata. JDBC allows a Java-powered program to issue SQL statements and process the results. In conjunction with JDBC, JavaSoft has released a JDBC-ODBC bridge implementation that allows any of the dozens of existing Microsoft ODBC database drivers to operate as JDBC drivers. The

JDBC-ODBC bridge can run on the server rather than client side using a JDBC driver that translates to a DBMS-independent network protocol. JDBC allows the applications to communicate directly with the data base server but it does not allow the applets. Applets can communicate with data base server only using the client server architecture as described in the following chapter. An applet is allowed only if it is downloaded from the host the server application is running. We have used JDBC for the data base connectivity via applets.

Interface Definition Language (IDL) is a neutral way to specify an interface between an object and its client when they are on different platforms. Remote Method Invocation (RMI) lets programmers create Java objects whose methods can be invoked from another Java virtual machine. RMI is analogous to a remote procedure call (RPC). One can use Java RMI to connect to Java components or to existing components written in other languages. RMI gives a platform to expand Java into any part of the system in an incremental fashion, adding new Java servers and clients when it makes sense. Figure 1.6 shows the application of JDBC.

The Java Commerce API is useful for secure purchasing and financial management via the Internet. The initial component of the Java Commerce API is the Java Wallet, which defines and implements a client-side framework for conducting network-based commerce. Basically a Java Wallet is an empty wallet ready to hold credit cards and cash, and a blank ID card ready to be filled in with personal information. "The Java Wallet provides: Storage of personal information about the shopper (such as name, billing address, and shipping address), Payment instruments (such as credit cards, debit cards,

and electronic cash), Details of every purchase transaction (such as date and time, item descriptions, quantities, and dollar amounts), Support for two new types of signed applets, Payment cassettes, which implement a specific payment protocol, such as the Secure Electronic Transaction (SET) supported by Visa and MasterCard Service cassettes, which implement value-added services, such as budgeting and financial analysis, Extensibility to install new payment and service cassettes dynamically by downloading them from the network, Strong cryptographic services that make it possible to implement all of the facilities described above in a secure environment" [KRAMER 96].

Java Server API enables programmers the development of a whole family of Java-based Internet and intranet servers. It contains server-side class libraries for server administration, access control and dynamic resource handling on server side. The framework also encompasses the servlet APIs that are platform-independent Java-powered objects, the server side counterpart of applets. Servlets can reside locally on the server, or be downloaded to the server from the net through certain security restrictions. HTTP servlets and servlets using JDBC/ODBC that offer database connectivity are the two extremes of the application of servlet.

The Java Management API encapsulates distributed network, system, and service management components that make up the computing infrastructure. They provide guidelines for developing interfaces for configuration and troubleshooting of these elements. The components of the Java Management APIs are briefly described as follows. Admin View Module is an extension of the Java Abstract Window Toolkit (AWT) specifically designed for creating integrated management solutions. They are

used for hypertext style of navigation. Base Object Interfaces support constructing objects that represent distributed resources and services that make up the enterprise computing environment. These interfaces allow developers to define abstractions that contain distributed attributes and methods, and persistent attributes. The Managed Notification Interfaces provide the basic foundation from which more complex event-management services can be easily built. The model provides asynchronous event notification between managed objects or management applications by providing the interfaces to an implementation of a basic event-dispatching service. Managed Container Interfaces allow managed objects to be grouped so that management applications can perform actions on a single group, instead of each instance. This permits management applications to scale upwards by allowing for multiple instances to be treated as one. Container interfaces are of two types: Extensional, which are constructed programmatically through simple add and remove methods, and Intentional, which store only the query to be executed by using the Managed Data Interfaces that generate the instances for the container. Managed Data Interfaces support mapping attributes of extensions to the Base Object Interfaces to a relational database. These interfaces are implemented on the appropriate subset of Java Database Connectivity. Managed Protocol Interfaces implement the distribution and security capabilities for extensions of the Base Object interfaces.

1.3.3 Java Applets and Applications

A Developer can create two different kinds of Java programs: Applets and Applications The following sub sections describe about Java applets and applications. the similarities and differences between these two Java programs.

1.3.3.1 Java Applications

Like any other applications, Java applications can read and write to the local file system and make network connections to any computer on the network, invoke native code by invoking externally linked functions which written in any language, such as C or C++. Java interpreter is used to run the Java application. Java applications also require the Java Platform to run; but this platform support can come in any way, including as a separate program, embedded as part of the operating system, or as part of the Java application itself [KRAMER 96].

```
public class Application
  {
  public static void main (String args[])
    {
     system.out.println ("Hello World !");
    } /* main */
  } /* Application */
```

Figure 1.6: Java Application Example

In the above example Java writes "Hello World" on the standard output screen. Like C and C++ the control starts from the function main which has to be instantiated only once. Unlike C or C++ the command line argument list does not include the application name.

1.3.3.2 Java Applets

Applets are Java programs that are not standalone and require either a browser or an appletviewer to run [PATRICK 97]. Java applets provides a developer with the flexibility to develop a more sophisticated user interface. Java applets provides the full range of event-driven pop-up windows and graphical user interface widgets [VONDRAK 97]. Since applets can be downloaded from anywhere and run on the local machine. Applet is like a "killing beauty" that can cause an irreparable damage to the host it is running security is very important. The <applet> tag is embedded in an HTML page and name of the applet to be run.

```
improt java.awt.*;
import java.applet.*;

public class Myapplet extends Applet{

    public void paint (Graphics g) {

        g.drawstring ("Hello World", 40,40);
    }
}
```

Figure 1.7: Java Applet Example, Part 1 of 2

When that HTML page is accessed by a user, either over the Internet or corporate intranet, the <applet> tag causes the bytecode files to be downloaded over the network from the server to the client's browser in the Java Platform. At the client end, the bytecodes are loaded into memory and then verified for security before they enter the Virtual Machine. In the Virtual Machine, the bytecodes are interpreted by the Interpreter. Any classes from the Java Class Libraries (API) are dynamically loaded as needed by the

applet. Since applets are required to be downloaded, they usually are kept small or modular but there is no restriction on their sizes. Figure 1.7 shows the example of Java applet, while figure 1.8 shows the HTML file that contains Myapplet applet.

```
<HTML>
<TITLE> Here is my sample Applet </TITLE>

<BODY>
The Myapplet applet will appear below in the Java Enabled
Browser.

<applet code= "Myapplet.class" width=200 Height=100>
</applet>

</BODY>
</HTML>
```

Figure 1.8: Java Applet Example, Part 2 of 2

Applet has to be derived from java.applet.Applet class or its ancestor while Application does not require to be derived from any Java class [MARY 96]. Java applications also start differently from Java applets. In Application the *main* method is the entry point and is invoked by Java interpreter while *init* method is invoked by the browser when an applet is downloaded over the network. Applications are stand-alone Java programs that do not require a browser to run and have no built-in mechanism to be downloaded. In other way applications are like programs in other languages such as C++, C etc. "They can perform traditional desktop works such as that done with a word processor, spreadsheet or graphics application" [KRAMER 96]. While an application can communicate directly with a database server using JDBC (Java Data Base Connectivity), an emerging standard to establish connection between Java-powered applications and data base server, about which will discuss later in this chapter; an applet is not allowed by JDBC to communicate with data base directly for security reasons.

However, using client-server methodology a connectivity can be established between these two end-points. We will discuss about the client-server architecture and its capabilities in the following chapter.

While applets and applications have some differences, for the most part they have the same access to a wide range capabilities. For example, both an applet and an application can access data from local host, do data processing, and store the results back to that host. However, an applet requires a network connection to work, while an application does not. "Applications have greater freedom in that they have full access to system services. For example an application, unlike an applet, can have normal read and write access to files on any disk. Since an applet can potentially be downloaded from an untrusted Web page, it is restricted from having read or write access to any file system except the server from which it came. This constraint will be relaxed when applets can be marked with digital signatures, allowing the end-user to be assured that it has been downloaded and unaltered from a trusted source" [KRAMER 96].

1.3.4 Pros and Cons of Java Platform

This section describes in brief about the advantages and disadvantages of the Java platform as a whole. Java provides benefits to the three types of communities End-users, Developers, and Support and administrative people. End-Users: The Java Platform provides live, interactive content on the Internet. Due to portability of the applications users have been made free from the monopoly of operating systems. Smaller, less expensive and dedicated systems can be easily made available for custom applications. Developers: With a comprehensive set of APIs, Developers can create "Write Once, Run

Anywhere," applications that provide tremendous marketing leverage over other languages. The ability to "Write Once, Run Anywhere" is one of the major reasons for some developers to turn to the Java Language as an alternative to C or C++. The feature of the Java language to allow the developer to use shared and reusable objects reduces the cost by permitting the developer to concentrate on developing only what is new. Administratives and Supports: Since Java applications can be kept in central repository version control and upgrades are simplified. In multivendor, multiplatform environments, the number of platforms to support is virtually reduced to one. With the network computers, data can be managed centrally while data processing is done locally. Companies with large intranets can run Java applications on all their existing machines without upgrading to the latest memory-consuming operating system. By providing corporate data in a format readable by Java-powered applications, corporations give users the platform-neutral access to the data they need using the Internet and thereby can reduce time spent on order-entry by having customers fill in order-entry forms through the Internet.

Though much claimed Java is not 100% portable. Java uses the AWT to provide an abstract interface into the native window systems, such as X/Motif on UNIX, Windows on Microsoft Windows, and Toolbox on Macintosh. For Java applets and applications to be completely cross-platform, all its dependencies and libraries, including access to third part software, have to be cross-platform. The interface between the Java Virtual Machine and the native operating system or the windowing environment will always be implementation and platform dependent [SHIFFMAN 97].

1.3.5 Deployment of Java Platform

There is a great momentum towards deploying the Java Platform in three stages from browsers, to desktop, workstation and network operating systems, and finally to embedded devices. Currently, the Java Base Platform is embedded in most widely used Internet browsers, Netscape Navigator™, Microsoft Internet Explorer™ and HotJava™. The Java Base Platform is also embedded in all leading desktop, workstation, and network operating systems, e.g. Microsoft Windows, Macintosh, OS/2, and UNIX, see Figure 1.3.

The Java Platform is emerging as the platform for all network- and Web-based computing. With the JavaChip™ family of integrated circuits, the platform will be made ubiquitous in various consumer and industrial embedded devices such as dedicated Cell-Phones, Network Computers, printers and copiers. Following is the brief discussion about the JavaChip family that deploys the Java platform.

1.3.5.1 JavaChip™ Family

JavaSoft is, with Sun Microelectronics™, developing the picoJava™, microJava™, and UltraJava™ family of microprocessors. The picoJava is actually a standard specification for the design of a microprocessor that supports the Java Virtual Machine; to license chip manufacturers. This design is a new architecture that is not SPARC-based-it is optimized for the unique demands of Java, such as multithreading and garbage collection. The microJava and UltraJava are actual chips being developed by Sun Microelectronics based on the picoJava design. These chips have the Java Virtual Machine and Java Embedded API implemented in silicon, and vary in their application-specific I/O, memory,

communications and control functions. The JavaOS can run in RAM on JavaChip. The JavaChip family enables the Java Virtual Machine to run in the most efficient, cost-effective manner, bringing high performance to dedicated Java-powered devices, such as Network Computers. JavaOS™ is an operating system that implements the Java Base Platform for running Java-powered applets and applications. It implements the Java Virtual Machine, Java Embedded API, and the underlying functionality for windowing, networking and file system. JavaOS is designed for Network Computers, consumer devices, and network devices for embedded applications, such as printers, copiers and industrial controllers. These devices will have instant turn-on, no installation setup, no system administration, and, when on a network, can be automatically upgraded. JavaOS will be widely ported to a range of microprocessors, including the JavaChip family. When JavaOS runs on a JavaChip, the microprocessor's silicon Java Virtual Machine is used.

1.3.6 A Word about Java Language

Java has proven itself ideal for developing secure, distributed, network-based end-user applications in environments ranging from network-embedded devices to the World-Wide Web and the desktop. Scripting languages are portable but slow. Compiled languages are non-portable but fast. Java is both portable and provides performance that is comparable with compiled languages when used with just in time compilers. The Java Compiler generates bytecodes from Java source code that are executed on the Java Platform. It is object-oriented (with no multiple inheritance), statically and strictly type-checked, multithreaded, dynamically linked, and has automatic garbage collection

mechanism for memory management. Its syntax is similar to C and C++, so it is quite easy to pick up for C++ / C programmers. By removing programmer control over pointers it has eliminated entire classes of programming errors. "Java has the standard primitive data types of C++, but removes the *struct* and *union* data types and adds the *boolean* data type. Java, as part of its effort to support internationalization, uses the Unicode character set for the *char* data type" [DEEPAK 98]. Automatic garbage collection, part of the Java run-time system, also increases a Java application's ease of development and reliability. Besides C/C++, Java borrows from other languages such as Eiffel, SmallTalk, Objective C, and Cedar/Mesa [GOSLING 95].

Java encourages software reusability. The virtue of Inheritance reduces the redundancy. Less redundancy facilitates developers to understand more easily someone else's code. As shown in appendix A.1 classes Square and Circle have been derived from class Coordinates and hence the developers need not redefine the methods in these classes defined inside their parent class. This inheritance can theoretically be extended to infinite level. In fact, the applet itself has to be derived from Applet class defined in the java.applet package. In our example, Inheritance class uses the drawstring method in order to display string defined in one of its forefathers. This capability of inheritance relieves Java programmers from re-inventing the wheel and gives them opportunity to develop something novel. Also, Java unlike C++ does not support operator overloading. This further reduces redundancy. However, like C++ Java supports function overloading. Java has elements of concurrency from Mesa, exceptions from Modula-3, dynamic linking and automatic storage management from Lisp, interface definitions from Objective C, and ordinary statements from C/C++ [LINDEN 96]. Interfaces were

introduced to Java to enhance Java's single-inheritance model. The designers of the Java created an elegant way through interface in order to eliminate difficulties programmers and compilers suffer due to multiple inheritance and overcome some of the drawbacks of the single inheritance [GOSLING 95]. Due to interface developers can take enjoyment of multiple inheritance and polymorphism like C++. For example, as shown in A.1, the interface Shape serves as an abstract class. A Circle class implements the interface Shape. Hence a class derived from class Circle can have inheritance of interface Shape and class Coordinates. It provides three different kinds of programming symbolic, numeric, and systems [KRAMER 96], it is object-oriented, has dynamic linking, and has a class hierarchy with single inheritance. For numeric programming, the Java Language has platform-independent data types, array bounds-checking which is not in-built in C++, and well-defined IEEE arithmetic which provides good grounding for writing stable numerical algorithms that give repeatable results. For systems programming, expressions, statements, and operators in the Java Language are in most cases the same as in the C language. It has strong data typing, automatic garbage collection, array bounds checking, lack of automatic type coercion, and has no notion of pointer. These safeguards encourages catching bugs during development before the software is released. This is quite fruitful particularly in this era of the Internet where rapid deployment of software is critical. Java supports exception handling, which is also shown in Appendix A.3. Exceptions are unexpected events that can be caught by using the *try* statement and the *catch* statement. When any Java statement executed within the *try* block results in an error, an exception is thrown. Different *catch* blocks, containing the exception handler code, can be created to catch different exceptions. The *throw* statement can be used to

throw user defined or system defined exceptions. Appendix A.3 shows the use of Java exception handling mechanism. As can be seen, Java traps the error earlier at the moment error occurs throwing respective exception. This is a definite advantage over languages like C where error is recognized when there is no way to come back. For example, if the proper URL is not found for the given file, Java throws an IOException via try block and catches that exception via catch block , thus helps the developer trap error earlier.

The in-built multithreading with synchronization mechanism helps avoid concurrency problems. Due to this capability Java can be easily used in development of systems where same data need to be accessed from multiple client or servers. Appendix A.2 shows the thread example where in two classes Consumer and Produces need to be synchronized. The condition is that a producer can not produce an item until a consumer consumes it and consumer can not consume an item until a producer produces it. The Producer class that has been derived from Thread class implements two methods need to be synchronized. The *produce()* method produces data and stores into an array. The *consume()* method tries to consume items from that array. The third class Consumer, extended from Thread implements *run()* method that invokes the consume() method of the Producer class infinite times to consume the item producer has produced. Thus the serialization is easily obtained using the threads in Java.

Java has extensive set of API to support multimedia on the Internet. As shown in Appendix A.3, the applet accommodates media Player to play video, Visual and Control components to start and stop the playing of the media. The *init()* method sets up the applet environment. The applet reads file name from the HTML page and creates URL of

the applet. It also creates the instance of player. When this applet is started, it immediately begins to play the media clip. When the end of media is reached, the clip replays from the beginning. The *start()* method is invoked when the applet is first loaded or every time the page containing the applet is loaded. It starts video playing. The *stop()* method is invoked when a user leaves the page containing that applet. It stops the video clip and releases the resources the applet has acquired from the system. The method *controllerUpdate()* is the method is the main method that handles all the jobs. It directs the event and invokes the appropriate method.

Does Java support connectivity with the Database? Of course, JDBC API have been developed to allow Java applications to communicate with the database server. The developer who has JDBC drivers for a certain database does not need to worry about changing the code for the Java programs if a different type of database is used. Also, the JDBC is not only specification for using data sources in Java applets and applications, but it also allows developer to create and use low-level drivers to connect and talk with the data sources[PRATIK 96]. For the security reasons JDBC discourages the applets to communicate with the database server directly. However using the client server architecture, applets can be made to communicate with the data base. Appendix A.4 shows the Java application that fetches the data from the database via database server using JBDC API and displays the result to the standard output. Appendix B.1 shows the client server mechanism for the Java applet to communicate with the data base. Client uses GUI to allow the user to query the database and send the request using network IO mechanism to the application server, which runs on the host the applet is originated from. The server accepts the request from the client, parses that command and invokes the

appropriate method and passes the query to the database server. Database server provides the appropriate results from the database and passes the result to the application server and application server in turn, returns the results to the client using socket mechanism. The client then displays the results in the text area.

Figure 1.9 shows the comparison of Java language with the other languages. Languages at the level of the Shells and TCL are fully interpreted high-level languages. These languages are suitable for very fast prototyping Scripting languages are also highly portable. Their primary drawback is performance; they are generally much slower than either native machine code or interpreted bytecodes. At the intermediate level comes Perl that share many characteristics in common with Java such as robustness, dynamic behavior, architecture neutrality, and so on. At the lowest level are compiled languages such as C and C++, in which one can develop large-scale programming projects that will deliver high performance. The Drawback is the high cost of debugging unreliable memory management systems and the use of multithreading capabilities that are difficult to implement and use. And of course when you use C++, you have the perennial fragile superclass issue. Last but definitely not least, the binary distribution problem of compiled code becomes unmanageable in the context of heterogeneous platforms all over the Internet. The Java language has adopted middle way between very high-level and portable but slow scripting languages and very low level and fast but non-portable and unreliable compiled languages.

	Java	SmallTalk	TCL	Perl	Shells	C	C++
Simple	●	●	●	◑	◑	◑	○
Object Oriented	●	●	○	●	○	○	◑
Robust	●	●	●	●	●	○	○
Secure	●	◑	◑	●	◑	○	○
Interpreted	●	●	●	●	●	○	○
Dynamic	●	●	●	●	◑	○	○
Portable	●	◑	●	●	◑	◑	◑
Neutral	●	◑	◑	●	◑	○	○
Threads	●	○	○	●	○	○	○
Garbage Collection	●	●	○	○	○	○	○
Exceptions	●	●	○	●	○	○	◑
Performance	High	Medium	Low	Medium	Low	High	High

● Feature exists

◑ Feature somewhat exists

○ Feature doesn't exist

Figure 1.9: Java Language Comparison Source: "The Java Language Environment." A White Paper. Sun Microsystems, Inc. 1995.

In addition to being extremely simple to program, highly portable and architecture neutral, the Java language provides a level of performance that's entirely adequate.

However it is not suitable for the most compute-intensive applications. From the diagram above, you see that the Java language has a wealth of attributes that can be highly beneficial to a wide variety of developers. It is seen that Java, Perl, and SmallTalk are comparable programming environments that offer the richest set of capabilities for software application developers [GOSLING 95].

With the growth of multiprocessing and a decrease in processor costs, the Java Language is definitely the language of current and next generation. The language includes dynamic linking of classes at runtime. The .class format defines what binary compatibility is, which is one of the important reasons for Java to be portable. So what will the future hold for companies and their use of Java? Only time will tell, but one thing is certain: it is unlikely that letters complaining about multiple and incompatible software APIs will ever need to be sent again.

CHAPTER 2

CLIENT/SERVER PARADIGM

2.1 Introduction

Today you will hardly find any technical magazine, report, periodical or newsletter that is not overwhelmed by the praises of the client server technology: "client/server did that", "client/server does this", "client/server will do this" etc. So the first question comes in the mind of the person novice to the computer field is that what is the client/server technology? Let us understand it in a simply way. In the universe, since the inception of the human society there is an existence of the client/server system. A customer requests for something useful to a producer or provider and the producer returns the requested item or items. In this example a customer can be though of as a client and a producer or provider as a server. A student and teacher can also be considered as a client server system where in a student (client) asks the question (makes a request) and the teacher (a server) does (replies) his best to satisfy his/her thirst of knowledge. In other words, Client/Server system is nothing but a fruitful, two-way communication between two parties having some common interests.

What is the relevancy of Client/Server system with the computer? And why has it been discussed in this thesis? Client/Server is emerging as the hot field in the technological area of computer software and is competing with the IBM mainframe technology. In computer terminology, client is the piece of software that makes a request and server is that intelligent piece of software that identifies the request and returns appropriate results. From the first chapter it is well understood that the technology we used for the implementation of the REPI tool is the Internet. Client/Server technology

exists in the computer field since the inception of the Internet. In fact, World Wide Web is a classic example of the client server technology in which a browser acting as a client requests for something at the specified URL and the Web server fulfills that request. We will discuss about this in detail later in this chapter. This chapter first defines the client /server terminology then quotes few commonly used systems as the examples of the client/server methodology. It discusses in brief about the role of the middlewares in this system. This chapter also explains the client/server architecture specifically about the most widely used two-tier and three-tier client/server systems. And finally describes about the client/server application developed for the implementation of the REPI.

2.2 Client/Server Definitions

What is a client? A client is a member of a class of programs in a modern software architecture paradigm called client/server software architecture, in which all applications have two components: the client component and the server component with the following typical characteristics.

- A client is executed only when there is a need for it i.e. it is invoked on-demand.

- All the dialogues in the client/server system are always initiated by a client. Thus client is an active entity.

- Since a client initiates a dialogue it most of the time, accommodates interfaces with humans that make a client always run in foreground.

- Just like a producer or service provider can have many customers, at any given time there can be numerous instances of the client component of the application running

on different computers throughout the network, all of which are connecting to only one or more servers.

- A client usually works for one person and does one job after another in sequence, as the user commands it [ANONYMOUS 3].

What is a server? A server is a member of a class of programs in a modern software architecture paradigm called client/server software architecture, in which all applications have two components: the client component and the server component, with the following typical characteristics:

- In many cases a server has to provide numerous types of services that make a server consume many critical system resources. Due to this, a server is most often centralized, usually on a large computer capable enough to provide adequate food to cater the hunger of a server in terms of system resources.

- A server is a passive entity. It does little or nothing until it receives an explicit request from a client to do something.

- In order to provide a service to a number of clients a server usually has to run continuously waiting for requests unlike a client that is called on-demand.

- A server rarely interfaces directly with humans. Most of the time it has to deal with other applications (clients). Therefore, usually, a server is kept running behind the curtain.

- A server can be of two types: one that handles multiple requests concurrently and the other that replies in order it accepts requests. A concurrent server processes multiple requests in parallel while sequential server replies the clients one after the other [ANONYMOUS 3].

2.3 Basic Client/Server Model

Client/Server model is a concept for describing communications between computing processes that are classified as service consumer's (Clients) and service provider (servers)[UMAR 97]. Figure 2.1 represents the simple conceptual client/server model.
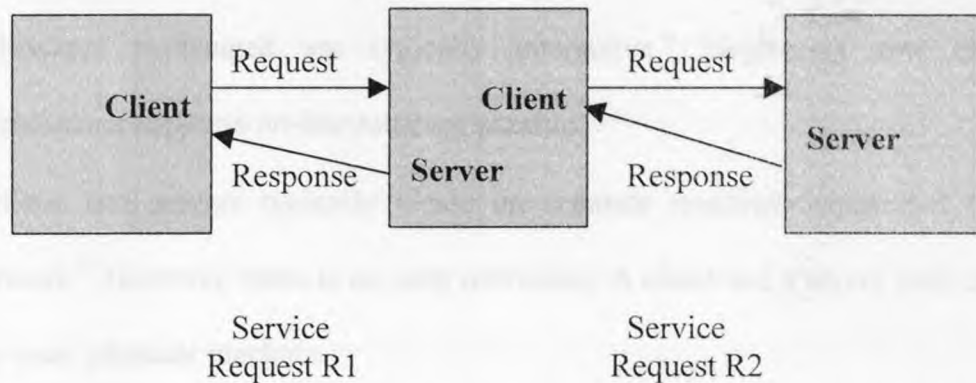
Figure 2.1: Conceptual Client/Server Model   Source: *Application (Re) engineering.* Amjad Umar, Bell Communication Research (Bellcore), Picataway, NJ. 1997

According to [UMAR 97], following are the basic features of the client/server model:

- "Clients and Servers are functional modules with well defined interfaces (i.e. they hide internal information). The functionality of a client and a sever can be a set of software modules, hardware components or a combination of both.

- "Each client/server relationship is established between two functional when one module (client) initiates a service request and the other (server) chooses to respond to the service request. "retrieve the sold quantities", "enter the quantities" are examples

of the service requests. For a given service request, clients and servers can not interchange their roles. However, a server for R1 can become a client for R2.

- "Information exchanged between clients and servers is strictly through messages (i.e. no information is exchanged through global variables)." In this extremely crucial feature of client/server model, a client sends a request as a message and a server's reply is also a message.

- "Messages exchanged are typically interactive." Neglecting few exceptions, client/server supports on-line message passing.

- "clients and servers typically reside on separate machines connected through a network." However, there is no such restriction. A client and a server both can run on the same physical machine.

2.4 Client/Server Architecture

Client/Server architecture is striking to all parties from theorists to developers to end users. It is clear that the term "client/server" implies that clients and servers are separate logical entities that work together, usually over a network, to accomplish a task. Client/server is not just a client and a server communicating across a network; but Client/server has asynchronous and synchronous messaging techniques with the assistance of middleware to communicate across a network. Here are the traditional examples of a client/server system.

As mentioned earlier, the World Wide Web works under the popular model of client/server paradigm. Figure 2.2 shows the client/server architecture of the web. There are essentially three components which together form the World Wide Web, the medium

which has brought this document to your screen. There are essentially three components which together form the World Wide Web: The Internet, Web Server and Web Client [ANONYMOUS 3].

Technically speaking, the Internet about which we have discussed in chapter 1 consists of the wires, machines and networking software that connects many heterogeneous computer systems all over the world to each other.

A Web server is an application running on a computer with the sole purpose serving documents to other computers when requested by a web client. Since the server does not perform any complicated calculations, it does a minimal amount of work and operates only when a document is requested, it put a minimal amount of workload on the computer running it. Information servers run on computers connected to the Internet and are executing computer software which deliver information as requested from users connected to the public Internet. The commonly used information servers on the Internet today have been discussed in brief in the following paragraph.

World Wide Web servers primarily deliver data that can be easily by humans such as hypertext and multimedia. Due to the underlying Hypertext Transport Protocol, they are also called as http servers. Gopher servers are the immediate predecessors of World Wide Web servers. They servers present files in distributed archives to a user as hierarchical menus. Using a gopher client, when a user selects a file from this menu. If it is a text file, it appears on your screen. If it has data in a format other than text, for example an image, the file is transferred to local computer where a user has to use a separate program to view or use it. After a file is read or downloaded, a gopher client always displays the previous menu from which a user had selected the file. FTP, or File

Transport Protocol Servers: allow FTP clients to copy files of any kind such source and executable applications, images, text etc. between the client and server machines. FTP allows a user to use commands and filenames to send to, receive from, and otherwise manage files and directories on a remote computer. When you retrieve a file from a remote computer using FTP, you have to invoke a separate program after your FTP session to view that file (a text editor, an image viewer, etc.). Graphical FTP clients for Windows or Macintosh relieve the user of the need to learn most of the FTP commands, allowing the user instead to simply drag and drop files to and from the directory as though it were a local drive on the machine. FTP is considered as the most efficient and fastest way of transferring files from one machine to another machine. NNTP, or Network News Transport Protocol servers deliver Usenet newsgroups and articles. Simple Mail Transport Protocol servers (SMTP) send and receive electronic mail messages. Archie server searches indices of FTP archives for files when given a file name or name fragment. Veronica server searches gopher menus for words or phrases. Telnet servers allow the Internet-user to login and conduct a terminal session on the remote computer running the server from anywhere on the network. These sessions usually are UNIX terminal sessions conducted via "VT100" terminal emulators. A telnet server can be used for all sorts of character-based interactive applications. Wide Area Information Servers (WAIS) search distributed pre-indexed volumes of text for words and phrases, and rank results based on a score -- how closely each document satisfied the search criterion [ANONYMOUS 3].

Web Clients or Web Browsers are the third important component of the World Wide Web. A Web client is a piece of software that interfaces with the user and requests for documents from a server when a user commands. The web browsers such as Netscape's Navigator and Microsoft's Internet Explorer have revolutionized the way the information is exchanged over the Internet. The capabilities of these browsers have relieve user from many burdens. In the past, accessing the servers mentioned in the previous paragraph required a user to use a separate program for each server type. To access a gopher server, a user had to run a gopher client program. To access an FTP server, one had to run your FTP client. To search for a file using Archie or Veronica, running either an Archie or a Veronica client was necessary. Since World Wide Web browsers are multilingual; they can communicate with all of the servers listed above and more. This relieves the user of the complexities of having to learn and run a separate client for each server they wish to use. However for the efficient use of these server respective dedicated clients are more useful. For example transferring a file over a network is much faster by using FTP clients rather than using the Web Browsers. World Wide Web browsers have powerful capabilities to support graphical user interfaces. Many of these above servers require you to learn an arcane command language or enter UNIX commands. With a Web browser, clicking of mouse or drag and drop is enough to execute the commands. The browser takes care of the underlying network communications, interfaces, and commands, to bring what a user has requested. Web browsers allow the free-form organization and cross linking and referencing of information called hypertext, hypermedia, or hyperlink using languages such as HTML.

```
   FTP                        Web      Common        Auxiliary
  Server                     Server   Gateway         Server
                                      Interface

                                                  Comput-      DataBases
   News                                            ational
  Server                                           Services

              FTP

              NNTP

  Gopher
  Server
              Gopher                            Servers at a local or
                                                Remote sites

  ────────────────── Internet Protocols ──────────────
                                                Local workstation
                                                with TCP / IP

   Client                     Web
   Helpers                    Client

  Sounds, Movies,          Hypermedia
  spread-sheets,           browser
  arbitrary applications   GUI

                      User
```
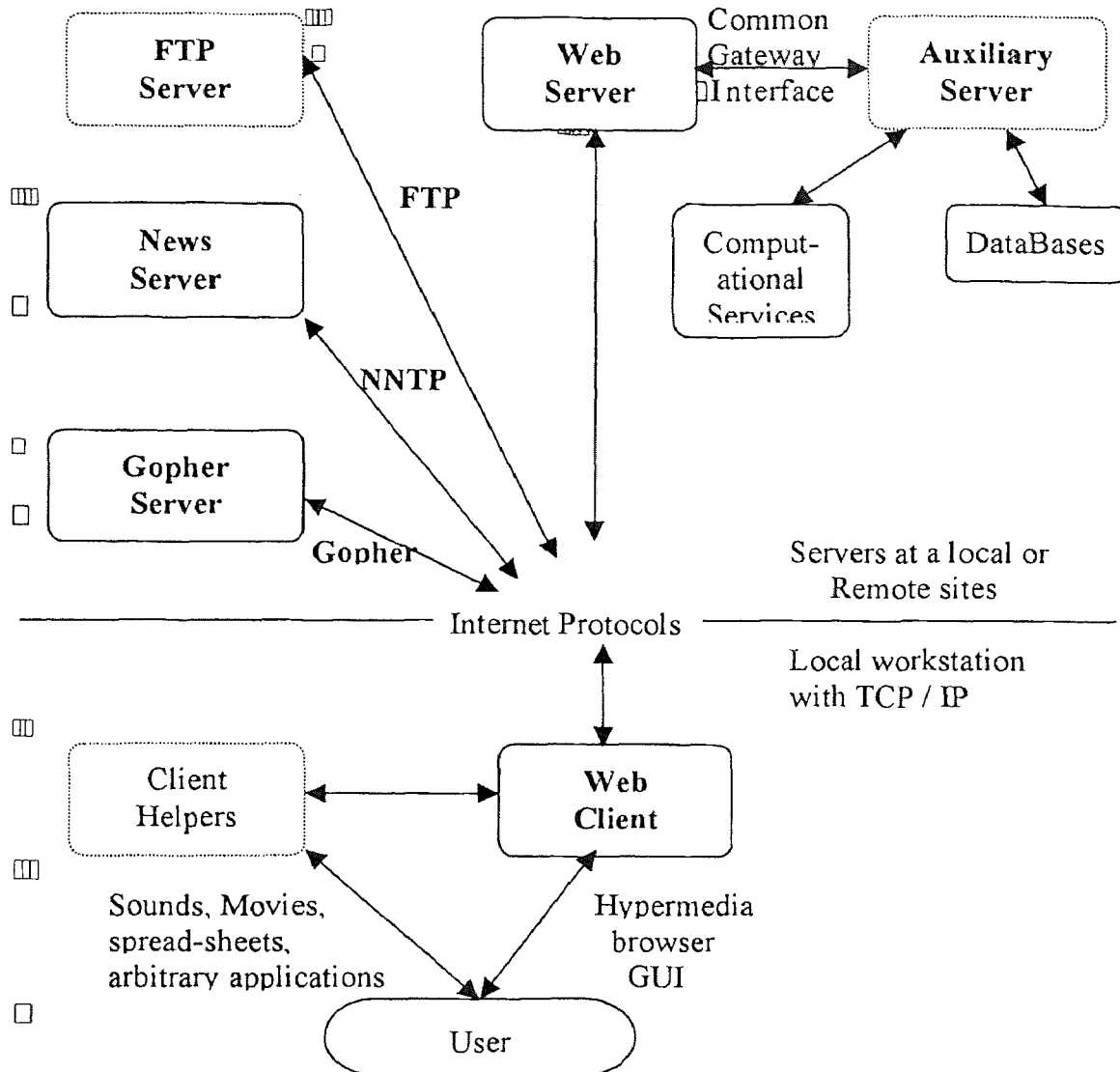
Figure 2.2: Client / Server Architecture of World Wide Web Source: "Journal of Human Computer Studies." Brian R. Gaines and Mildred L.G. Shaw. 1997

In this form of information organization, any item of information (a word, a phrase, an image) can be made to work as a "hotlink" using URL or Uniform Resource Locator, which tells a browser where to find the resource pointed to by that "hotlink" to any other item of information. Furthermore, anybody can create hotlinks in their documents to any

other publicly accessible resource. This structure creates freedom to organize and share information in myriad and novel ways, resulting in an anarchic, loosely structured web of information, art, music, data, software, literature, and just about anything else which can be represented in digital form and which some person or organization has a desire to share with the world. This is the World Wide Web.

Let us see how the process works. A user runs a Web client also known as web browser or browser, for example Netscape Communicator, and selects a piece of hypertext connected to another text - "The Client/Server Architecture". The Netscape Communicator establishes a connection with a computer specified by a network address called URL somewhere on the Internet and asks that computer's Web server for " The Client/Server Architecture". The Web server then responds by sending the text and any other media within that text (pictures, sounds, or movies) to the users terminal. The language that Web clients and servers use to communicate with each other is called the HyperText Transmission Protocol (HTTP). "All Web clients and servers must be able to speak HTTP in order to send and receive hypermedia documents. For this reason, Web servers are often called HTTP servers. The phrase "World-Wide Web" is often used to refer to the collective network of servers speaking HTTP as well as the global body of information available using the protocol"[KEVIN 93]. The World Wide Web uses HyperText Markup Language (HTML) to create and recognize hypermedia documents. Client Helpers are specialized application running locally on the same machine the browser is running and communicate with the browser through application programming interfaces or auxiliary servers running remotely on the same machine or network as the web server and communicate with it through its common gateway interface. Full power

of the local machine is available through Client Helpers. The advantage of the auxiliary server is that a single implementation for only one server platform needs to be developed and maintained. The transmission delay and limited capabilities for GUI are drawbacks of the auxiliary server system. Figure 2.3 shows in detail how client/server system works with World Wide Web. As shown, a client part consisting of Web browser that provides graphical user interfaces (GUI) through HTML documents, images, hyperlinks attached to text or icon, maps and HTML forms. A client communicates with a web server through Internet. A web server can be a direct implementation of HTTP protocol in an application server or it can be a standard HTTP server that provides a gateway to the application server. A server component consists of four parts. Encoding component decodes the incoming parameters and encodes the output of the application into HTML. Representation extracts incoming data from the output of the encoding and creates document from the output of the application. Data processing accepts input from the representation and performs appropriate processing on the data as defined by the developer of the server system. The output of the processing is sent back to the representation unit and consequently to the user via the path shown in the figure 2.3. Parameters form the fourth component of a server system. They are separate data structures or scripts that provides variants to a representation unit. A persistent storage unit provides a facilities to store the information from the client side, server side or from both sides.
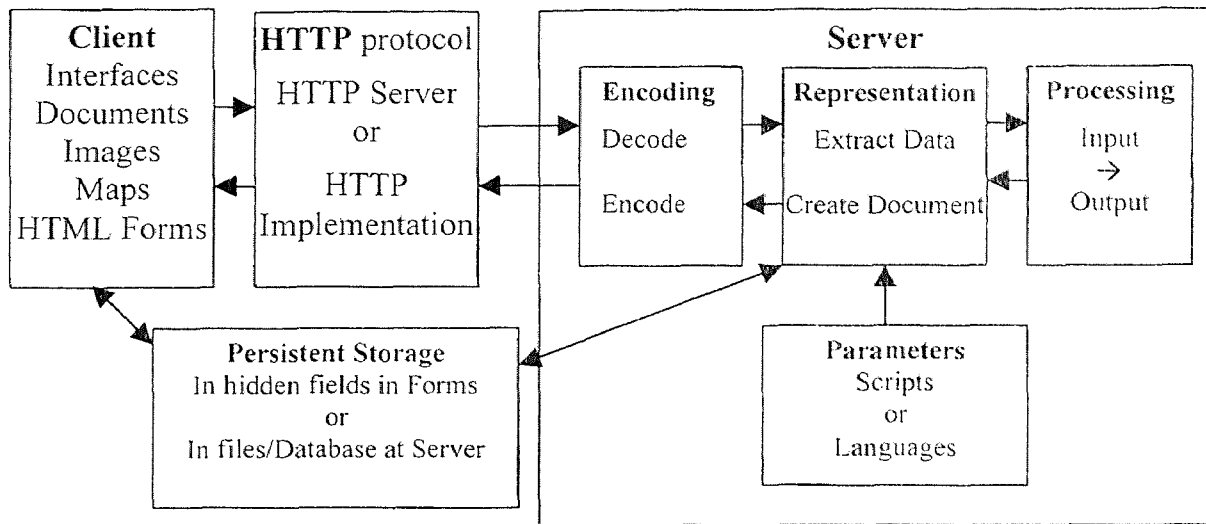
Figure 2.3: Client / Server Operation through the Web Source : "The Journal of Human Computer Studies." Brian R. Gaines and Mildred L.G. Shaw. 1997

The File Server system another example of client/server paradigm where clients request files from the File Server. This results in the entire file being sent to the client but necessitates many message exchanges across the network. The Database Server is example of the traditional client/server system where clients pass requests in terms of SQL queries to the server. The Database Server executes each SQL statement and passes the results back to the client. Open Database Connectivity (ODBC) is often used by a client to send SQL requests to the server to process. ODBC provides a standard SQL interface for sending requests to the server. Now JDBC is becoming the standard for the database connectivity. In REPI tool implementation Java applets make request to the Oracle Database Server through JDBC.

2.4.1 Client/Server - A Special Case of Distributed Computing
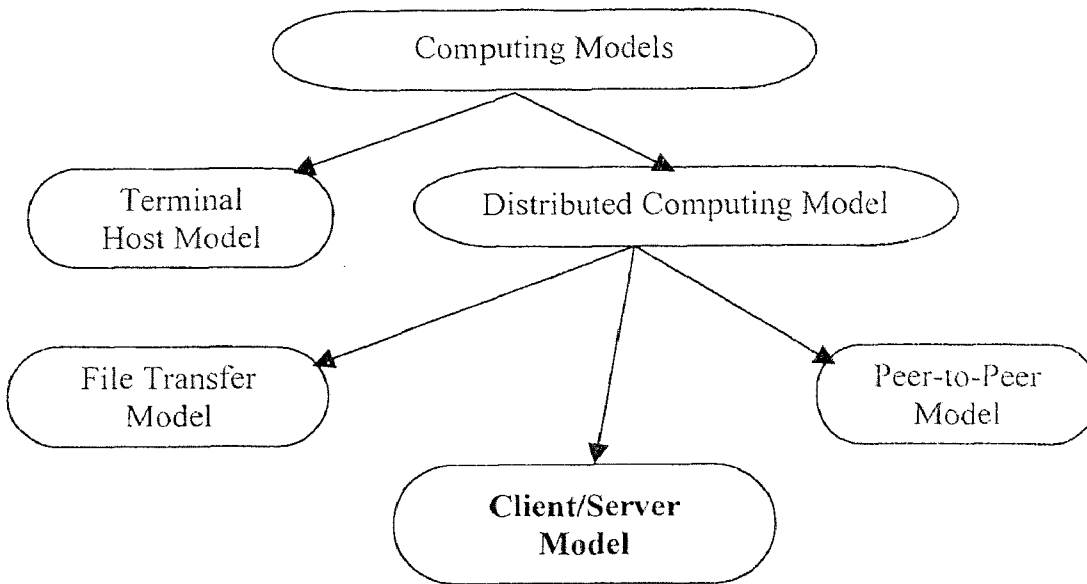
Figure 2.4: Interrelationships between Computing Models Source: *Application (Re)Engineering.* , Bell Communication Research (Bellcore), Picataway, NJ. 1997

Basically, client/server model is a special case of distributed-computing model as shown in figure 2.4. Distributing Computing System (DCS) is a collection of autonomous computers that exchange messages through a communication network. Distributed computing can be realized by File Transfer model, Client/Server Model or Peer-to-Peer Model as shown in figure 2.2. File Transfer model is one of the oldest models to realize distributed computing. Applications of a file transfer are exchanges of e-mails, media clips, news items etc. Client/Server model is a considerable improvement over file transfer as this model allows application processes at different sites to interactively exchange information. Peer-to-peer model allows the processes to invoke each other located at different sites. In peer-to-peer model the interactive processes can be a client, server or both while in client/server model one process one process acts as a server and the other one as a client [UMAR 97].

Client/Server systems can be realized by using sockets, Remote Procedure Calls (RPC), Remote Data Access (RDA) or Queued Message Processing. Using sockets is requires lot of coding work even for a simple client/server system; but very easy to debug the applications. A client writes a request in a predefined format and writes to socket. A server on the another end receives request from the socket it is bound to, parse and process the request and sends back appropriate response to a client via socket. The client/server architecture used in thesis of REPI implementation, socket mechanism has been used. Due to low-level working it is not used much in the practice. Remote Procedure Call or RPC implements a mechanism that allows a client process to invoke a remotely located server process. A server executes the procedure and sends the result back in response. RPC is widely applied to the real world problems from simple application to complex one. This distribution of processing reduces network traffic and improves performance. Site autonomy can also be increased by procedure modifications to locally executing applications. "The RPC mechanism makes it appear the client is directly calling a procedure, located on a remote server, as if it was calling a local procedure. Actually, the client application calls a local stub procedure instead of the actual remote procedure. The client stub then retrieves the required parameters from the clients address space, translates them, as needed, into a standard network data representation (NDR), and then uses the RPC run-time library API to send the request and parameters to the server. On the server side, the RPC run-time library API functions accept the request and call the server stub. The server stub moves the parameters from the network buffer and converts them into the format expected by the server. The server stub then calls the server passing it the parameters. The same mechanism is used, in reverse, to

return data from the server to the client" [ANONYMOUS 2]. Remote Data Access (RDA) paradigm allows client applications and/or end-user tools to issue ad hoc SQL queries against databases located remotely. For database applications RDA is heavily used. In Queued Message Processing (QMP) client/server communication takes place via queue. A client stores its request in a queue and waits for the response. When the server is free it fetches requests one by one from the queue clients have put their requests in and processes the request. The server stores the output of the processing to another queue for the access of the clients. Java Remote Method Invocation (RMI) like RPC mechanism allows to invoke remote methods from a local machine through net or Internet. Java Remote Method Invocation is now becoming popular mainly because of the portability of the Java language.

The vast majority of end user applications consist of three components: presentation (Distributed and Remote) functions, processing or business logic functions and data management (Distributed and Remote) functions. The client/server architectures can be defined by how these components are split up among software entities and distributed on a network. There are a variety of ways for dividing these resources and implementing client/server architectures. Following paragraph focuses mainly on the most popular forms of client/server computing systems namely two-tier and three-tier architecture.

2.4.2 Two Tier Client/Server Architecture

A robust client application development language and a versatile mechanism for transmitting client requests to the server are good nutrition for a two tier implementation. The two-tiered client/server computing model is a direct result of applying the three styles of application component distribution mentioned in the previous paragraph on the two platforms- a client and a sever. This can be implemented as one of the three styles. In Remote Data Management model the application code consisting of presentation logic and business or processing logic reside on the client while server supports database or file services to provide data management. The PC client carries on much of the responsibility for application (functionality) logic with respect to the processing component, while the database engine (server) handles data intensive tasks. The server is empowered with integrity checks, query processing capabilities and central repository functions. Such a client that is loaded with both presentation logic and processing logic is called *fat* client. In Distributed business or processing logic model presentation and data manipulation logic resides on client as well as on server. Presentation is handled exclusively by the client, processing is split between client and server, and data is stored on and accessed exclusively via the server. In Remote representation model or data access topology all application and database functionality is kept on server. Such a client whose only job is to perform representation is referred to as *thin* client. In this model, a data engine would process requests sent from the clients. Currently, the language used in these requests is most typically a form of SQL. Sending SQL from client to server requires a tight linkage between the two layers. To send the SQL the client must know the syntax of the server or have this translated via an API (Application Program Interface). It must also know the

location of the server, how the data is organized, and how the data is named. The request may take advantage of logic stored and processed on the server which would centralize global tasks such as validation, data integrity, and security. Data returned to the client can be manipulated at the client level for further sub selection, business modeling, analysis, reporting, etc.

Network
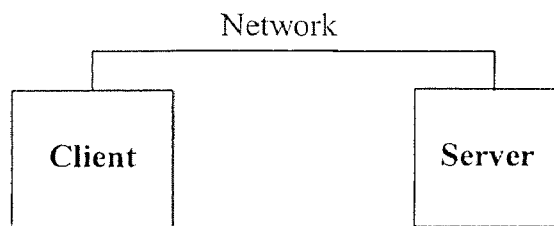
Client   Server

Figure 2.5: Two Tier Client/Server Architecture

The most compelling advantage of a two-tier environment is the rapid application development speed. In most cases it is quite easy to design and develop two-tier system. "Using any one of a growing number of PC-based tools, a single developer can model data and populate a database on a remote server, paint a user interface, create a client with application logic, and include data access routines. Most two-tier tools are also extremely robust. These environments support a variety of data structures, including a number of built in procedures and functions, and insulate developers from many of the more mundane aspects of programming such as memory management"[JOHN 96]. These tools can be used effectively for iterative prototyping and rapid application development (RAD) techniques, which can be used to ensure that the requirements of the users are accurately and completely met. Tools for developing two-tier client/server systems have

allowed many organizations to remove their applications backlog by rapidly developing and deploying smaller workgroup-based solutions. Two-tier architectural model works best in a homogenous environment, where all database servers are of the same type and where the business logic is relatively simple.

At the enterprise level the two-tiered architecture has many serious drawbacks. This architecture is less suited for dispersed, heterogeneous environments with rapidly changing rules. When a client has to connect to different server and DBMS systems, the client code is required to be modified. The two-tiered model relies on the DBMS for data integrity and consistency. In a heterogeneous environment a given server cannot guarantee the integrity of the transaction if a different DBMS server is involved [UMAR 97]. Due to this drawback relatively few IS organizations use two-tier client/server architectures to provide cross-departmental or cross-platform enterprise-wide solutions. Since the bulk of application logic exists on the PC client, the two-tier architecture faces a number of potential version control and application re-distribution problems. Modified clients would have to be re-distributed through the network a potentially difficult task given the current lack of robust PC version control software and problems associated with upgrading PCs that are turned off or not "docked" to the network [JOHN 96]. "System security in the two-tier environment can be complicated since a user may require a separate password for each SQL server accessed. The proliferation of end-user query tools can also compromise database server security. The overwhelming majority of client/server applications provide end-users a password which gives them access to a database. In many cases this same password can be used to access the database with data-access tools available in most commercial PC spreadsheet and database packages. Using

such a tool, a user may be able to access otherwise hidden fields or tables and possibly corrupt data [MAX 95]. Typically, a two-tiered architecture is implemented a synchronous processing model where in a client waits synchronously for the response from a server. This model offers limited scalability as the number of clients and transactions grow. Although it is possible to upgrade server hardware, it may require expensive changes in the infrastructure. An additional applications have to be placed on the client if the client is a *fat* one which could result in a very expensive and prolonged effort. Since clients are distributed throughout the enterprise, a change in client software results in a cumbersome job of upgrading of all the client software or even sometimes hardware [UMAR 97]. Client tools and the SQL middleware used in two-tier environments are highly proprietary. The client/server tools market seems to be changing at an increasingly unstable rate. In 1994, the leading client/server tool developer was purchased by a large database firm, raising concern about the manufacturer's ability to continue to work cooperatively with RDBMS vendors which compete with the parent company's products. The number two tool maker lost millions. The tool which has received some of the brightest accolades in early 1995 is supplied by a firm also in the midst of severe financial difficulties and management transition. This kind of volatility raises questions about the long-term viability of any proprietary tool an organization may commit to. All of this complicates implementation of two-tier systems. In such constantly changing situations, migration from one proprietary technology to another would require firm to scrap much of its investment in application code since none of this code is portable from one tool to the next [JOHN 96].

2.4.3 Three Tier Client/Server Architecture

The tree tier architecture has been developed to overcome some of the limitations of the two-tier scheme by separating presentation logic, business or processing logic and data management into separate, independent, distinct software entities called tiers as shown in figure 2.6.
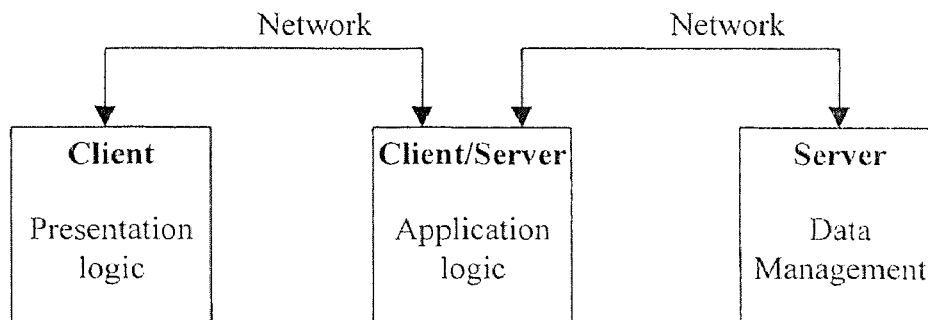
Figure 2.6: Conceptual Model of Three Tier Client/Server Architecture

When calculations or data access is required by a client, a call is made to a middle tier functionality server. This tier can perform calculations or can make requests as a client to additional server/servers. Middle-tier functionality servers may be multi-threaded and can be accessed by multiple clients, even those from separate applications. RPC is the most widely used method to implement three-tier client/server architecture. In RPC the requesting client simply passes parameters required for the request and specifies a data structure in which it is ready to accept returned values. This provides greater overall system flexibility than the SQL calls made by clients in the two-tier architecture. Unlike most two-tier implementations, the three tier presentation client is no longer required to use SQL. This allows the modifications in the overall structure of the back-

end data without altering presentation logic on client sites. Due to independence from SQL data management can be achieved through hierarchical, relational, or object format. This eliminates the introduction of new database technologies to access legacy data. Due to separate software entities, parallel development of individual tiers is encouraged by this architecture leading to faster development of the system. Also, having experts focus on each of these three layers can increase the overall quality of the final application. The three tier architecture provides more flexibility about allocation of resources. "Middle-tier functionality servers are highly portable and can be dynamically allocated and shifted as the needs of the organization change. Network traffic can potentially be reduced by having functionality servers strip data to the precise structure required before distributing it to individual clients at the LAN level. Multiple server requests and complex data access can emanate from the middle tier instead of the client, further decreasing traffic. Also, since PC clients are dedicated to just presentation, memory and disk storage requirements for PCs will potentially be reduced"[JOHN 96]. Reusable Modular middle tier can reduce subsequent development efforts, minimize the maintenance work load and decrease migration costs when switching client applications. In addition, implementation platforms for three tier systems such as OSF/DCE offer a variety of additional features such as integrated security, directory and naming services, server monitoring and boot capabilities for supporting dynamic, fault-tolerance and synchronizing systems across networks and separate time zones.

Figure 2.7 shows the platform dependent three-tier client/server architecture where in computing resources are shown distributed vertically. As shown in figure 2.7, the top tier is occupied by the most powerful computing system and the source of

corporate data, the mainframe. LAN servers with dual properties form the second tier. Not only they act as top-tier clients that make appropriate request to mainframe server; but act as servers for the workstations and PCs. The third tier contains workstations and PCs. As shown in figure 2.7 this architecture is motivated by market forces such as price/performance ratio, functionality and local autonomy. This architecture is complex as it has to accommodate intra-tier communication for network management, system performance, data integrity and reliability. However this architecture increases the overall computing capacity of a client/server system. This architecture eliminates the scalability problem suffered by two-tier architecture. The disadvantage of this architecture is that it is platform dependent. Modifications in second and third tier require subsequent changes in the client software. Figure B.3 shows more powerful function based model of three-tier client/server system that has been evolved to overcome the drawback- platform dependency- of the one shown in figure 2.7. The entire architecture can be divided into three different classes of systems: namely Clients, Application servers and Data servers. Presentation is handled by clients with the powerful capabilities of Graphical User Interfaces. Thus this architecture favors thin clients. Application servers carry on bulk of processing and data logic. They perform workgroup application functions, support the network domain operating systems, stores and execute common processing rules, support a data directory etc. The third part of this system consists of Data server. Data servers are mainly focused on supporting relational and object database management systems, data warehouse processing, systems management configuration of databases etc.
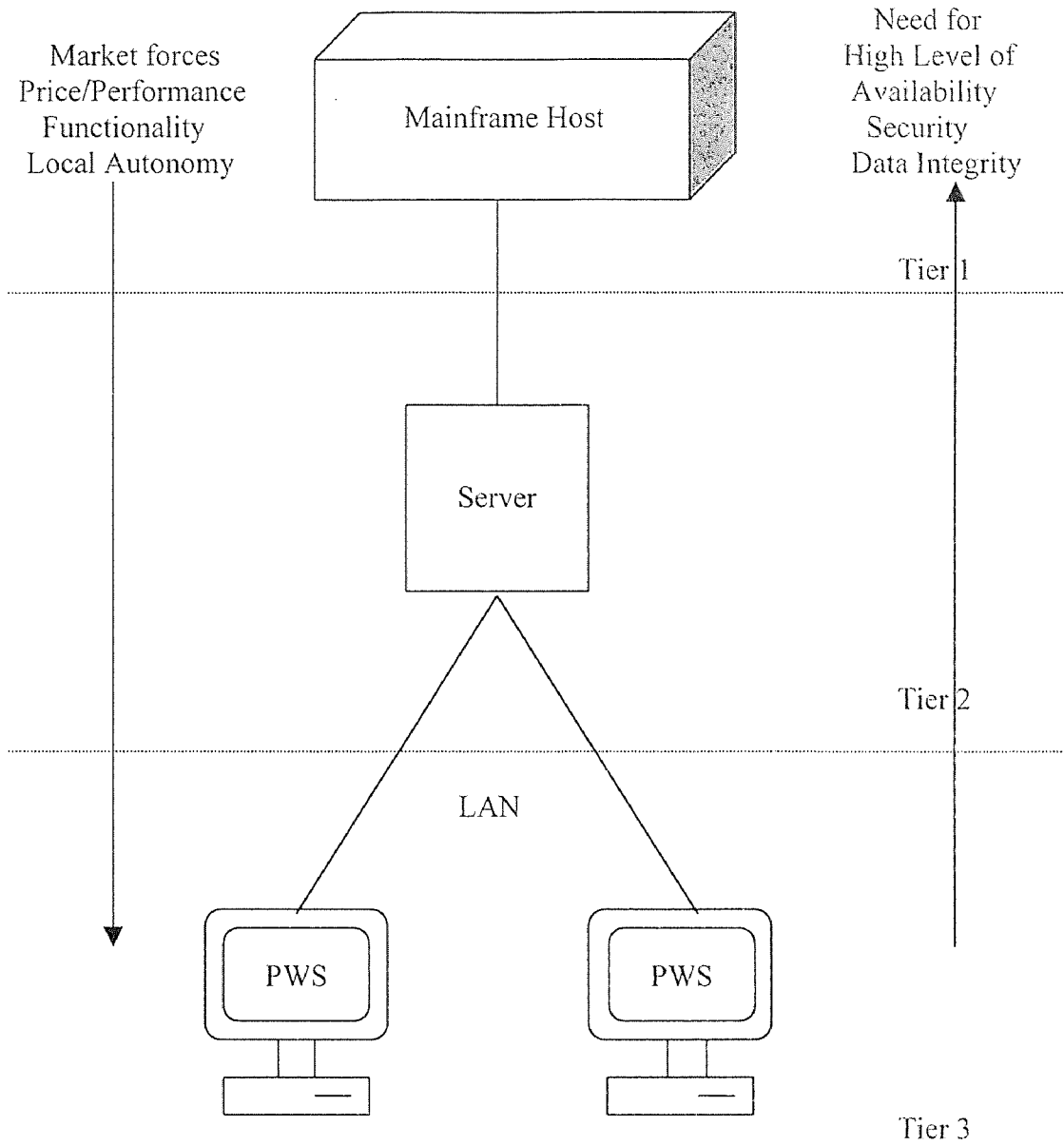
Figure 2.7: Platform Based Three Tiered Architecture Source: *Application (Re)engineering*. Amjad Umar, Bell Communication Research (Bellcore), Picataway, NJ. 1997

The advantages of this configuration is thin clients, manageability, scalability, platform independence and heterogeneity as can be imagined from figure B.3. Clients are thin because they have to worry about presentation only and are free from burden of business logic. It is manageable because all the three components are separate from each other. Clients can communicate Unix server, Windows NT server or NetWare server, thus it is platform independent and heterogeneous. New clients can be supported by adding new servers and thus scalable. However this architecture like other systems is not flawless. "Current tools are relatively immature and require more complex 3GLs for middle tier server generation. Many tools have under-developed facilities for maintaining server libraries – a potential obstacle for simplifying maintenance and promoting code re-use throughout an IS organization. More code in more places also increases the likelihood that a system failure will effect an application"[JOHN 96]. Three tier increases network traffic management, server load balancing, and fault tolerance responsibilities.

2.5 Client/Server Architecture for REPI Implementation

A three-tier client/server architecture as shown in figure B.3 has been chosen due to following reasons. REPI suggests that in order to prepare better requirement specifications effective communication has to be established between developer community and user community. The Internet can easily provide this facility by allowing team members to communicate with each other asynchronously and concurrently through database. As described Java language is proving itself on of the best Internet technology. Using Java applets both communities can suggest their opinions as well as can read the opinions of the other team members. Java Data Base Connectivity (JDBC), for security

reasons, does not allow an applet to communicate directly with the database server which is Oracle in this case. If this restriction is not placed and Java applets are allowed to modify the database directly it is quite possible that some internet hacker may corrupt the data. However, an application can communicate with the database server using JDBC protocols. Mainly because of this reason a need for the three-tier architecture has emerged as shown in figure B.3.
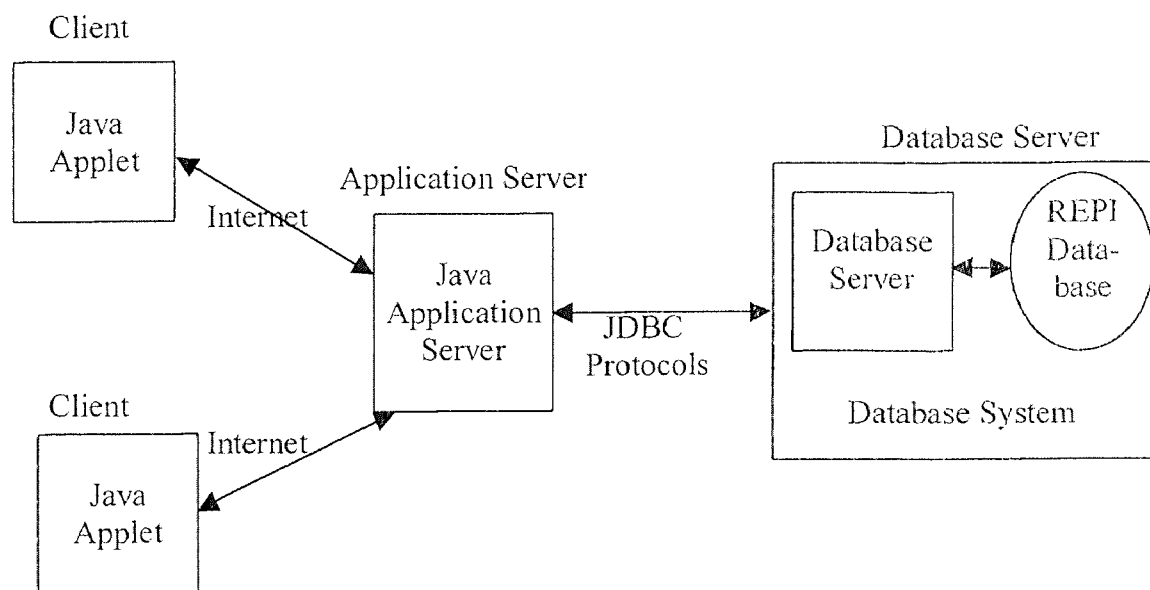
Figure 2.8: Overview of Three Tier Client/Server Architecture for REPI Implementation

When a user connects to REPI site the Java applet is down loaded over the Internet and executed by a browser. Applet communicates with the application server written in Java language. Application server can either run locally or to a site remote to the machine the database server is running on. Client/Server system REPI uses socket mechanism for message exchanges between Java applets and Java application server. When a user clicks a button labeled "SUBMIT", applet opens a socket connection,

creates a SQL query and writes it down to a socket. On the other side, application server binds itself to a particular port and waits for requests. When the server accepts request from applet it creates a new thread of execution that will take care of that applet as long that applet maintains the connection. When an applet writes to a socket, application server read from socket, parse the request string and invokes appropriate method to perform desired operation. The method using JDBC protocols communicates with the database server and returns the result back to the Java applet. Application server supports simultaneous access through multiple applets. Thus, three-tier client/server architecture is useful for realization of concurrent and simultaneous access to REPI Database.

2.6 Client/Server Evaluation

Today's powerful workstations can provide many powerful features only available from mainframes. Tremendous computing power coupled with very low prices PCs are becoming more favorable. Client/Server computing can definitely take advantage of this opportunity and provide powerful capabilities to the end users. It allows processing to reside near the source of data and thereby traffic on the network can be greatly reduced resulting in lower bandwidth and cost of installation of network systems. Due to amazing power of workstations in the field of GUI and multimedia applications organizations can provide visual presentations to their customers. Due to visual capability of the tools cost of training can be greatly reduced. It encourages open systems. Clients and servers can run on different hardware platforms and thereby frees end users from proprietary architectures. If implemented in a proper way, client/server system can reduce software maintenance cost, increase software portability, boost the performance of existing

network etc. However, as happens with every systems client/server is not perfect. It has some drawbacks too. If most of the application logic is implemented on the server side, that server can become bottleneck resulting in poor price/performance ratio. Since clients and servers are distributed over the network their design, development, implementation and debugging are complicated as compared to other traditional systems.

The tremendous growth rate of the Internet has opened plenty of opportunities for the client/server system. Certainly, client/server system is one of the hottest technologies of the present and future ages.

CHAPTER 3

REQUIREMENTS ELICITATION

3.1 Introduction

"Without an understanding of technologies one may aim for the impossible, and without an understanding of needs, one may solve the wrong problem."- Quotes in [CHRISTEL 92]

If you want to do something, the first question your mind will throw is what to do? If you want to go somewhere, the first question is where to go? Similarly, the first question comes to the mind of a developer is what to produce or develop? Then the questions like how and what come into picture. This shows that in any kind of activity that may be travelling, learning, teaching or developing is involved a certain kind of methodology and that is defined in scientific terminology as Requirements Engineering. The following section describes about the Requirements Engineering. Then the next section describes in detail about the Requirements Elicitation- an earlier stage of Requirements Engineering.

3.2 Requirements Engineering

Requirement Engineering is a key problem area in the development of complex systems.

"The hardest single part of building a system is deciding what to build. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later"[BROOKS 87].

What is Requirement Engineering? Here are some definitions of the Requirement Engineering. "The process of establishing the services the system should provide and the constraints under which it must operate is called Requirements Engineering. System requirement should set out what the system should do rather than how this is done"

[SOMMERVILLE 95]. "The process of acquiring, refining and checking client needs for a software system is called requirement engineering" [IEEE 90a]. [BOEHM 79] defines Requirement Engineering as "the discipline for developing a complete, consistent, unambiguous specification- which can serve as a basis for common agreement among all parties concerned- describing what the product will do (but not how it will do it; this is to be done in the design specification)". Requirement Engineering comprises those "processes by which the purchaser's statements of intention and requirement, written or spoken, are transformed into a precise, unambiguous, consistent and complete specification of system behavior including functions, interfaces, performance and constraints"[STARTS 87]. As per [LEITE 87] Requirement Engineering is defined as "a process in which "what is to be done is elicited" and modeled. This process has to deal with different viewpoints, and it uses a combination of methods, tools and actors. The product of this process is a model, from which a requirements document is produced."

3.2.1 Importance of Requirements Engineering

Figure 3.1 shows the simplified version of the development cycle model, of a product or system, that consists of several well-defined phases such as System requirements phase, Design phase, Implementation phase, Testing phase and Operation and maintenance phase. The output of each phase is a document that forms the starting point of another phase. These phases recur to accommodate client needs and requirements.

The system requirements phase describes the goals of project, investigates the client needs and the application domain and specifies requirements. The client needs and

Client Needs and Requirements

Operation and Maintenance

System Requirements

Product in Use

Specification

Testing

Design

Documentation

Architecture

Implementation
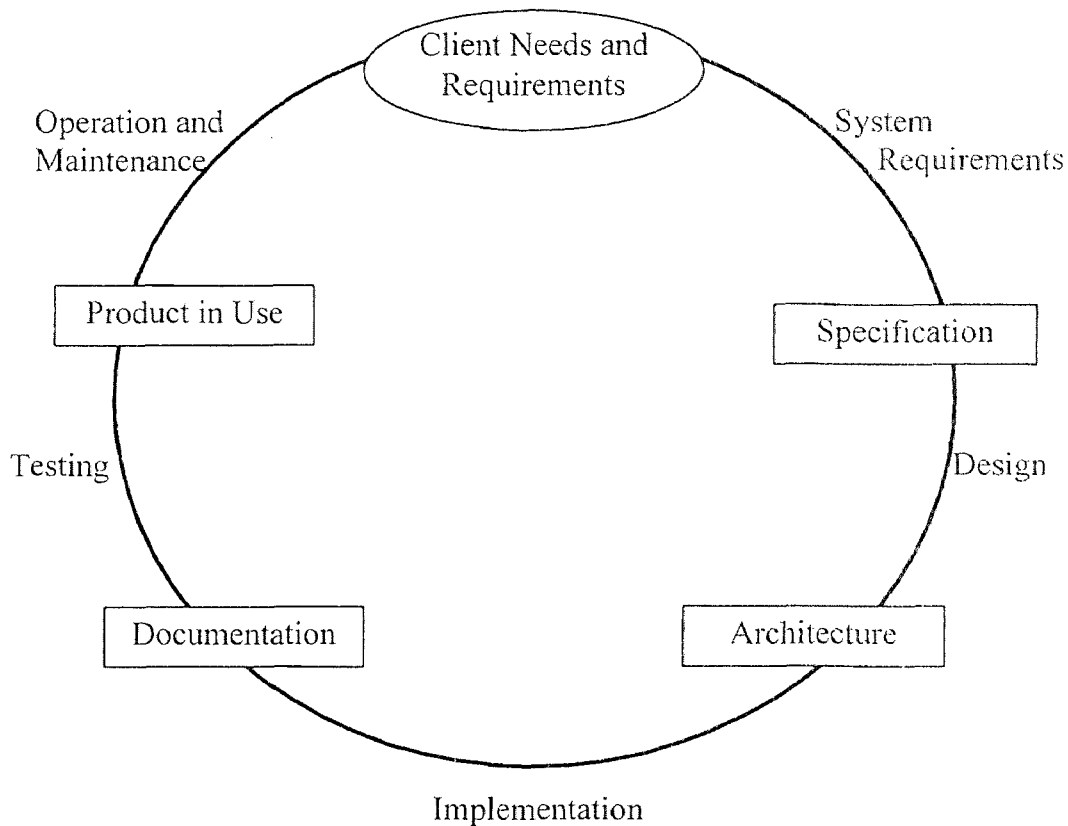
Figure 3.1: Development Process Model Source : "Requirement Engineering: A Survey of Methods and Tools." Hubert F. Hofmann, University of Zurich, Zurich, May 1993

requirements are nothing but "wish-list" for the system under consideration. This phase produce system specification. The design phase determines the system architecture i.e. its components and sub-components. The implementation phase generates an operational system based on concepts defined in earlier phases. The testing phase checks for the correctness of the system being put into operation. The operation and maintenance discovers errors not detected in earlier phases and enhances the system as per new formulated requirements [HOFMANN 93].

One can easily derive from figure 3.1, the importance of the system requirements. Requirements Engineering deals with the system requirements phase. Many times understanding the nature of a problem can be very difficult. If the system is novel, there is no existing system to help understand the nature of the problem. Hence it is very difficult to establish exactly what the system should do. Many requirements errors are passed undetected to the later phases of the life cycle, and correcting these errors during or after implementation has been found to be extremely costly [CONGRESS 90]. Various studies have shown that the later wrong development decisions are detected, the more expensive they are to repair [BOEHM 89]. This is very true for the decisions taken during Requirement Engineering as it is the very beginning stage of system development. The Department of Defense (DoD) Software Technology Plan states that "early defect fixes are typically two orders of magnitude cheaper than late defect fixes, and the early requirements and design defects typically leave more serious operational consequences" [DOD 91].

Requirement Engineering forces clients to consider their requirements carefully and review them within the boundaries of the problem. This process aims at a complete and accurate specification of the system. It records and refines requirements, enhances the transparency of the system by improving the understanding of the system among the clients and encourages the communication among clients and developers. The project is likely to fail unless clients and developers come on consensus about the requirement specification[HOFMANN 93]. As shown in figure 3.1, Requirement Engineering enables test plan development to prove design and implementation for correctness and completeness. It verifies this against the standards set out by requirement specification

phase of Requirement Engineering. Requirement Engineering provides the management people estimates of cost, time and resources needed in advance. It also enables them to define constraints for future changes and maintenance. If Requirement Engineering activities are insufficient, clients/users will lose confidence in the development team which will affect not only the current project but also clients' long-term attitude toward the development team [HOFMANN 93]. Experience has shown that incorrect, incomplete, or misunderstood requirements are the most common causes for poor quality, cost overruns, and late delivery of software systems [RAGHAVAN 94].

The above observations validates the appropriateness of the following quote for Requirement Engineering:

> "Studying user needs is a first step to any solution, along with gaining an understanding of available technologies and existing tools. These two tasks interact. Without an understanding of technologies one may aim for the impossible, and without an understanding of needs, one may solve the wrong problem"[CHRISTEL 92].

3.3 Requirements

Unfortunately, like Requirement Engineering, the term requirement is not used throughout industries in a consistent way. In some cases requirements are viewed as a high level, abstract statement of a service that the system should provide or a constraints on the system. At the other extreme, it is a detailed, mathematically formal definition of a system function [SOMMERVILLE 95]. [DAVIS 93] exemplified these differences as:

> "If a company wishes to let a contract for a large software development project, it must first define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that

several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called requirement documents for the system."

There exists s clear separation between requirements definition and requirements specification which many people are unable to clear out. "A requirement specification is a statement in a, natural language plus diagrams, of what services the system is expected to provide and the constraints under which it must operate. It is generated using customer-supplied information" and "A requirement specification is a structured document which sets out the system services in detail. This document, which is sometimes called a functional specification, should be precise. It may serve as a contract between the system buyer and system developer" [SOMMERVILLE 95].

[IEEE 90a], which is a well-known non-profit institute to promote innovative technologies, has defined requirements in the following terms:

"(1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (3) A documented representation of a condition or capability as in (1) or (2)"

[STEP 91] gives yet another definition of a requirement as quoted below:

"A requirement is a function or characteristic of a system that is necessary... the quantifiable and verifiable behaviors that a system must possess and constraints that a system must work within to satisfy an organization's objectives and solve a set of problems."

Requirements can be of two types: Functional requirements and Non-Functional requirements [HOFMANN 93] and [CHRISTEL 92]. Functional requirements define the functions that system or one of its components performs on inputs to generate outputs. For example, the requirements for a money dispenser have to answer a question like: What input is required to withdraw or deposit money? What have I to do get a receipt or money? Non-functional requirements are constraints on the system under consideration. Performance, time and space, accuracy, robustness, availability of equipment and technology, user interfaces, maintainability, security, standards, budget, political aspects etc are constraints that can be imposed on the system under development. Under what conditions can we change the personal code of our card? How long should the money dispenser wait for user input? are good examples of non-functional requirements. Sometimes they can become functional requirements. For example, a requirement for reliability can be translated into some function for error reporting [HOFMANN 93].

[SOUTHWELL 87] has given the following classification of requirements:

- Functional Requirements
- Non-functional Requirements
  - Performance Reliability

- Interfaces

- Design Constraints

[ASHWORTH 89] has made following classifications of requirements:

- Functions ("what")

- Data ("what")

- Non-functional requirements ("how well")

- Goals (Directions to guide developer to implements the user requirements)

- Implementation / Design Constraints (e.g. use C)

In [SAGE 90] Sage and Palmer classify requirements as:

- Technical system Requirements which are primarily functional requirements

- Management system Requirements which include cost and time constraints as well as quality factors for requirements

From the evolution perspective of requirements [SOMMERVILLE 95] classifies requirements into following two categories:

- Enduring requirements are relatively stable requirements which derive from the core activity of the organization and which relate directly to the domain of the system. For example, in a hospital there will always be requirements concerned with patients, doctors, nurses, treatments, and so on.

- Volatile requirements are likely to change during the system development or after the system has been put into operation. Government health-care policies are good examples of volatile requirements. Volatile requirements can be further categorized into five divisions:

- Mutable requirements changes due to changes in the environment the organization is operating.

- Emergent requirements emerge as the customer's understanding of the system develops during the system development.

- Consequential requirements result from the introduction of the computer system as the introduction of computer system may change organization's processes and disclose novel ways of working which may result in the generation of new requirements.

- Compatibility requirements depend on the particular systems or business processes within an organization.

[IEEE 90b] has defined six types of requirements: functional requirements, performance requirements, interface requirements, design requirements, implementation requirements and physical requirements.

3.4 Requirements Engineering Process

Figure 3.2 shows Requirement Engineering process, which is a set of activities that lead to the production of the requirements, definition and requirement specification. Other miscellaneous information such as feasibility report may also be produced during this process.
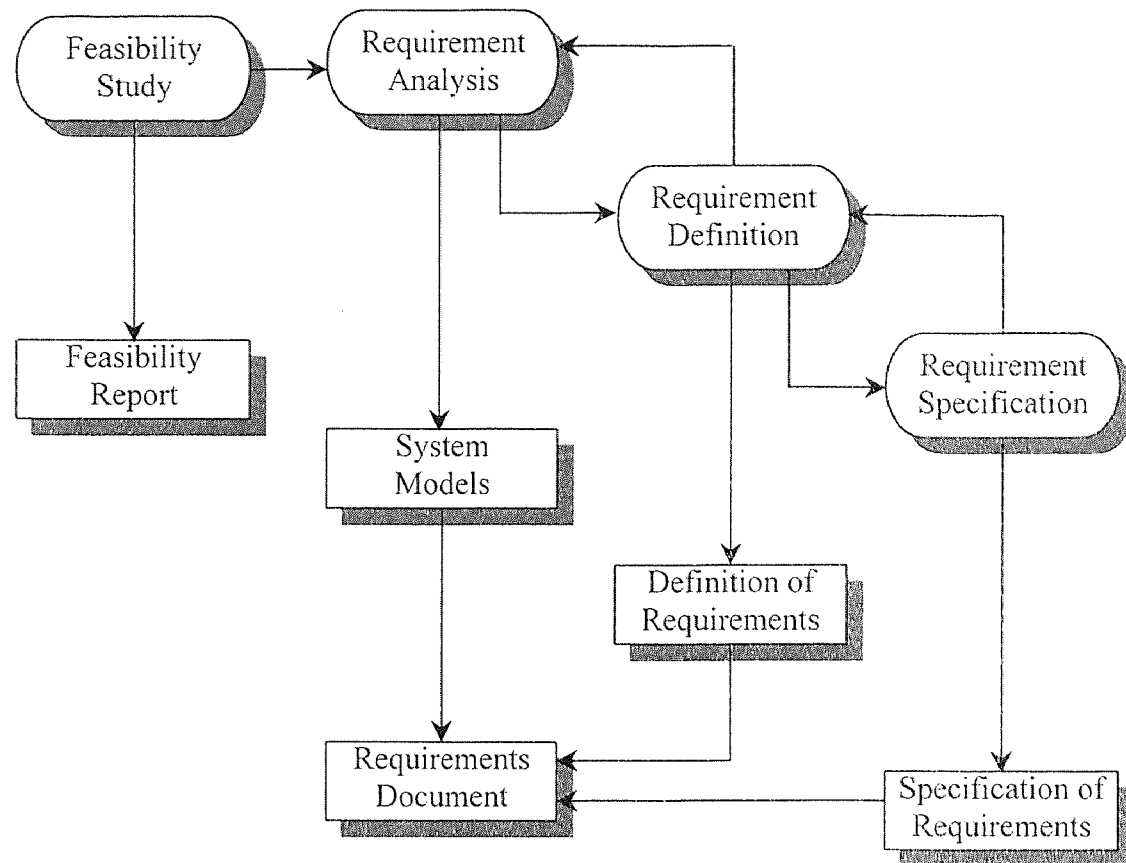
Figure 3.2: The Requirement Engineering Process Source: *Software Engineering*, Ian Sommerville, Fifth Edition, Addison-Wesley Publishers Ltd. 1995

The four principle stages of requirement process suggested by [SOMMERVILLE 95] are:

1. Feasibility study

2. Requirement analysis

3. Requirement definition

4. Requirement specification

During feasibility study phase an estimate is made of whether the identified user wants may be satisfied using the current available technologies or not. The outcome of the study will suggest from a business point of view the cost effectiveness of the

proposed system and if it can be developed within the given budgetary constraints. The feasibility study has to be relatively cheap and quick and has to give the indication for the green signal of the proposed system or not.

Requirement Analysis or requirement elicitation is the process of deriving the system requirements through observation of existing systems, interviewing potential users and procurers, task analysis and so on. After feasibility studies, the first major step of the Requirement Engineering process is requirements analysis or elicitation. This may involve the development of one or more system models to help the analyst perceive the system to be specified. Sometimes prototypes may be developed to acquire proper understanding of the system under consideration. This phase is the point of attention in this thesis and we will discuss about this phase in detail in the following section. In requirement definition phase information gathered during analysis stage is translated into a document. This document is nothing but the definitions of a set of requirements that reflects the customer "wish-list". Requirement specification phase is responsible for the settlement of the contract between the client and developer as in this phase a detailed and accurate description of the system requirements is set out. As evident from the figure 3.2, the activities in requirement process are not sequential but are carried out in iteration. The requirement analysis continues during definition and specification. New requirements may arise during the process [SOMMERVILLE 95].

3.5 Requirements Analysis or Elicitation

As we have perceive through above discussion, Requirements Elicitation is one of the earlier stages of Requirements Engineering, which itself is one of the earlier stages of system development process. Requirements Elicitation is defined as "the process through which customers, buyers, or users of a software system discover, reveal, articulate, and understand their requirements" [RAGHAVAN 94]. Requirements Elicitation can also be thought of as the process of gathering different viewpoints from various sources and reaching consensus on a common ground. "The prime objective of the requirements definition process is to achieve agreement on what is to be produced" [BRACKETT 90]. The Requirements Elicitation process has also been called identifying, gathering, determining, formulating, extracting, or exposing requirements [RAGHAVAN 94].

[RAGHAVAN 94] describes the Requirements Elicitation steps as:

- Identify the sources of requirements,

- Gather information about their needs,

- Analyze the information for implications, inconsistencies, or unresolved issues,

- Reconcile the differences in understanding between users and analysts, and

- Generate the requirements statements.

These steps are typically repeated again and again until a complete and common understanding of the system to be developed has been reached among the people involved.
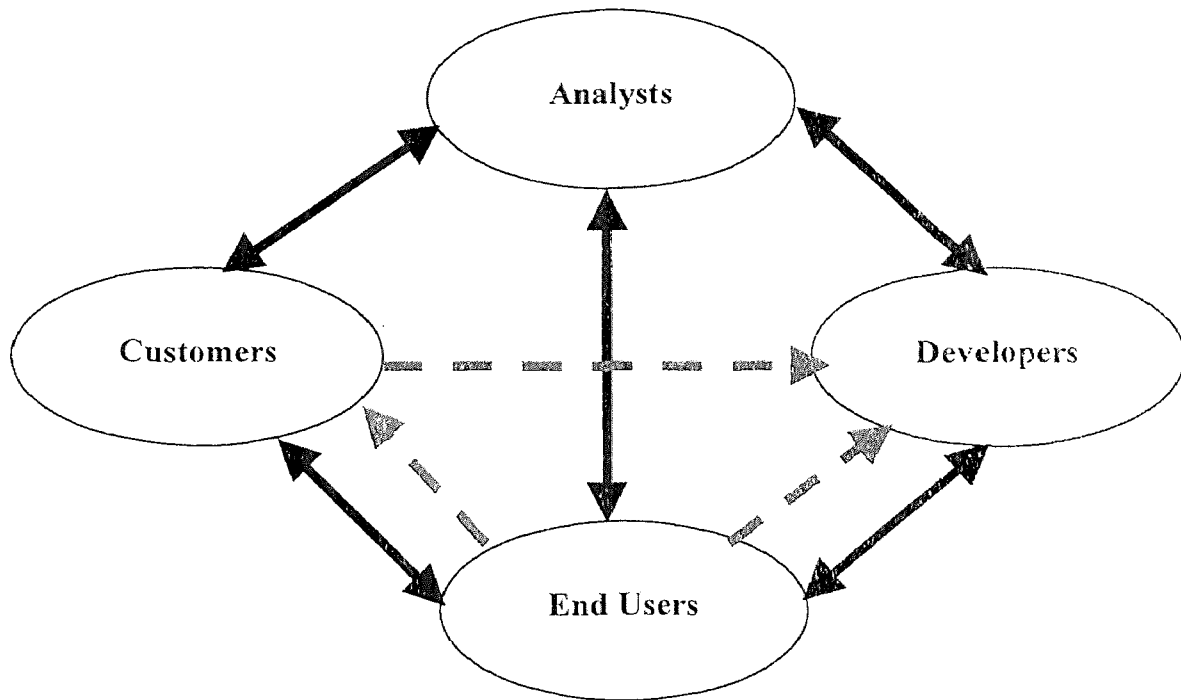
Figure 3.3: The General Model of Communication for Requirements Elicitation

The people involved in the Requirements Elicitation process are: analysts, developers, customers and end users. Elicitation can be defined as "the process of identifying and bridging the disparities among the involved communities for the purpose of defining and distilling requirements to meet the constraints of these communities" [CHRISTEL 92]. Figure 3.3 shows the general model of communication for requirement elicitation. The information flow showed with dark lines is bi-directional between most of the involved communities. The dashed lines in the figure are indication of controlling influence rather than direct communication.

Requirements Elicitation's importance is directly tied to the significance of the Requirements Engineering in general because eliciting requirements is the first major step of Requirements Engineering. The problems associated with Requirement Engineering such as defining the scope of the system under consideration, improving

understanding among different communities involved in and affected by the development of the given system and volatility of the requirements may lead to poor requirements and the cancellation of the system, or else the development of a system that is unsatisfactory or unacceptable, too costly or has high performance cost. By improving requirement elicitation, the requirements engineering process can be improved reflecting eventually into enhanced system requirements and thereby a much better system [CHRISTEL 92]. So the Requirements Elicitation process is essential for the development of quality software products [MILLER 93] and [CHRISTEL 92]. Because of the importance given to Requirements Elicitation, many techniques have been developed for this process. Many techniques have elaborated on the general procedure described above. Some are high-level frameworks, process models, or methodologies that provide general guidelines for eliciting requirements. Others are low-level techniques or methods that provide specific tactics for eliciting requirements. These techniques give detailed processes, specific questions or categories of questions to ask, structured meeting formats, individual or group behaviors, and templates for organizing and recording information [RAGHAVAN 94]. However, this thesis is basically the extract from the requirement elicitation process defined by the Software Engineering Institute (SEI) of Carnegie Mellon University. The rest of the sections of this chapter deal in detail with the SEI's method of requirement elicitation technique.

3.5.1 Issues in Requirements Elicitation

Requirement Elicitation issues have been categorized by [CHRISTEL 92] into three groups:

- Problems of scope, in which requirements may not convey proper amount of information.

- Problems of understanding, in which different people have different views about the perception of the given requirements, and

- Problems of volatility, i.e. the changing nature of requirements.

The requirement elicitation problems given in [MCDERMID 89] can be classified according to the above framework as:

- Problems of scope

  - the boundary of the system is not defined improperly

  - more than design information may be given

- Problems of understanding

  - user have vague understanding about their needs

  - poor understanding of capabilities and limitations of available technologies

  - analysts have little knowledge of problem domain

  - differences in languages of users and analysts

  - omitting of obvious information

  - differences in viewpoints of users

- Problems of volatility

  - requirements evolve over time

SEI's Requirements Elicitation framework consists of a process model, a methodology, and a set of techniques or methods. Figure 3.4 shows this framework and inter-relations of its components. The idea behind the framework is to take an individual
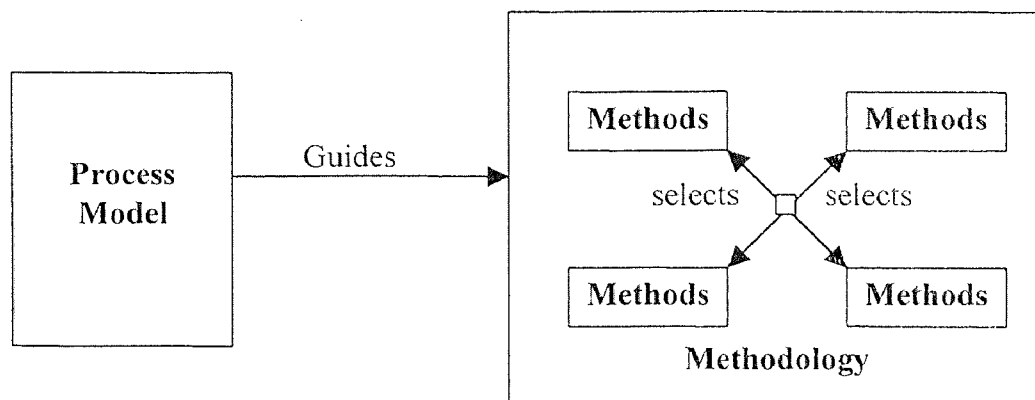
Figure 4.4: Requirement Elicitation Framework Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

methods and techniques and combine them into a methodology which can be tailored for each situation [CHRISTEL 92]. For this purpose the process model is used to guide the methodology. The process model provides a strategy that improves upon the problems of Requirements Elicitation mentioned above. The methodology recommends the methods and techniques to be used based on the situation at hand [MILLER 93].

There is difference between a method and a methodology. The degrees to which requirements will be influenced by contextual factors, require communication among the people involved, involve large quantities of data, and change over time are dependent on the target system. To address this dependency, the requirements elicitation approach to solving the problems associated with requirements elicitation will be a methodology, as

opposed to a single method [CHRSITLE 92]. [CHECKLAND 89] defines the difference in these words: "It is the essence of a methodology-as opposed to a method, or technique that it offers a set of guidelines or principles which in any specific instance can be tailored both to the characteristics of the situation in which it is to be applied and to the people using the approach... Such is the variety of human problem situations that no would-be problem solving approach could be reduced to a standard formula and still manage to engage with the richness of particular situations." Method is a particular instance of the techniques applied to solve a particular problem. Methodology is "a collection of models for specification and a set of rules for their use" [MULLERY 89]. The rest of the chapter describes this framework in more detail. The next section describes the proposed process model for the framework. Methodology and Methods along with the different stages of the process model and the activities of each stage in the process model are then discussed at the end of this chapter.

3.6 Requirements Elicitation Process Model

This model is a first step towards integrating partially successful past elicitation techniques into a more meaningful methodology based on the process model to be discussed hereafter. The proposed requirements elicitation technique comprises of five phases: fact-finding, information gathering, and integration as shown in figure 3.5. The first Fact finding phase includes examination of environment into which the target system is to operate, high level target system's role or mission statements, constraints on the architecture, existence of similar systems and so on. In the second phase of requirement gathering determines what is to be built through multidisciplinary views. Evaluation and

Rationalization phase discloses inconsistencies that may exist in the information gathered during second phase of requirement elicitation. The fourth phase of prioritization and planning determines the relative importance of the requirements. Integration and Validation forms the final phase of the requirement elicitation process. The job of this phase is to bring together the information collected from the previous steps into a set of requirements on which all the people concerned with the system under development reach general agreement. Elicitation implies communication between different sets of people: analysts, customers, developers, and users. The requirements analysts are responsible for the capture of system requirements from the user community and its communication to the developer community [CHRISTEL 92]. The analyst is the middleman between the user and the developer and therefore it is the analyst who has to make sure that other people, such as customers, are involved in the Requirements Elicitation and that all the affected groups have a common understanding of these requirements. Recognizing the importance of this communication, a structured model for this process of communication is created as the main part of the framework. This process model governing the framework is shown in Figure 3.5. The figure shows that the requirement elicitation process is not a linear one but iterative in nature. Fact-finding phase can be re-entered from any of the other phases. Requirement and Gathering phase can be re-entered from three stages. Similarly, prioritization and planning phase can be re-entered from integration and validation phase. Even the iteration can take place between any of the two consecutive phases depending upon the nature of the requirements. Since not all of the requirements for a target system are typically known immediately, iteration among these five phases of requirement process is necessary to

detail and improve requirements documents. This model recognizes the importance of communication between different stakeholders. The proposed requirement elicitation model consists of two sets of activities to address diversities on the backgrounds and motivations of the elicitation participants. One set of activities is user-oriented and the other set of activities is developer-oriented.
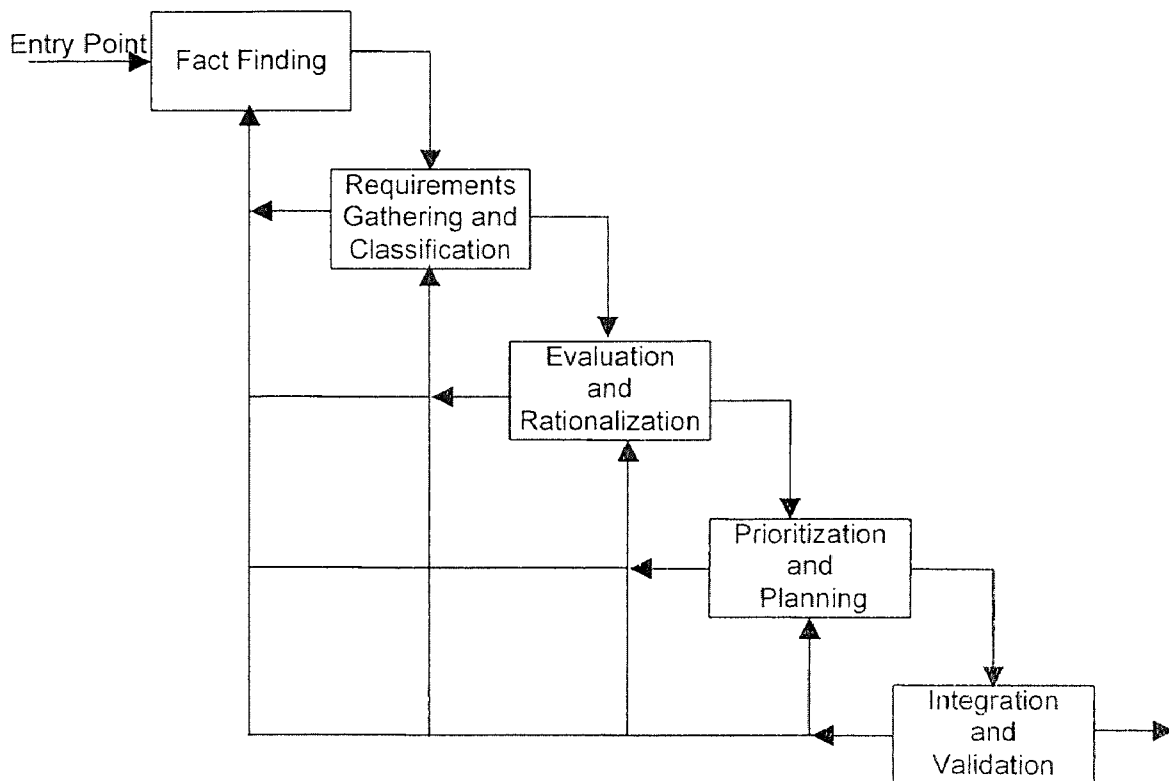
Figure 3.5: Requirement Elicitation Process Model Source: "Issues in Requirements Elicitation." Christel, Michael G and Kyo C. Kang, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. September 1992.

There are numerous other methods for requirements elicitation besides the one suggested by SEI's. Determining directly, deriving from existing systems, normative analysis, strategy set transformation, critical success factors, key indicator analysis, prototyping, scenarios, information needs analysis etc [CHRISTEL 92]. However, this thesis as mentioned before will follow the guidelines set up by Software Engineering Institute for the implementation of REPI prototype. The next section deals with the five phases in general then the next chapter provides the detail information about the REPI implementation. We will discuss about these two sets of activities in chapter 4.

Table 3.1 shows the user-oriented and developer–oriented tasks for the five stages of requirements elicitation process.

3.6.1 Fact Finding

This is the first phase of the Requirements Elicitation process model discussed above. The main goals of this phase are "determining what problem is to be addressed, who needs to be involved in the decision making process, and who will be affected by the problem's formulation and solution" [CHRISTEL 92]. The outcome of these activities are "defined to be a statement of problem context, a statement of the overall objectives, and supporting representations of the boundaries and interfaces of the system" [MILLER 93]. Specifically these goals are separated into tasks and are distributed between user community and developer community. These tasks are: identify relevant parties; identify domain experts; determine operational and problem context; identify domain and architectural models; identify similar systems; conduct technological surveys; perform context analysis; and assess cost/implementation constraints [CHRISTEL 92].

Table 3.1: Requirements Elicitation Process Model's Tasks.   Source: "Issues in Requirement Elicitation."   Technical Report CMU/SEI-02-TR-12 or ESC-TR-92-012. Christel, Michael G. and Kang, Kyo C. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. September 1992.

Stages / Phases	User-Oriented Tasks	Developer-Oriented Tasks
1. Fact Finding	Identify relevant parties.	Identify domain experts.
	Determine operational and problem context. This should including mode of operation, goals, and mission scenarios.	Identify domain model and architectural model.
	Identify similar systems.	Conduct technological surveys.
	Perform context analysis.	Assess cost/implementation constraints.
2. Requirements Gathering and Classification	Get "wish list" of the parties involved.	Classify wish lists according to functional, non-functional, environment and design constraints.
3. Rationalization and Evaluation	Perform abstraction to answer questions of the form "Why do you need X?" this in effect moves from statements of "how" to statements of "what".	Perform risk assessment, addressing technical, cost and schedule concern.
	Capture rationale to support future requirements evolution.	
4. Prioritization and Planning	Determine critical functions for the mission.	Prioritize requirements based on cost and dependency.
		Identify architectural models, which support incremental development.
5. Integration and Validation	Address completeness issue.	Resolve conflicts i.e. perform consistency checking.
	Check that requirements are in agreement with the original goals.	
	Obtain authorization to move to the next step of development	

In well-understood problem domains, general understanding about the requirements may already exist in some form or be readily available. This can obviate multiple passes of this phase since the execution of this phase need not be as complex as in other cases [CHRISTEL 92], [MILLER 93]. In the case where such an understanding does not exist or the information can not be made readily available, multiple passes of this phase is needed, looping back even from the validation phase. Even if multiple passes are needed, later passes through this phase need not be as complex as earlier phases [CHRISTEL 92].

The Joint Application Design (JAD) technique; structured interviews; graphical Issue-Based Information System (gIBIS); Organizational Requirements Design for Information Technology (ORDIT); Customers Actors, Transformation process, Weltanschauung (world view) Owner, and Environmental constraints (CATWOE), objectives analysis model, domain models and technical surveys are few of the techniques that are used in this phase of the process model [MILLER 93].

Since JAD is structured meeting technique, JAD is used as a framework for the activities of this phase. A tailored version of the JAD technique, as used in this process model, consists of several stages executed in order: project research, preparation, session, and the final phase [MILLER 93]. In the project definition and research stages of JAD, structured interviews are used to capture information, which can be documented, using gIBIS. In the preparation stage, ORDIT, CATWOE, objectives analysis models, domain models, and technical surveys are used to represent the acquired information. The formalized representation is then checked for conflicts, inconsistencies, and unresolved or missing information. The JAD sessions are used to gather the missing information and

correct other errors in the gathered information. In the final phase, newly acquired information is integrated with previously gathered information. Once again the information is checked for conflicts, consistency and completeness. If any issues remain after this check, the fact-finding phase is reiterated, otherwise this phase is considered to be finished [DEEPAK 98].

3.6.2 Gathering and Classification

The main goal of requirements gathering and classification phase "is to obtain information regarding what is to be built in relation to the goals, objectives. and constraints developed in the fact-finding stage" [MILLER 93]. This phase has two main stages: get "wish list" for the clients and classify "wish lists" for the developers [CHRISTEL 92]. Developers have to classify these '"wish lists" according to their understanding about the system under development. The output of these activities are "representations and documents detailing the customer and user oriented objectives and needs" [MILLER 93].

JAD is used as the framework for this phase also. Structured interviews and questionnaires are used to capture the information directly from the end-users and stakeholders. The gIBIS technique is used to extract the underlying rationale behind the gathered information. Multiple viewpoints may exist in this activity because users and customers providing requirements information may have diverse understanding about the system to be built. These diversities in viewpoints make it difficult to analyze and identify conflicting viewpoints and other inconsistencies. COntrolled Requirement Expression (CORE) method can be used to organize these viewpoints. To handle the

problems of volatility of requirements and incremental requirements development, these viewpoints can further be divided into meaningful components. Entity diagrams and data flow diagrams can be used to model these components [CHRISTEL 92].

The JAD technique begins with the problem research stage to gather the objectives, needs, and requirements from the users and customers. Structured interviews, questionnaires, observations, and Scenario Based Requirements Elicitation (SBRE) are used to gather information during this stage [CHRISTEL 92], [MILLER 93]. The JAD preparation stage organizes and evaluates information obtained in the research step. The JAD session discusses and compares the requirements gathered in this phase and their relation to the objectives, goals, and constraints gathered from the fact-finding phase. Any conflicts found in the preparation stage are discussed during the session. The last stage of the JAD technique, the final phase, formally documents the information from this stage of the process model. This documentation also includes any conflicts found in this stage.

3.6.3 Evaluation and Rationalization

"The goal of this phase is to fully develop and evaluate the underlying rationale behind the requirements gathered to this point" [MILLER 93]. The activities for this phase are: to rationalize about the requirements for the users/customers and to perform risk assessment for the developers [CHRISTEL 92]. The purpose of these activities is to ensure completeness and consistency of the requirements gathered. The objectives, goals, and constraints developed in the first phase of the Requirements Elicitation process model are compared with the requirements detailed in the second phase of the process

model. The comparison is performed to verify that the requirements address the right issues complying with the right goals and solving the right problems. For that, a series of interviews, between the analyst and the stakeholders, is needed to evaluate the requirements model against the rationale provided by the usage of gIBIS in earlier phases. This evaluation identifies missing rationale and unnecessary items instead of looking just at the explicitly expressed requirements. This rationalization process also extracts true requirements hidden behind the rationale [CHRISTEL 92]. Technical surveys from the first phase of the model are used to perform risk assessment in this phase. "Once the rationale has been collected and examined, inconsistencies can ideally be found and better choices on decision points or issues made to both resolve these inconsistencies and address the needs reflected in the rationale. In addition, this rationale is extremely useful as documentation on why particular choices were made"[CHRISTEL 92]. Incremental changes made to the requirements can be checked for consistency against these underlying rationale.

Issues-positions-arguments of the gIBIS method are used for capturing the rationale behind the requirements as its framework is well suited for this purpose. Domain analysis and its models, such as features model and entity-relationship model, are also very useful in this phase. Domain analysis is nothing but the definition of features and capabilities common to systems in advance of development of system [CHRISTEL 92]. The entity relationship model is useful for communicating to the developers the issues for end-users, and the features model is useful for communicating to the end-users the issues for the developers [DEEPAK 98].

3.6.4 Prioritization and Planning

As the name suggests, "The goal of the prioritization phase is to arrange the requirements in order of relative importance from the view of the client and view of the developer" [MILLER 93]. The logic behind the execution of this phase is to make considerable savings in terms of time and cost due to changes in the inevitable requirements during the development of the system. The activities of this phase consist of a review of the requirements and arrangement of them based on mission criticality, cost, dependency, user needs and ability of the requirements to be incremented [MILLER 93], [CHRISTEL 92]. The Quality Function Deployment (QFD) method is an ideal technique for prioritizing the user requirements gathered in the earlier phases of the process model.

The JAD is applied for the framework for this phase. In the preparation stage QFD inputs are organized for the participants of the meeting. QFD inputs may come from the requirement models, objectives, goals, and constrains created in the earlier phases of the elicitation process model. The JAD session, is used to construct the QFD matrix from its inputs based on the "wants" of the user community and the "hows" of the developer community [DEEPAK 98]. "The 'wants' and 'hows' create the two community's desires and abilities" [MILLER 93]. In the final phase of the JAD process, the completed QFD matrix is evaluated by the stakeholders to finish the prioritization phase of the process model [DEEPAK 98].

3.6.5 Integration and Validation

The goal of this phase is to "reduce the conflicts found in the requirements, to address completeness and to validate the requirements" [MILLER 93]. This phase checks for completeness and correctness of requirements by filling in uncompleted requirements and consistency, conflict, and validation checking to determine if the requirements meet the original goals, objectives, and constraints from the fact-finding phase of the process model. Outputs of this phase are a set of requirements. If these requirements are complete then the Requirements Elicitation process is considered as done, but if they are incomplete then more iterations through the process model are needed to complete the requirements [DEEPAK 98].

The techniques such as JAD that promote an improved definition of scope and reduced chance for future requirements changes stress that integration of multiple views should occur through the involvement of all the affected communities. If the final integration and validation is performed by the developer community it could be viewed as the developers' interpretation of the requirements. In such situation, a shared ownership among the developer community and the user community would be lost and the purpose of requirement engineering would be defeated [CHRISTEL 92].

There are many ways to tackle this problem. The primary contribution of the JAD technique is in its use as "a means to validate information already gathered" in earlier phases [CHRISTEL 92]. The integration tasks are usually performed by the requirements analysts. The analysts in the preparation stage of JAD process organize and package all the documents and models from the previous phases of the process model and then review these documents and models for consistency, completeness, and validity. In the

JAD session any open issues are resolved and the priority of the requirements are reviewed. A decision on how to proceed is determined based on this review of requirements. The decision is either to proceed to the next step in the system development or to remain in the Requirements Elicitation phase and reiterate through the process model [DEEPAK 98].

CHAPTER 4

REP1 IMPLEMENTATION

4.1 Introduction

Requirements Engineering is the very important step for successful accomplishment of the system or project. Requirements Elicitation, considered to be the first major step in Requirements Engineering, is the focus of this thesis. Requirements Elicitation mainly deal with communications among different people involved in the project under development or consideration. The participants could be within an organization undertaking the project or external personnel whose expertise is considered the valuable for the success of the project. In Requirements Elicitation process, various stakeholders or users need to be able to convey their requirements to the developers, and the developers need to be able communicate their understanding and generate feed back to the end users for validation. Currently employed Requirements Elicitation techniques produce a lot of documentation and require a lot of communication between and among stakeholders of a product and the developers of the product. It also consumes lot of time. Communication costs can prove themselves very costly, particularly in these days where speedy completion of a project is utmost importance from the business point of view. Easing communications between stakeholders and developers makes the process of eliciting requirements easier, leading to better requirements specification. This eventually leads to the development of a better product. The Internet is used as a vehicle for easing communications between stakeholders and developers. The Internet provides freedom to the project members from coming physically to the meeting place and thus helps reduce considerable commuting costs. It also saves the invaluable time of the other team

90

members, as they don't have to wait even for a single person. Using the database and database inference engine, documentation cost can be reduced considerably. The REPI is implemented via client/server technology. The server used here provides simultaneous access of the database to the team members. Thus it helps reduce the cost of running multiple copies of the document. It is hoped that applying Internet technologies will ease the process for Requirements Elicitation.

Several approaches for automated Requirements Elicitation exist; the approaches taken are computer-assisted group processes, automated analysis of documents, automated requirement verification, and CASE prototyping [PLAYLE 96]. This chapter describes the application of Internet technologies for the problems of Requirements Elicitation using SEI's framework for Requirements Elicitation. In this chapter, the Requirements Elicitation Process through Internet (REPI) is described in detail. This thesis is a first step towards the implementation of REPI prototype. REPI is a GroupWare type system that uses the Internet as the meeting place for the group to meet for the purpose of eliciting requirements. Many Internet technologies could be used to develop a web site. Each technology has its own learning curve and its own benefits and limitations. The World Wide Web (WWW) technology of the Internet is used as the platform for this system. REPI is being implemented with the two of the hottest technologies of today's technological era, namely: Java and Client/Server architecture.

4.2 A Word about REPI Implementation

This thesis is the first attempt to implement the prototype developed for the Requirement Elicitation. The thesis follows the guidelines set by the REPI prototype mentioned in [DEEPAK 98]. An attempt is made is to implement the functionality of the REPI prototype. The following sections are heavily influenced by [DEEPAK 98]. The implementation of REPI prototype can broadly be divided into two phases: front end and back end. The front end deals with the presentation of the forms that direct both communities to enter requirements in a format specified by Software Engineering Institute's guidelines for Requirements Elicitation. The back end deals with the storage and retrieval of these requirements to and from the database. Java has been chosen for the front end GUIs while Oracle is selected for back end database. In order to use the prototype developed in [DEEPAK 98], CGI scripts were required to be used. CGI scripts are slow in performance and have many loopholes regarding the security of the back end system. Security becomes one of the predominant factors particularly when the project under consideration is to be kept restricted to authorized persons only. The loopholes in CGI scripts may lead to the corruption of the database. Java provides protection against security violations. Java bytecode verifier verifies the bytecode downloaded over the network and prevents unwholesome applets to access the resources on the local host on which it is running. Also, Java applets are not allowed to communicate with the database server directly for the security reasons. This provides the safety against the corruption of the database. Thus, Applets are allowed to communicate with the database server in a controlled manner. This leads to the development of the three-tier client/server architecture. Moreover, only those applets are allowed to communicate with the

application server, also written in Java, that are downloaded from the host the application server is running on. This discourages the hackers to play in their way with the application server and thereby the ultimate database server. Java applets or applications are compiled to bytecode by the Java compiler and then this bytecode is interpreted by the Java interpreter. However, since the Java code is precompiled, it is considerably faster in performance as compared to CGI scripts. Using Just-In-Time (JIT) compiler the performance of Java programs can further be improved. The user and developer communities store or retrieve their opinions to or from Oracle database using Java applets. The Java applets prepares appropriate SQL queries from the data entered by the team members and passes this information to the application server running on the host the applet originated from. The application server provides services to the multiple applets concurrently using the thread mechanism. The server reads the request made by these applets, parses these requests and invokes the appropriate methods. These methods communicate with the database server on behalf of the applets by passing their request to the database server. The database server provides the requested services to the application server. The application server then responds to the applets waiting for the server's response. The subsequent sections of this chapter deal with the description of the phases of REPI and their implementation.

4.3 REPI Web Site Description

The REPI web site, which is developed to be used as the platform for eliciting requirements in a distributed, and an asynchronous manner. It is distributed because members of a given project need not be at the same location neither do they have to come to a meeting place physically. They contribute their information electronically using the Internet and the REPI web site. Requirements and other information are collected and stored in a database on the server. As different users from different locations log in, the collected knowledge of the whole project is displayed to each and every authorized member of the project. It is asynchronous because users need not be logged in at the same time and these people are not only separated by distance; they could also be separated by time zones. It is concurrent because the team members can simultaneously access the central database where all the information about the requirements has been stored. Individual users are working alone, at their own pace and at their time. But the whole project's work is collected and the most up to date information is displayed to all the users at the same time when users request them simultaneously.

LOGIN Screen

Users' Main Menu — Developers' Main Menu

Users' Main Menu:

SEI Phase Menus | Utility Menus | Help Menus

SEI Phase Menus:
- Fact Finding
- Gathering and Classification
- Evaluation and Rationalization
- Prioritization and Planning
- Integration and Validation

Utility Menus:
- Read Messages
- Send Messages
- What's New
- Todo Tasks

Help Menus:
- Fact Finding
- Gathering and Classification
- Evaluation and Rationalization
- Prioritization and Planning
- Integration and Validation

Developers' Main Menu:

SEI Phase Menus | Utility Menus | Help Menus

SEI Phase Menus:
- Fact Finding
- Gathering and Classification
- Evaluation and Rationalization
- Prioritization and Planning
- Integration and Validation

Utility Menus:
- Read Messages
- Send Messages
- What's New
- Todo Tasks

Help Menus:
- Fact Finding
- Gathering and Classification
- Evaluation and Rationalization
- Prioritization and Planning
- Integration and Validation

Figure 4.1: REPI Web Site Structure Overview Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

The REPI web site is organized as a series of pages bifurcated into two primary menu pages: Users' Main Menu and Developers' Main Menu, as shown in Figures 4.1. The two primary menu pages divide the complete set of web site users, for a given project, along their lines of responsibility: the client side and the development side. The client side people are responsible for providing the requirements for a product, while the development side people are responsible for understanding the product. Each of these two primary menus has been divided into three sets of menus: SEI Menus, Utility Menu, and Help Menu. The SEI menu has been classified into the five phases of the SEI's Requirements Elicitation process model: Fact-Finding, Gathering and Classification,

Evaluation and Rationalization, Prioritization and Planning, and Integration and Validation. The Utility menu has been divided into four parts: Read Messages, Send Messages, What's New, and Todo Tasks. The Help menu like SEI phase menu consists of five divisions for both the communities: Fact-Finding, Gathering and Classification, Evaluation and Rationalization, Prioritization and Planning, and Integration and Validation. Figures 4.2 and 4.3 show the structures of the client side community and the development side community, respectively. A "Login" page serves as the entry point on the REPI web site, this page is used to branch off into the two primary menu pages. The "Login Screen" is described in Section 4.3.1. The web pages branching off from the primary menu structures are described in Sections 4.3.2 to 4.3.8.

```
                          ┌──────────────┐
                          │   Users'     │
                          │  Main Menu   │
                          └──────────────┘
```

Fact Finding	Gathering and Classification	Evaluation and Rationalization	Prioritization and Planning	Integration and Validation
Identify relevant people	List Requirements	Perform Abstraction	Prioritize the Requirements List	Address Completeness
Describe the Problem	Add Requirements	Capture Rationale		Validate Requirements
Define Goals				Obtain Authorization
List Mission Scenarios				
Identify Similar Systems				

Figure 4.2: REPI Web Site Client Side Structure Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

Developers'
Main Menu

Fact Finding

Gathering and Classification

Evaluation and Rationalization

Prioritization and Planning

Integration and Validation

Identify domain experts

Identify domain models

Conduct technological survey

Assess constraints

Classify Requirements

List Requirements

Add Requirements

Risk Assessment

Feasibility Analysis

Cost/Benefit Analysis

Prioritize Requirements

Plan development stages

Identify architectural models

Resolve Conflicts

Figure 4.3: REPI Web Site Developer Side Structure Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis. NJIT. January 1998

The screen shots shown in this chapter are from the appletviewer supplied by the Java tool kits versions 1.1.2 and 1.1.3 as displayed on the Microsoft Windows NT 4.0/ Windows 95 platforms. Due to unavailability of support for Java version 1.1.2 from the popular browsers Netscape Navigator 4.04 and Internet Explorer 4.03, these snap shots have been taken by running the Java applets through appletviewer. The appendices also show several snapshots taken on the UNIX platform. As the REPI web pages developed for this thesis is a first attempt to implement the REPI demo version, many of the menu pages described above have not been implemented. Also, since the database the collects all the requirements information is under development, the back-end functionality to be described are not implemented in all the applets at this time. However, the back-end functionality to be supported by the client/server architecture has been implemented for demo purposes in one or two applets. This can be easily extended to all of the applets as

the work of the database development will be over. The next chapter describes about the implementation of this back end functionality in detail.

4.3.1 Login Screen

It is very difficult to trace the human personality on a large network. Specially, with the parabolic growth of the Internet all over the world this task seems very difficult if not impossible because currently there is no restrictions on the use of the Internet. [AL-RAWAS 96] describes the problem of requirements traceability as the inability to trace the human source for the actual requirements and their related information. Requirements traceability is very important particularly, during the later stages of the Requirements Engineering, for validation and review. Traceability is also important during later stages of the software development cycle, if changes need to be made to the requirements or if more detailed information is necessary. [AL-RAWAS 96] shows that most requirements are only linked to people by their job titles, user groups or departments. In the long-term projects, people could be prompted. It is also possible that people change groups or even companies. In such liquid situation, Requirements linked to an individual's name can serve traceability better than other forms of linkage.

The REPI web site could be located either on an internal server within the companies Intranet or on a server on the Extranet that can be shared by many cooperating organizations. The REPI web site can be implemented even on the public server connected to the Internet, in cases where requirements are elicited from the general public or from people who work for many different organizations. All this connectivity requires some level of security.

The main page for the REPI web site provides a simple user id and password based security for the REPI web site. Initially project managers from the client side organization and the development side organization will be provided with a management level user id and password. These managers will give authority to the people involved in the project using the "Identify Domain Experts" task of the Fact-Finding Phase. As people are identified, they will be given either user level or management level access as required. They will also be partitioned as either client side users or development side
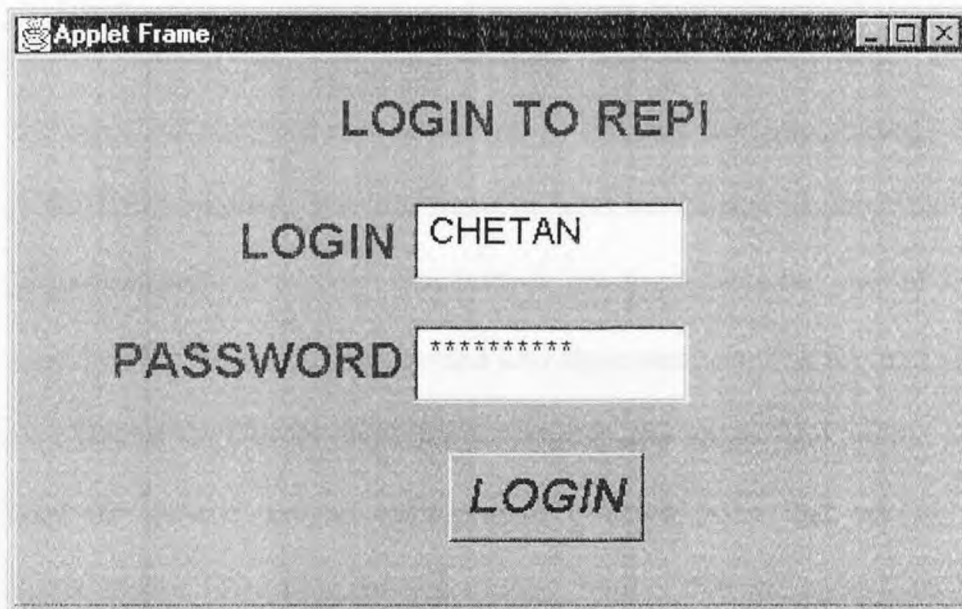
Figure 4.4: REPI LOGIN Screen

users. The back end process for this page should either load the "User's Main Menu" or the "Developer's Main Menu" based on the type of login id and password entered by the user. Clicking on the "LOGIN" button will collect the user name and password and verify against those stored in the back end database. Upon verification, it will allow the authorized person to access REPI web site. As the back end database is under development this process is not implemented in this version of the REPI web site.

4.3.2 Menu Screens

As mentioned in section 4.3 the REPI project is divided between user-oriented forms and the developer oriented forms. Two sets of menus are provided for these two types of project members, client community and development community. Two menu screens: the "User's Main Menu" screen and the "Developer's Main Menu" screen have not been modified in this thesis.

Both sets of menu screens are designed to be as similar as possible. The visual appearance and the front-end behavior of the two pages have been kept identical for the sake of consistency. The real difference between the two pages are in the messages displayed and the back end behaviors. As the user navigates through the different pages of the REPI web site, the title frame reflects the current phase of the SEI Requirements Elicitation process model. The bottom left frame lists the current menu items for the screen. This frame itself is divided into three sections. The top part provides a link into each task of the current phase for the current user group. The middle part of this frame is used for generic project management oriented tasks that are not part of the SEI Requirements Elicitation process model. Rather, they are generic useful items which in order to make using the REPI web site easier. The last part is used for generic web site related links; specifically links such as "Main Menu," "Help," and "Logout" are provided [DEEPAK 98]. Currently, this thesis does not provide the support for this functionality. The bottom right frame is used as the main display area for the actual forms needed in each of the tasks. The contents of this frame are completely dependent on the current phase, the current task and the type of user logged in to the REPI web site.

4.3.3 Users' Tasks

The users here mean the customers as well as the end-users of the system or product under development. As mentioned in the section 4.2, according to SEI's guidelines, users' tasks in Requirement Elicitation can be divided into five phases. Each phase is explained in the following sub sections.

4.3.3.1 Fact-Finding Phase

The main purpose behind the Fact-Finding phase of the SEI Requirements Elicitation process model is to examine the context of the project and the product to be developed. The client community of the project examines their organization and reasons for the system or project. The development community of the project examines the technology to be used and the domain of the product to be developed. Table 4.1 lists the mapping between the SEI tasks for the Fact-Finding phase, as described in [CHRISTEL 92], and the REPI web site tasks for the Fact-Finding phase.

Table 4.1: Users' Fact-Finding Phases of SEI and REPI Prototype Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Client Side Tasks	Identify relevant parties.	Identify relevant people
	Determine operational and problem Context	Describe the Problem
		Define Goals
	Identify similar systems.	List Mission Scenarios
	Perform context analysis	Identify similar systems.

4.3.3.1.1 Identify Relevant People: Any Requirements Elicitation methodology requires a set of people to work with. This is the task of the project managers to identify the potential stakeholders of the project who could contribute to the success of the project. The people identified might be end users of the actual product to be built, the customers or owners of the product to be built, the people who actually authorized the project or are going to pay for it, or the people whose expertise has been considered valuable for the successful completion the project. They could be management level people, such as supervisors of end users, whose input could be of use for the project's success. The applet displays a simple form for users to enter the stakeholders name and basic contact information. It also has an option checkbox group to categorize the stakeholders into one of three pre-defined categories: "End User," "Customer," or "Management" [DEEPAK 98]. Figure C.3 shows the snap shot of this task taken on Microsoft Windows NT 4.0 platform.

4.3.3.1.2 Describe the Problem: To know the right problem is to solve it half. The identification of the pertinent problem is the first step towards solving that problem. Many times it is realized that the solution found was not the one they were looking for. This happens when the problem is not defined in a clear-cut way. This task allows the users to describe the perceived problem from their point of view. This allows each stakeholder to define their views, possibly separate and contradictory to others' views. Figure C.6 the applet presents a text area to describe the problem to store into the database when "SUBMIT" button is clicked.

4.3.3.1.3 Define the Goal: It is always important to decide what to do before starting any activity. This task expects the users define the major goals from their point of view to be achieved by the project. Each stakeholder can enter multiple goals by repeatedly using the form and giving a separate goal name each time. This provides the developers with a list of major areas of work to be done for the project. Figure C.9 presents a text field for the goal name and the text area for the users to enter the goal description. The "*IMPORT*" button when clicked, pops up the file dialog box to import the description of the goal into the text area. This helps the users prepare and store their goal description into the files and thereby increase the probability of providing better project goal description. Clicking the "SUBMIT" button will form appropriate query and transport this query via socket to the database through client/server architecture. "CLEAR" button clears the text field and text area in order to nullify the information entered into these fields.

4.3.3.1.4 List Mission Scenarios: This task is created to identify Major scenarios for the product's use. The scenario has a name and general description associated with it. The framework of a given scenario can be divided into three steps: event, action, and reaction. A scenario name and its description can be entered through a text field and a text area, respectively. The "*IMPORT*" button pops up the file dialog box in order to bring the description of the scenario into the text area. This helps the users prepare scenario description in advance and store into the files and thereby increase the probability of getting better scenario description. The text fields have been provided to enter event, action, and reaction for a given scenario. Clicking the "SUBMIT" button will

form appropriate query and transport this query via socket to the database through client/server architecture. "CLEAR" button clears the text field and text area in order to nullify the information entered into these fields. Figure C.12 makes the above description more clear.

4.3.3.1.5 Identify Similar Systems: As the name suggests, this task is used to identify systems that are in some way, shape, form, or functionality similar to the product to be developed. This exposes the users to the potential sources of materials that can be reused, either in the form of analysis, design or perhaps the implementation level details itself. These potential sources have to be identified and categorized according to their usefulness in order to use them for the product under development. This task is not just limited to identify similarities between these identified systems and the one under consideration, but also to expose the differences between these systems and the product to be developed. In this version of REPI this phase is yet to be developed.

4.3.3.2 Gathering and Classification Phase

To capture and organize a set of requirements of the product to be developed, Gathering and Classification phase of the SEI Requirements Elicitation process model is considered useful. The client community provides the requirements based on their needs and the development community of the project classifies the requirements, provided by the client community, based on various attributes. In this phase both the communities are allowed to provide information for the requirement's attributes. Table 4.2 lists the mapping between the SEI tasks for the Users' Gathering and Classification phase and the REPI web site tasks for the Users' Gathering and Classification phase [DEEPAK 98].

Table 4.2: SEI Compared with REPI for the Users' Gathering and Classification Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Client Side Tasks	Get wish list.	List Requirements
		Add Requirements

4.3.3.2.1 List Requirements: Even though this is not the task defined by the SEI Requirements Elicitation process model, it has been added here as it can serve as a utility function useful in this phase of the process model and throughout the Requirements Elicitation effort. It allows the users to list all the available requirements. It also allows the users to filter, sort, and define their own viewpoint into the requirements database. Several commonly useful pre-defined views can also be provided here [DEEPAK 98]. In the current version of REPI, this task is not implemented.

4.3.3.2.2 Add Requirement: "Add Requirement" is one of the most important tasks of the Requirements Elicitation process model as it is the primary means of adding requirements of the users. Client side members are allowed to add or modify the requirements in this task. A requirement can have many attributes associated with it. These attributes provide supplementary information about the requirement, its relationship to other requirements and assist in requirements management [KAR 96]. A unique method of identification is needed, both for the software and for humans, to distinguish between the different requirements. Different types of categories can be

useful to classify these requirements. The "Add Requirements" page allows user to provide both the requirement and its supplementary properties [DEEPAK 98] as shown in figure C.15.

The label "REQ. ID" stands for requirements identification number and is used to identify the unique identification method. The requirement title, a user provided text string, is used as the identification method. Requirements can have different categories REPI uses several pre-defined categories, based on the problem domain, to classify the requirements. Users can define their own categories. The purpose behind this facility is to complement the requirements categories defined by REPI. But if each and every user defines his or her own category it defeats the purpose of having categories. So some method of social control or software assisted security control needs to be provided, to allow only selected users to define new categories. Others should be restricted to selecting a pre-defined category as they add new requirements. Each requirement will fall into the following four compliance levels: Mandatory, Goal, Objective, and Optional. Choice provides the facility to categorize the requirement into exactly one category. The exact semantics of these compliance levels has to be based on some external common understanding, such as a contract document. A requirement evolves during different passes through the elicitation model phases. The label "Current Status" provides this evolution. The requirements status can be such as "To Be Determined" (TBD), "To Be Reviewed" (TBR), "Defined," "Verified" and "Deleted." As we discussed in chapter 3, requirement can be classified into several broad categories such as "Functional requirement," "Non-Functional requirement," or "Interface requirement." Some of the broad categories have several sub-categories such as "Performance requirement" or "User

Interface requirement." The "Requirement Type" attribute is used for this classification including the classification for information that is deemed to be a "Design Constraint" rather than an actual requirement. A given requirement has to be verified before it is used. Different verification methods can be used to verify the requirements depending on the type of requirement and the information available on it. The "Verified By" attribute is used to indicate the type of verification method desired for a given requirement. REPI uses the following verification methods: "Inspection," "Analysis," "Demonstration," and "Test." These attributes provide additional information about the requirement. Besides all these attributes each requirement has to have a description that defines the requirement itself [DEEPAK 98].

Figure C.15 shows the REPI implementation for the "Add Requirement" task of the Gathering and Classification phase. This page is designed to display all the properties that need to be entered for a requirement to be fully defined. A drop down choice box is used to list the available requirement identification number. A text field facilitates the user to enter the requirement title. A choice list labeled "CATEGORY" is used to list the pre-defined requirement categories. A text field is provided for the users to enter their own category. The next row provides the list of compliance levels and current status indicators. Either an end user or a management level user can select the proper compliance level needed for this requirement and define its current status. In the Next a text area is presented for the user to enter the requirement description itself. Requirements can come in several forms, for example a graph or table might be a requirement specifying the need to meet some performance level. A picture or some other multimedia element can be used to provide a sample for some quality requirement.

A requirement might refer to a standards document specifying the need to meet that standard. To handle these types of requirement an import button is provided. When clicked, a file dialog box will be popped up to bring this information into the text area. This external file can contain any type of data and can be in any format. The next row allows the selection of any one of the four requirements types because these requirement types are grouped using Java's checkbox group. The Non-Functional requirement type and the Interface Requirement types are further categorized. A choice list has been used for each of these two to select Non-Functional requirement type and Interface requirement type. The second to last row of the applet form displays the different types of verification methods used by REPI. Once again, a grouped checkboxes are used to select the verification method. The two buttons "SUBMIT" and "CLEAR" are used to submit and reset the requirement and its attributes respectively.

The details of the above attributes are dependent upon the source used. For example, in the area of requirement categorization, several different possibilities are listed: "Program Requirement" versus "Product Requirement" and "Primary Requirement" versus "Derived Requirement" [KAR 96] and [HARWELL 93]. A Requirement application attribute identifies the object of a requirement with several different types of parameters [HARWELL 93]. The "Product Requirement" can be "Qualitative" or "Quantitative." The "Program Requirement can be subcategorized into "Task," "Compliance Evaluation," and "Regulatory." According to [HARWELL 93] requirement's compliance level can be "Mandatory," "Guidance," or "Information," while [KAR 96] suggests "Mandatory" and "Goal or Objective."

4.3.3.3 Evaluation and Rationalization Phase

The purpose behind the Evaluation and Rationalization phase of the SEI Requirements Elicitation process model is to expose inconsistencies in the gathered information and for "determining why the information has been expressed as a requirement" [MILLER 93]. Table 4.3 lists the mapping between the SEI tasks for the Evaluation and Rationalization phase, as described in [CHRISTEL 92], and the REPI Users' tasks for the Evaluation and Rationalization phase.

Table 4.3: SEI Compared with REPI for the Users' Evaluation and Rationalization Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Client Side Tasks	Perform abstraction to answer questions of the form "Why do you need X?"; this in effect moves from statements of "how" to statements of "what."	Perform Abstraction
	Capture rationale to support future requirements evolution.	Capture Rationale

4.3.3.3.1 Perform Abstraction: This task allows users to describe their requirements in somewhat more detail. The main purpose of this task is to extract underlying rationale for a given requirement. It has been proposed to answer the question: "Why do you need this requirement?" To fulfill this purpose the web page allows the users to display each requirement, along with its description, and answer this question. For each selected requirement from choice list, the user will be represented with the title, category and detail description of the requirement. Based on this a user will be asked to enter or import the need for the requirement he or she had asked for. With the clicking of the "SUBMIT"

button the answer of the question "Why do you need the requirement?" will be collected into the remote database. As usual, "CLEAR" button will reset the applet form. At present the database is under development and therefore the back end support for this task is not provided. Figure C.16 shows the form for Users' Perform Abstraction task.

4.3.3.3.2 Capture Rationale: This task is similar to the previous task, in the sense that this task also requires the user to enter detail information about each requirement. However, in this task, the user is asked to provide underlying rationale behind the requested requirement. The rationale for a requirement provides data that support the requirement. "The supporting data may include the reason or reasons a requirement is needed; any assumptions made at the time the requirement was formulated . . ." [KAR 96]. The web page for this task is very similar to that of the previous task. However, this task has the more impact on the justification of the requirements as compared to the one shown in figure C.15. Figure C.17 shows the snap shot of Capture Rationale task taken on Microsoft Windows NT 4.0 platform.

4.3.3.4 Prioritization and Planning Phase

The Prioritization and Planning phase of the SEI Requirements Elicitation process model determines "the relative importance of each requirement and the relative order the requirements should be addressed in" [MILLER 93]. Table 4.4 lists the mapping between the SEI tasks for the Users' Prioritization and Planning phase, as described in [CHRISTEL 92], and the REPI web site tasks for Users' the Prioritization and Planning phase.

Table 4.4: SEI Compared with REPI for the Users' Prioritization and Planning Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Client Side Tasks	Determine criticality.	Prioritize the Requirements list

4.3.3.4.1 Prioritize Requirements: It is a unique characteristic of the nature that no two elements can be exactly same. Similarly, in any system not all requirements can be the same. Some requirements are necessary while some can be auxiliary. This task is used to prioritize the requirements by considering their relative importance with respect to each other. When the conflicting requirements comes into picture during the requirement evolution through the REPI phases, those which are insignificant or auxiliary can be sacrificed for the 'must' requirements. The users are required to set a priority level for each of the requirement listed, indicating its importance for the project from their point of view. Thy are also required to judge the level of understanding obtained on the given requirement by the development team and by the users themselves. This indicator can be used to see which requirements are less understood and thus require more study or more detailed explanation. The REPI web site lists the available requirements along with two drop down boxes to set these indicators: priority level and understanding level. The priority level choice boxes indicates the level of importance given to the requirement by the users, while the understanding level choice box indicates the understanding obtained

by the project team on the given requirement. Currently, this page is a static one. Once the database will be developed, this can be made dynamic.

4.3.3.5 Integration and Validation Phase

There is difference between verification and validation. "Verification means the check that specifications describe the system right. Whereas validation means the check that specifications meet the client requirements" [HOFFMANN 93]. The Integration and Validation phase of the SEI Requirements Elicitation process model determines the validity of the gathered information and it is also responsible for obtaining missing information. Table 4.5 lists the mapping between the SEI tasks for the Users' Integration and Validation phase, as described in [CHRISTEL 92], and the REPI tasks for the Users' Integration and Validation phase.

Table 4.5: SEI Compared with REPI for the Users' Integration and Validation Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Client Side Tasks	Address completeness issue	Address Completeness
	Check that requirements are in agreement with the original goals	Validate Requirements
	Obtain authorization to move to the next step of development	Obtain Authorization

4.3.3.5.1 Address Completeness: The purpose of this task is to address any requirements that might not have been completely defined in the earlier phases of the Requirements Elicitation process. For example unknown or less understood requirements can be created and marked, during the "Gathering and Classification Phase," as "To Be Determined." These "TBD" requirements, as it is commonly listed, have to be eventually defined before the requirements stage of the development process is finished. The "Address Completeness" task of the "Integration and Validation Phase" is used for this purpose [DEEPAK 98]. The structure of this form is same as that of "Add Requirement" phase described above.

4.3.3.5.2 Validate Requirements: Before a set of requirements is converted into requirement specification documents, they have to be verified and validated. This task is used for the purpose of validating requirements and verifying that they are in agreement with the originally stated goals for the project. This part is not implemented in this thesis.

4.3.3.5.3 Obtain Authorization: This task is the final step of the client community that indicates the finished status of the Requirements Elicitation process. By "signing" this form the client community indicates that the requirements have been properly elicited from them and that they authorize the developers to proceed for the next step of the development process [DEEPAK 98].

4.3.4 Developers' Tasks

The developers are responsible for the development of the system under consideration. Therefore it becomes they responsibility to perceive the need of the customers. Hence in SEI's framework of Requirement Elicitation process it is important to involve the development community. Analysts are responsible for bridging the communication gap between the user community and developer community. Sometimes the organization deploys the analysts in the project under development. In that sense, analysts fall in the developer community. As mentioned in the section 4.2, according to SEI's guidelines, developers' tasks in Requirement Elicitation can be divided into five phases. Each phase is explained in the following sub sections.

Table 4.6: SEI Compared with REPI for the Developers' Fact-Finding Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Development Side Tasks	Identify domain experts.	Identify domain experts.
	Identify domain and architectural models.	Identify domain models.
	Conduct technological surveys.	Conduct technological surveys.
	Assess cost/implementation constraints.	Assess constraints.

4.3.4.1 Fact-Finding Phase

Table 4.6 shows the developers' tasks for the Fact-Finding phase of the Requirement Elicitation process. The table shows the similarities between SEI framework and REPI framework. The Fact-Finding phase for the developers' is divided into three tasks: Identify domain experts, identify domain models, conduct technological surveys, and assess constraints. These tasks are described in the following sub-sections.

4.3.4.1.1 Identify Domain Experts: This task is similar to the user-oriented task "Identify relevant people" executed in the same phase of the SEI's Requirements Elicitation process model. This task identifies the domain experts and the development experts for the project. While the user-oriented task identified people to contribute their specific needs for the product, the developer-oriented task identifies people to contribute the specific needs for developing the product in a given domain. The application expert or the domain expert is expected to have knowledge in the general area of the product. For example a project that develops an application for a medical might require domain experts with knowledge about the laws that apply in the healthcare industries. The development expert has knowledge in the product development areas. For example a project that develops the software for Medicare or medicate application through telephone lines might need experts in the areas of network and network security issues.

Figure D.3 displays an applet form for the project manager or leader on the development side to enter the expert's name and basic contact information such as name, phone no, e-mail address etc. It also has an check box for categorizing the expert into one of two pre-defined categories: Application Expert or Development Expert. Here a

person to be identified can be Application Expert as well as Development Expert. Therefore, these two attributes have not been kept in group box. A click on the "SUBMIT" button will store this identification into the back end database. "CLEAR" button will clear the form.

4.3.4.1.2 Identify Domain Models: It is important to recognize the different models that are pertinent or useful in someway for the development of the project at hand. This task allows developers to identify these models that will be used during the product's development. The web page for this task allows the developers to provide information about the domain model and the architectural model. The domain model deals with the information about the product's general area while the architectural model deals with the information about the design of the product [DEEPAK 98]. For example if the product to be built is an online transaction of the stock exchanges for a financial firm then the domain model will contain the information about how the transactions for stock exchanges are applied in the stock industries. The architectural model in this case might contain information about using client/server model through the network for the development of front end and back end modules. Figure D.6 shows the snapshot of the REPI page for this task that accommodates two text areas to enter the information about these models. This page also contains an "IMPORT" button to deliver already prepared model information from the external file that contains the necessary information.

4.3.4.1.3 Conduct Technological Survey: This task is used for technological survey information about the technologies that can be applied in the development process of a product. For example if the project's purpose is to provide the software for the online transaction for stock exchanges then this task can be used to enter information about the Internet security technologies such as Java, CGI scripts, Secure Sockets Layer (SSL) protocols etc can be used secure web applications. The REPI web site applet provides a text field and a test area to enter a name of the technology survey and the detail of the survey information respectively. An import button is provided to link into an external file that contains the survey information. A "SUBMIT" button will submit the query to the database and a "CLEAR" button will clear the form. Figure D.9 makes shows the page for this task

4.3.4.1.4 Assess Constraints: This task is gathers information about the constraints that are imposed by the client community. A constraint can be defined as an implied requirement that limits the design, solution or implementation level choices of the system to be developed. The budgetary limitations are good examples of a constraints. A user wants that a software system development must be done in Java. In such a case the developer community needs to gather information about this constraint's implications. One such implication might be the need for a developer familiar with the Java Application Programming Interface (API). As shown in figure D.12, the web page for this task contains a drop down choice menu to select the constraint, a text area to enter the detail information about the above selected constraint.

4.3.4.2 Gathering and Classification Phase

The development side members of the project classify the requirements based on various attributes. In this phase of Requirement Elicitation process, the developer community performs three tasks: Classify Requirements, List Requirements, and Add Requirements. Table 4.7 lists the similarities between the SEI tasks for the Developers' Gathering and Classification phase, as described in [CHRISTEL 92], and the REPI web site tasks for the Developers' Gathering and Classification phase.

Table 4.7: SEI Compared with REPI for the Developers' Gathering and Classification Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Development Side Tasks	Classify wish lists.	Classify Requirements
		List Requirements
		Add Requirements

4.3.4.2.1 Classify Requirements: The requirements entered by the users are required to be classified from the development perspectives by the developer community. This task allows developers to properly categorize these requirements so that a detailed requirements hierarchy is generated. Even though requirements have been classified in the "Add Requirements" task of the users in the Gathering and Classification phase, they may not be properly categorized because of many reasons. For example, full information about the requirements may not be available at the time it is added or most end users are not properly qualified to classify requirements based on their type such as, "Functional," "Non-Functional," or "Interface" or most end users are not able to judge a requirement

as a "Design Constraint" or as an actual requirement. Also requirements might go through several versions and evolve slowly, with just the title given, to a fully defined and categorized stage. Due to such an evolution requirement can not be classified properly at the beginning. Developers might need to consult with users if they need to change a requirement's category to produce a requirements hierarchy. All available information about each requirement can easily be viewed by selecting the requirements detail levels. Developers are assisted in categorizing requirements so that they can categorize new requirements by viewing already categorized requirements. At present the support for this page in this thesis in not provided.

4.3.4.2.2 List Requirements: Since this task is exactly same as the one for users, it is not described in this section. This task is not implemented through this thesis.

4.3.4.2.3 Add Requirement: The "Add Requirement" for the Developer's side of the REPI web site is the same task described in the User's task of the REPI web site. For the sake of avoiding redundancy, this section is not re-discussed here.

4.3.4.3 Evaluation and Rationalization Phase

As mentioned earlier in the section 4.3.4.4 of this chapter, the Evaluation and Rationalization phase of the SEI Requirements Elicitation process model is responsible for exposing inconsistencies in the gathered information and it is also responsible for "determining why the information has been expressed as a requirement" [MILLER 93]. Table 4.8 maps the SEI tasks for the Developer's Evaluation and Rationalization phase,

as described in [CHRISTEL 92], to the REPI web site tasks for the Developer's Evaluation and Rationalization phase.

Table 4.8: SEI Compared with REPI for the Developers' Evaluation and Rationalization Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Development Side Tasks	Perform risk assessment.	Risk Assessment
		Feasibility Analysis
		Cost/Benefit Analysis

4.3.4.3.1 Perform Risk Assessment: The task of Risk Assessment allows developers to keep track of the risks associated with each requirement. For the online transactions for stock exchange, the requirement might state the need for the web application to be compatible with older browsers. But if these older browsers do any other form of secure transactions a certain amount of risk is associated with the requirement of using older browsers. Risks associated with such requirements can be described in this task [DEEPAK 98].

Figure D.14 shows the REPI web site structure. A choice list is used to list requirements. The text area next to this list will display the selected requirement's description. The second row contains the text area in which the developers can enter the information about risks associated with this requirement or can import the risk assessment associated with the selected requirement using "*IMPORT*" button. A "SUBMIT" button is provided to enter the risk assessment into the database and a "CLEAR" button to clear the form.

4.3.4.3.2 Perform Feasibility Analysis: A requirement's feasibility can be entered into the database through the task of Feasibility Analysis. It is quite possible that not all the requirements demanded by the users are feasible due to some constraints set up by the users themselves. The development time needed to meet the requirement might be longer than the time allowed for the project completion. This task is used to enter such feasibility. Or this task could be used to record information that states the conditions under which a given requirement can be met [DEEPAK 98].

Figure D.15 shows the REPI web site structure for this task. A choice list lists the requirements entered by the users. The text area next to this list will display the selected requirement's description. The second row contains the text area through which the developers can enter the feasibility of the selected requirement or can import the feasibility of the selected requirement using "*IMPORT*" button. A "SUBMIT" button is provided to enter the risk assessment into the database and a "CLEAR" button to clear the form.

4.3.4.3.3 Cost/Benefit Analysis: The efficiency of any system can never be infinite. In order to meet a requirement, one has to put in pertinent resources. It is also possible that, from developers' point of view, some benefits can be obtained by releasing or putting more constraints on a system or product to be built. Analysis of costs associated with a given requirement and the benefits that can be derived from that requirement is described in this task of Cost/Benefit Analysis. Figure D.16 shows a front end for the cost/benefit analysis. Rest of the description is same as that of section 4.3.4.2.2.

4.3.4.4 Prioritization and Planning Phase

The Developers' Prioritization and Planning phase is divided into three tasks: prioritize requirements, plan incremental development stages, and identify architectural models as shown in table 4.9. Table 4.9 lists the mapping between the SEI tasks for the Developers' Prioritization and Planning phase, as described in [CHRISTEL 92], and the REPI web site tasks for the Developers' Prioritization and Planning phase.

Table 4.9: SEI Compared with REPI for the Developers' Prioritization and Planning Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Development Side Tasks	Prioritize requirements based on cost and dependency.	Prioritize the Requirements
		Plan incremental development stages
		Identify architectural models

4.3.4.4.1 Prioritize Requirements: This task allows developers to prioritize the set of requirements and is very similar to the "Prioritize Requirements" task for the User oriented task. However they differ in a way prioritization is being done. While the users prioritize requirements based on priority level and understanding level, the developers prioritize the requirement based on cost level and dependence level attributes. The cost level implies the costs associated with a requirement. The more complex requirement the more costs in development time. The dependence level describes the number of associate requirements for a given requirement is described by requirement dependence level. A requirement is "refers to" if a change in this requirement causes changes in other

requirements. A requirement is "referred to" if changes in other requirements bring change in this requirement. With increase in the dependence level the cost of meeting that requirement increases.

Figure D.17 shows the implementation of this task. An attempt is made to ease the perception of the requirement dependency. A drop down list box displayed in the second row, listing the full set of requirements in the database. The list on the left side of the first row lists the "refer to" requirements, while those on the right side of the first row are "referred by" requirements for the requirements selected on the centered choice list. Based on this vision of the dependency, developers can easily set both the cost level and the dependence level [DEEPAK 98].

4.3.4.4.2 Plan Incremental Development Stages: This task sorts the full set of requirements into subsets based on its attributes as judged by both the communities. The input for this task is the output of the "Prioritize Requirements" task from both the sides. The users contribute to the level of importance and the level of understanding, while the developers contribute the cost level and the dependency level. The combined value is calculated based on these levels. The REPI web site would sort the full set of requirements according to these combined values. This page is not implemented in this version of REPI.

4.3.4.4.3 Identify Architectural Model: This task helps developers identify architectural models that support the incremental development stages. Figure D.18 shows the structure of the REPI web site implementation. A text area is provided to enter the information about architectural model that can be used for the task 2 of the Prioritization

and Planning phase. "*IMPORT*" button brings up the file dialog box that will enter the identified architectural model that supports incremental development stages. "SUBMIT" will submit the architectural model, while the "CLEAR" button will clear out the applet information.

4.3.4.5 Integration and Validation Phase

The Developers' Integration and Validation phase of the SEI Requirements Elicitation process model has been distributed in three tasks: Table 4.10 lists the mapping between the SEI tasks for the Integration and Validation phase, as described in [CHRISTEL 92], and the REPI tasks for the Integration and Validation phase. The user oriented or client side tasks are listed first and then the development side tasks are listed.

Table 4.10: SEI Compared with REPI for the Developers' Integration and Validation Phase Source: "Applications of Internet Technology for Requirements Elicitation." Deepak Pandit. A Master's Thesis, NJIT. January 1998

	SEI's Tasks	REPI's Tasks
Development Side Tasks	Resolve conflicts	Resolve conflicts

4.3.4.5.1 Resolve Conflicts: After this final phase of Requirement Elicitation process model is over requirement specification preparation stage of Requirement Engineering starts. The future development of the system is carried out according to the requirement specification documentation. The output of Requirement Elicitation is the input of requirement specification stage. Hence it becomes more important to validate the

requirements by resolving the conflicts that may exist in the requirements demanded by the user community. This task is created to help the developers validate the requirements and resolve any conflicts found in them. Based on comments made by the users about their requirements the developers are required to assess the validity of the requirement and resolve any conflicts that might arise from the different viewpoints presented by the users [DEEPAK 98].

The REPI web site demo uses a drop down list box to present a list of requirements tagged as unverified or unresolved. Next to this the requirement's category is displayed and its description is displayed below that. A table is used to display the list of users who commented on this requirement. The left column of the table displays the user id, the next column displays the date and time the comment was made. The last column displays the actual comment made by that user. Below this table, a text area is provided where the developer can resolve any conflicts found.

4.4 Critique on REPI

This section evaluates the REPI web site from two perspectives: design and usefulness. The main purpose of the design perspective of REPI is make it easy for the people to understand it, while the objective of usefulness perspective is to improve the communication between developers and users to complete the project effectively and efficiently. These purposes are explained in detail in [DEEPAK 98]. This thesis is the first attempt to implement the REPI prototype. The structure for most of the web pages developed for this thesis has been kept similar to that of [DEEPAK 98]. Due to the use of the Internet it is anticipated that the communication costs for the Requirement Elicitation

process model will be greatly reduced as the team members will have freedom from the constraints of time and place. Due to use of Java, security to the database system is also improved over the REPI that would have been implemented via HTML and CGI scripts. Java applets provides the front end GUI for the users of REPI site. The Java bytecode downloaded over the network is first verified by the bytecode verifier and then allows to run on local host if it is found to be originated from the reliable source. Client/ server technology further increases the security of the database where description of all the requirements for the project to be developed is stored. The three-tier architecture increases the flexibility in modification. The application logic can be kept on the client side or moved to the server acting as a bridge between Java applets and Oracle database server. Oracle database provides the communicating link between users and developers. Users enter their requirements to the database at their wish and developers view these requirements from the database at their convenience. With the tremendous growth of the internet and Java along with the computer industry it is anticipated that the REPI when implemented fully will prove itself invaluable.

CHAPTER 5

CONCLUSION AND FUTURE WORK

This thesis is the first attempt to apply two hot technologies for the purpose of implementing Requirements Elicitation through Internet (REPI): Java, which is the hot Internet technology and client/server technology. Chapter 1 of this thesis described about Java as it is emerging as the current and future technology of the Internet and why Java is more suitable as compared to the combination of HTML forms and CGI scripts for the Requirement Elicitation process. Chapter 2 described the another hot technology namely, client/server architecture which is on the way to outdate the mainframe technology and how this client/server technology can be used to implement Requirement Elicitation process. Chapter 3 described Requirement Elicitation process at a part of Requirement Engineering and Software Engineering Institute's Requirements Elicitation framework and the process model. Chapter 4 of this thesis described the implementation of Requirements Elicitation Process through Internet (REPI) web site using the prototype developed in [DEEPAK 98] and the technologies described in chapters 1 and 2.

5.1 Advantages of the REPI

The process of eliciting requirements for a product to be built requires different people from different areas of expertise to work together as a group. During Requirements Elicitation part of a project communication between the various members of the project is the primary issue because group members have to work together as a unit. To arrive at a common understanding is the main purpose of the Requirement Elicitation. This general consensus can be easily obtained through information sharing. Information sharing

127

requires communication between the two parties namely, stakeholders and developers. Easing communications between these two parties is the first step to make the process of eliciting requirement easier, which should lead to better requirements specification and eventually a better product. The communications between the stakeholders and the developers include both a channel and a technique. A channel is the medium for communication, while a technique is the method for communication. REPI provides assistance for these aspects of communication. The medium of communication is the Internet, while the technique used is based on the Software Engineering Institute's framework for the Requirements Elicitation process [DEEPAK 98].

Using the Internet as the platform, "facilitates the distribution of the application and its data to geographically-separated users on diverse computing platforms" [GIRGENSOHN 96]. One of the major benefits of REPI is that it allows people and organizations separated in space and/or time to exchange information and come to general agreement on the needs of the end-users. The REPI provides a distributed asynchronous environment for eliciting requirements; such an environment provides several advantages as well as some limitations. Project members using the REPI web site "meet" or communicate with one another at different times, from different places. The advantages of such meetings is that "group members do not have to be physically in the same place to meet, nor must they communicate with one another at the same time" [OCKER 95]. These two characteristics of distributed asynchronous communication extend the definition of a meeting; this expanded definition of a meeting loosens the constraints in an organization and thus increases the means by which groups can accomplish their work [OCKER 95].

"Organizational and social issues have great influence on the effectiveness of communication activities" and therefore on the overall success or failure of a given project [AL-RAWAS 96]. If the communication channel is expensive between project members, limitations are placed on the number and type of communication resources between these two groups of people. The number and type of people selected as representatives for each side are likely to be limited. Sometimes surrogates or intermediaries are used as a representative to communicate with the development side people or the client side people instead of the actual clients or developers communicating with the other party; such forms of communications are labeled as indirect links. [KEIL 95] reports that direct links are better than indirect links because intermediaries might filter or distort messages between the two groups and they might not have a complete understanding of customer needs. [KEIL 95] also reports that up to a certain point the more links between customers and developers, the better it is for the development process. Another possible limitation of the expensive communication channel is that it might be restricted to one way communication [AL-RAWAS 96]. The development side people might produce documents based on their understanding and send these voluminous documents for the client side people. They might not take the time to properly validate the requirements, even if they understand the notations used in the specification document. All these limitations of the expensive communication channel reduce the accuracy of the information obtained during the Requirements Elicitation process, which may lead to poor design and consequently poor product development.

The REPI is helpful to tackle the problems associated with the communication issues. Using REPI is inexpensive compared to other forms of communication channels

such as meetings or passing documents. Written documents are also non-interactive communication channels. REPI provides faster turnaround time as REPI provides soft copies of the documents when compared with hard copies of documents that need to be sent from one location to another. This translates into an easier form of communication because the delay is greatly reduced between the responses. Theoretically there is no limit on the number of people that can be accommodated in the Requirement Elicitation process for the product to be built from the communication point of view. Also, considering communication as the major issue to be tacked with the Requirement Elicitation, REPI places no limitation of the place on the team members as far as the team members are able to get connected to the Internet. With the tremendous growth and popularity of the Internet the problem of accessing the REPI documents through the Internet can practically be considered as eliminated. Thus, limitations imposed by the communication factors are greatly reduced by REPI implementation.

Due to enormous growth of the Internet, many people are quire familiar with the World Wide Web and its clients, such as Netscape Navigator or Microsoft Internet Explorer. As the user interface for REPI is nothing more than a series of web pages, using REPI will be as easy as browsing through the web.

What you see is what you got. Another advantage of REPI is that it imposes a structured elicitation process model derived from the SEI's Requirements Elicitation process model for Requirement Elicitation. The structure such visual presentation using the text fields, text areas, buttons, choices boxes, drop down list etc. Many times, visual presentation improves the perception in the minds of a user. REPI web site provides convenience to the people involved to communicate with each other better than before.

5.2 Limitations of REPI

Like all other system, REPI project also is not flawless. It has some limitations too. The distributed asynchronous environment nature of the REPI web site creates some disadvantages. Many advantages of face-to-face meetings are lost in these distributed group meetings: "points of reference for indexing communication by time, place, and talk sequence are all missing" [OCKER 95]. Client side pulls or server side push can be used to display the current picture of a project member's face or perhaps the current display as seen by this project member. This should provide information such as what this member is doing right now, a sense of presence that is available in a face-to-face meeting. The distributed nature of the meeting also provides greater freedom in "attending" meetings. Some members of the project might contribute their input much later than others, such that communication in these "meetings" might seem disjointed. Some level of discipline and social control needs to be created to require project members to regularly login and contribute via the REPI web site [DEEPAK 98].

The disadvantages of using the web is that not all the current generation of web browsers support Java. Another disadvantage is that, despite the claims for cross platform portability of the web pages created using Java, the user interfaces look different depending on the platforms they are running. The use of data controls and widgets are the most visible source of inconsistent behavior. The following sub sections describe the problems associated with the implementation of REPI prototype.

5.2.1 Support for Java 1.1.2

At present, REPI is implemented using the Borland's JBuilder. JBuilder is a visual tool for Java. It uses JDK version 1.1.2. Since Java is in its childhood and constantly improving support for Java applets of the latest versions of Java Development Kit (JDK) is not available even through the popular browsers such as Netscape's Navigator and Microsoft's Internet Explorer. Java applets developed for REPI implementation when run through the appletviewer supplied with JDK, poses no problem. The Java applet runs well on all the three platforms: Sun Sparc workstation, Microsoft Windows 95 and Windows NT 4.0. But when they are run within the widely known browsers such as Netscape's Navigator and Microsoft's Internet Explorer 4.0, they came up different problems. Communicator 4.03 version when downloads and runs applets written using JDK 1.1.2 or 1.1.3, gives the message like "Security violations. Method verification error," while Internet Explorer does not understand what is happening. Also, the code generating style of JBuilder creates some problem to these browsers.

5.2.2 Deployment of Java Code

There are some problems associated with the deployment of the Java code. These problems can prevent the Java applets or programs from running if they are not taken care. The sections 5.2.2.1 and 5.2.2.2 deal with these problems the suggest one of the possible remedies that have been used for the implementation of REPI.

5.2.2.1 Portability

Even though Java is claimed to portable across the multiple platforms, this is not true all the times. This affects the presentation of REPI web pages created using Java. The user interfaces look different depending on the factors such as client software and its platform, screen and color resolutions, and monitor size. The use of data controls and widgets are the most visible source of inconsistent behavior. Java applets use Abstract Windowing Toolkit (AWT) in order to create GUI components such as text areas, text fields, buttons, choice boxes, drop down lists etc. Java Virtual Machine that is running these applets containing Abstract Windowing Toolkit components will always use the native data controls and widgets to display the Java form controls. Theses native data controls and widgets heavily depend upon the underlying platforms. So naturally none of these Java forms can be designed to provide the exact same "look and feel" across browsers and platforms.

5.2.2.2 Event Handling Mechanism

Event handling mechanism of JDK 1.1.2 or higher version is also a problematic in some cases when used with "package" statement. The style in which JBuilder generates the code to handle the events for the GUI components adds fuel to fire. JBuilder generates a separate class in the source file for each of the GUI component for which an event is to be handled. These two together make browsers and appletviewer confuse when running the applets. For example, this thesis attempted to implement REPI prototype by creating three packages: User, Developer and Login.

This problem created due to package statement of Java is tackled as described below. The directory called "JBuilder" contains all these three packages. The class "DBClient" does not contain any package statement as it is the generic class that is to be used by all the classes defined in these three packages. The "JBuilder" directory will have all the html files containing the applets for the Requirement Elicitation tasks, the "User" directory, the "Developer" directory, and the "Login" directory. The html files will have "." as the codebase value. The code value will be the relative path for the class the html file will be containing. For example, the file U_ER_1.html will have the statement like this

code= "User.U_ER_1.class"

Similarly, the file D_ER_1.html will have

code= "Developer.D_ER_1.class"

5.2.3 Client/Server Architecture

The Application server written in Java runs well on the UNIX platform. Applets can communicate with Oracle database via this server. However due to some mystic problem, Java applets or clients that runs quite well on one system hang up on another system. The client/server programs in which Java applets work clients communicates easily with database server when the Application server is running on the UNIX system known as "logic," while the same applets stuck up when run on another UNIX system called "homer". The Application server does its job satisfactorily; but the applets do not read from sockets in which the Application server has transport its reply.

5.3 Future Work

The REPI web site developed for this thesis is the first attempt to implement REPI prototype. This thesis does not implement REPI prototype fully. Many tasks of the Requirement Elicitation process have not been implemented. All the help menus are yet to be provided. The forms are created using Java applets. These applets are yet to make run through browser. Due to this the web page that can be used to navigate through different phases of Requirement Elicitation process has not been developed. However, the html pages, for the navigation purpose, developed in the REPI prototype demo version can be easily used. Also, the back-end database development is under progress, at present, all the applets are not communicating with the database. The client/server architecture has been implemented. For the REPI web site, a database has to be designed to store the requirements and other information generated as the project members use the web site during Requirements Elicitation. Once the database is developed only few changes are required to be made. Consider the following scenario. Database is developed. You want to store the information collected from Task 1 Perform Risk Assessment of Users' Evaluation and Rationalization phase when "SUBMIT" button is clicked in a relation called "test". Follow the description:

- Import DBClient class into the source file U_ER_1.java by writing "import DBClient;" statement

- Instantiate the object of DBClient class by writing following statement in the class defined for button "SUBMIT" event in the source file:

  DBClient store = new DBClient(getDocumentBase().getHost(), 6001);

- Form the query and invoke the "ProcessCommand" of the DBClient object as shown below:

```
store = new DBClient(getDocumentBase().getHost(), 6001);
String query = "insert into test values (" + "'" + reqList + "'" + "," + "'" + reqTitle +
"'" + "," + "'" + category + "'" + "," + "'" + reqDesc + "'" + "," + "'" + whyReq + "'"
+ ")";
store.ProcessCommand(query);
```

Software Engineering Institute's Requirements Elicitation framework is designed to be flexible. The framework recommends use of various combinations of methods and techniques based on the characteristics of the project. The process model itself can be followed in a different manner based on the project needs and the current understanding of the project goals and requirements. The product or system to be built and its history also affects the choice of techniques during the Requirements Elicitation process. The current REPI web site imposes one type process model with one set of tasks implemented in a particular way. The REPI web site's support for Requirements Elicitation can be thought of as one instance of the SEI's process model. SEI's process model can be implemented along different lines using different methods and ways through the process model. Although the path through the process model is left up to the project members, the flexibility provided by the REPI web site doesn't match that of the Requirements Elicitation framework. This aspect of the REPI web site could be improved upon by providing alternative tasks, methods or techniques at each phase of the process model [DEEPAK 98].

APPENDIX A

JAVA EXAMPLES

```
import java.awt.*;
import java.applet.*;

interface Shapes
  {
   abstract double getArea();
   abstract double getPerimeter();
  } /* Shapes */

class Coordinates
  {
   int x,y;

   public Coordinates (int x, int y)
     {
      this.x = x;
      this.y = y;
     } /* Coordinates */
  } /* Coordinates */

class Square extends Coordinates implements Shapes
  {
   public int width, height;

   public double getArea()
     { return (width * height); } /* getArea */
   public double getPerimeter()
     { return (2 * width + 2 * height); } /* getPerimeter */

   public Square (int x, int y, int width, int height)
     {
      super (x,y);
      this.width = width;
      this.height = height;
     } /* Square */
  } /* Square */

class Circle extends Coordinates implements Shapes
  {
   public int width, height;
   public double radius;
```

```
public double getArea()
  { return (radius * radius * Math.PI); } /* getArea */
public double getPerimeter()
  { return (2 * Math.PI * radius); } /* getPerimeter */

public Circle (int x, int y, int width, int height)
  {
  super (x,y);
  this.width = width;
  this.height = height;
  radius = (double) width / 2.0;
  } /* Circle */
} /* Circle */

public class InheritanceApplet extends Applet
 { Square box = new Square (5, 15, 25, 25);
   Circle Oval = new Circle (5, 50, 25, 25);

   public void paint (Graphics g)
     { g.drawRect (Box.x, Box.y, Box.width, Box.height);
       g.drawString ("Area: " + Box.getArea(), 50, 35);
       g.drawString ("Area: " + Box.getPerimeter(), 50, 40);
       g.drawOval (Oval.x, Oval.y, Oval.width, Oval.height);
       g.drawString ("Area: " + Oval.getArea(), 50, 70);
       g.drawString ("Area: " +Oval.getPerimeter(), 50, 75);
     } /* paint */
 } /* InheritanceApplet */
```

Figure A.1: Java Inheritance Example. Source: *Java Now!* Jamsa, Kris. Jamsa Press. 1996

```
Public class plum{
   Public static void main (String args[]){
      Producer p = new Producer();
      p.start();

      Consumer c = new Consumer(p);
      c.start();
   }
}

Class Producer extends Thread{
   private String [] buffer = new String[8];
   private int pi = 0; // Produce index
   private int gi = 0; // Consumer index

   public void run(){
      for(;;) produce();
   }

   private final long start = System.currentTimeMillis();
   private final String banana(){
      return " " + (int)(System.currentTimeMillis() -
start);
   }

   synchronized void produce(){

      while((pi-gi+1) > buffer.length()){
         try{wait(); }
         catch (Exception e){}
      }

      buffer[pi&0x7] = banana();
      System.out.println("Produced[" + (pi&7) + "] " +
buffer[pi&7];
      pi++;
      notifyAll();
   }


   synchronized String consume(){

      while(pi == gi){
         try {wait(); }
         catch (Exception e){}
```

```
    }
    notifyAll();

    return buffer [gi++&0x7];
  }

}
class Consumer extends Thread{
  Produce whoIamTalkingTo;

  Consumer (Producer who) { whoIamTalkingTo = who;}

  public void run(){
    java.util.Random r = new java.util.Random();

    for(;;){
        String result  =  whoIamTalkingTo.consume();
        System.out.println ("consumed: " + result);

        int randomtime = r.nextInt() % 250;
        try{  sleep(randomtime);} catch (Exception e){}

    }
  }
}
```

Figure A.2: Java Thread Example Source: *Just Java* Peter van der Linden. 2nd Edition, Sun Soft Press, Sun MicroSystems Inc. 1997

```java
import java.applet.Applet;
import java.awt.*;
import java.lang.String;
import java.net.URL;
import java.net.MalformedURLException;
import java.io.IOException;
import java.media.*;
/**
This is a Java Applet that demonstrates how to create a
simple media Player with a media event listener.  It will
play the media clip right away and continuously loop.
*/

public class TypicalPlayerApplet extends Applet implements
ControllerListener {
// media Player
   Player Player = null;
   // component in which video is playing
   Component visualComponent  = null;
   // controls gain, position, start, stop
   Component controlComponent = null;
   // displays progress during download
   Component progressBar       = null;

// Read the applet file parameter and create the media
Player.

   public void init() {
     setLayout(new BorderLayout());

     // input file name from html param
     String mediaFile = null;

     // URL for our media file
     URL url = null;

     // URL for doc containing applet
     URL codeBase = getDocumentBase();

     // Get the media filename info.
     // The applet tag should contain the path to the
     // source media file, relative to the html page.


     if ((mediaFile = getParameter("FILE")) == null)
         Fatal("Invalid media file parameter");
          try {
```

```
        // Create an url from the file name and the url to the
        // document containing this applet.
              if ((url = new URL(codeBase, mediaFile)) == null)
                 Fatal("Can't build URL for " + mediaFile);
              // Create an instance of a Player for this media
              if ((Player = Manager.createPlayer(url)) == null)
                 Fatal("Could not create Player for "+url);
           // Add ourselves as a listener for Player's events
                 Player.addControllerListener(this);
           } catch (MalformedURLException e) {
              Fatal("Invalid media file URL!");
           } catch(IOException e) {
              Fatal("IO exception creating Player for "+url);
                 }
/* This applet assumes that its start() calls
Player.start().This causes the Player to become Realized.
Once Realized, the Applet will get the visual and control
panel components and add them to the Applet.  These
components are not added during init() because they are
long operations that would make us appear unresponsive to
the user.
*/
}
 /*
Start media file playback.  This function is called the
first time       that the Applet runs and every time the
user re-enters the page.
*/
    public void start() {
    // Call start() to Prefetch and start the Player.
      if (Player != null)
        Player.start();
      }

// Stop playback and release resources before leaving the
page.

    public void stop() {
      if (Player != null){
      Player.stop();
      Player.deallocate();
      }
    }
 /*
```

This controllerUpdate function must be defined in order to
implement a ControllerListener interface. This function
will be called whenever there is a media event.
*/

```
   public synchronized void
   controllerUpdate(ControllerEvent event) {
// If we're getting messages from a dead Player, just leave
  if (Player == null)
       return;
   // When the Player is Realized, get the visual
   // and control components and add them to the Applet
  if (event instanceof RealizeCompleteEvent) {
     if ((visualComponent = Player.getVisualComponent())
        != null)
           add("Center", visualComponent);
      if((controlComponent =
Player.getControlPanelComponent())!= null)
          add("South",controlComponent);
   // force the applet to draw the components
     validate();
   }
   else if (event instanceof CachingControlEvent) {
   // Put a progress bar up when downloading starts,
   // take it down when downloading ends.
      CachingControlEvent  e = (CachingControlEvent) event;
      CachingControl       cc = e.getCachingControl();
      long cc_progress      = e.getContentProgress();
      long cc_length        = cc.getContentLength();
   // Add the bar if not already there ...
      if (progressBar == null)
        if ((progressBar =
          cc.getProgressBarComponent()) != null) {
            add("North", progressBar);
            validate();
        }
   // Remove bar when finished downloading
      if (progressBar != null)
        if (cc_progress == cc_length) {
          remove (progressBar);
          progressBar = null;
          validate();
        }
      }


      else if (event instanceof EndOfMediaEvent) {
// We've reached the end of the media; rewind and start
```

```
        Player.setMediaTime(0);
        Player.start();
    }
    else if (event instanceof ControllerErrorEvent) {
    // Tell TypicalPlayerApplet.start() to call it a day
        Player = null;
        Fatal (((ControllerErrorEvent)event).getMessage());
    }
 }
void Fatal (String s) {
// Applications will make various choices about what
// to do here.  We print a message and then exit
  System.err.println("FATAL ERROR: " + s);
    throw new Error(s);   // Invoke the uncaught exception
} }
```

Figure A.3: Java Multimedia Example, Part 1 of 2 Source: Javasoft, Sun Microsystems Inc.

```
<HTML>
<TITLE> Welcome to Media Player </TITLE>
<BODY>
<applet code=TypicalPlayerApplet width=320 height=300>
<param name=file value="Astrnmy.avi">
</applet>
</BODY>
</HTML>
```

Figure A.4: Java Multimedia Example, Part 2 of 2 Source: Javasoft. Sun Microsystems Inc.

```java
import java.net.URL;
import java.sql.*;
import java.math.*;

class Test
{
  public static void main (String [] arg)
  {
  try
    {
    DriverManager.registerDriver(new
oracle.jdbc.OracleDriver ());
      String url = "jdbc:oracle:oci7:@host";
      Connection con =
DriverManager.getConnection(url,"userid","password");
      //Connection con =
DriverManager.getConnection(url,"","");
      Statement stmt = con.createStatement();
      ResultSet  rs = stmt.executeQuery("select * from
Worker");
      System.out.println("Got Result");

      while(rs.next())
      {
        String name = rs.getString(1);
        String age = rs.getString(2);
        String lodging = rs.getString(3);
        String w_distance = rs.getString(4);

        System.out.print(name + " " + age + " " + lodging
+ " " + w_distance);
        System.out.println();

      }
     stmt.close();
      con.close();
    }
    catch(Exception e)
    { e.printStackTrace(); }

  }
}
```

Figure A.5: Java Database Connectivity Example

APPENDIX B

CLIENT / SERVER CODE FOR REPI IMPLEMENTATION

```java
import java.awt.List;
import java.awt.Frame;
import java.net.*;
import java.io.*;
import java.util.*;
import java.sql.*;

public class ApplicationServer extends Thread {
    public final static int DEFAULT_PORT = 6001;
    protected int port;
    protected ServerSocket server_port;
    protected ThreadGroup CurrentConnections;
    protected List connection_list;
    protected Vector connections;
    protected ConnectionWatcher watcher;
    public Frame f;


    // Exit with an error message, when an exception occurs.
    public static void fail(Exception e, String msg) {
        System.err.println(msg + ": " +  e);
        System.exit(1);
    }

    // Create a ServerSocket to listen for connections on;
start the thread.
    public ApplicationServer(int port) {
        // Create our server thread with a name.
        super("Server");
        if (port == 0) port = DEFAULT_PORT;
        this.port = port;
        try { server_port = new ServerSocket(port); }
        catch (IOException e) {fail(e, "Exception creating
server socket");}
        // Create a threadgroup for our connections
        CurrentConnections = new ThreadGroup("Server
Connections");

        // Create a window to display our connections in
        f = new Frame("Server Status");
        connection_list = new List();
        f.add("Center", connection_list);
```

```
        // modified by me 1/14/98
          // f.resize(400, 200);
          f.setSize(400,200);
        f.show();

        // Initialize a vector to store our connections in
        connections = new Vector();
        // Create a ConnectionWatcher thread to wait for
other threads to die.
        // It starts itself automatically.
        watcher = new ConnectionWatcher(this);
      // I changed this from "writer" to "watcher"
        // Start the server listening for connections
        this.start();
    }


/* The body of the server thread.  Loop forever, listening
for  and  accepting  connections  from  clients.   For  each
connection,   create   a   Connection   object   to   handle
communication  through  the  new  Socket.   When  we  create  a  new
connection,  add  it  to  the  Vector  of  connections,  and
display  it  in  the  List.   Note  that  we  use  synchronized  to
lock  the  Vector  of  connections.   The  ConnectionWatcher
class  does  the  same,  so  the  watcher  won't  be  removing  dead
connections  while  we're  adding  fresh  ones.
*/
    public void run() {
        try {
            while(true) {
            Socket client_socket = server_port.accept();
            ServerConnection c = new
ServerConnection(client_socket, CurrentConnections, 3,
watcher);
        // prevent simultaneous access.
        synchronized (connections) {
          connections.addElement(c);
          connection_list.addItem(c.getInfo());
            }
            }
        }
        catch (IOException e) {fail(e, "Exception while
listening for connections");}
      f.dispose();
      System.exit(0);
    }
```

```java
    // Start the server up, listening on an optionally
specified port
    public static void main(String[] args) {
        int port = 0;
        if (args.length == 1) {
        try {port = Integer.parseInt(args[0]);}
        catch (NumberFormatException e) {port = 0;}
        }
        new ApplicationServer(port);
    }
}


/* This class is the thread that handles all communication
with a client. It also notifies the ConnectionWatcher when
the connection is dropped.
*/
class ServerConnection extends Thread {
    static int numberOfConnections = 0;
    protected Socket client;
    protected ConnectionWatcher watcher;
    protected BufferedReader in;
    protected PrintWriter out;
    Connection con;

    // Initialize the streams and start the thread
    public ServerConnection(Socket client_socket,
ThreadGroup CurrentConnections,
            int priority, ConnectionWatcher watcher) {
        // Give the thread a group, a name, and a priority.
        super(CurrentConnections, "Connection number" +
numberOfConnections++);
        this.setPriority(priority);
        // Save our other arguments away
        client = client_socket;
        this.watcher = watcher;

        // Create the streams
        try {
            in =  new BufferedReader( new InputStreamReader
( client.getInputStream() ) );
            out = new
PrintWriter(client.getOutputStream());
        }
        catch (IOException e) {
        try {client.close();} catch (IOException e2) {
```

```
            System.err.println("Exception while getting
socket streams: " + e);
            return;}
        }
        // And start the thread up
        this.start();
    }


    // Provide the service.
    // Read a line, reverse it, send it back.
    public void run() {
        String inline;

        try {
        // Loop forever, or until the connection is broken!
            while(true) {
                // read in a line
                inline = in.readLine();
            inline.trim();
            //System.out.println("FROMCLIENT "+inline);
                if (inline == null) break;

            switch(inline.toCharArray()[0]){

            case 'S' :
            case 's' :
                        //inline = in.readLine();
                        //System.out.println("I read : " +
inline);

ConnectToDatasource("jdbc:oracle:oci7:@host", "userid");
//System.out.println("I am in S option : ");
                out.print(RunQuery(inline));
                out.println("DONE");
                out.flush();
                break;

        case 'I' :
        case 'i' :

                    //inline = in.readline();

ConnectToDatasource("jdbc:oracle:oci7:@host", "userid");
                    /* InsertRecord(inline); // to be defined
                    out.print("Record is inserted");*/
```

```
                out.print(InsertRecord(inline));
                out.println("DONE");
                out.flush();
                break;

        default:
                out.println("Unknown Command");
                out.println("DONE");

      } //switch
            } // while


      }
      catch (IOException e) {}
```

/* When we're done, for whatever reason, be sure to close
the socket, and to notify the ConnectionWatcher object.
Note that we have to use synchronized first to lock the
watcher object before we can call notify() for it.
*/

```
      finally {
      try {client.close();}
      catch (IOException e2) {
          synchronized (watcher) {watcher.notify();}
      }


      }
    }
```

/* This method returns the string representation of the
Connection. This is the string that will appear in the GUI
List.
*/

```
    public String getInfo() {
      String inline="";

      //      try {  inline = in.readLine(); }
      //      catch (IOException e)
{System.out.println("Caught an Exception:" +e);}
      return (inline+" connected from: " +
client.getInetAddress().getHostName());
    }

  // DB specific stuff follows!
private void ConnectToDatasource(String url, String Name) {
try {
```

```
        DriverManager.registerDriver ( new
oracle.jdbc.OracleDriver());
        url = "jdbc:oracle:oci7:@host";// applied on homer
        con = DriverManager.getConnection(url, "userid",
"password");
    }
    catch( Exception e ) {
       e.printStackTrace();
System.out.println(e.getMessage());
    }
}

private String RunQuery(String QueryLine) {
   String Output="";
   int columns;
   int pos;
   try {

     //System.out.println("I got the Query : " + QueryLine);
     Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery(QueryLine);
     columns=(rs.getMetaData()).getColumnCount();

       while(rs.next()) {

      for( pos=1; pos<=columns; pos++) {

        Output+=rs.getObject(pos)+" ";
      }
      Output+="\n";

       }
       stmt.close();
       con.close();
     }
     catch( Exception e ) {
       e.printStackTrace();
       Output=e.getMessage();
     }
     return Output;
   }
   // end DB specific stuff


private String InsertRecord (String input){
String output = "";
    try {
```

```
    Statement stmt = con.createStatement();
    stmt.executeUpdate(input);

      stmt.close();
      con.close();
    }
    catch( Exception e ) {
      e.printStackTrace();
      output=e.getMessage();
      }
  return output;
} // InsertRecord


} // end class Connection


/* This class waits to be notified that a thread is dying
(exiting)and then cleans up the list of threads and the
graphical list.
*/
class ConnectionWatcher extends Thread {
    protected ApplicationServer server;
    protected ConnectionWatcher(ApplicationServer s) {
        super(s.CurrentConnections, "ConnectionWatcher");
        server = s;
        this.start();
    }


/* This is the method that waits for notification of
exiting threads and cleans up the lists.   It is a
synchronized method, so it acquires a lock on the `this'
object before running.   This is necessary so that it can
call wait() on this.   Even if the Connection objects never
call notify(), this method wakes up every five seconds and
checks all the connections, just in case. Note also that
all access to the Vector of connections and to the GUI List
component are within a synchronized block as well. This
prevents the Server class from adding a new connection
while we're removing an old one.
*/
    public synchronized void run() {
        while(true) {
        try {this.wait(10000);}
            catch (InterruptedException e){
                System.out.println("Caught an Interrupted
Exception");
                }
```

```
                  // prevent simultaneous access
            synchronized(server.connections) {
                  // loop through the connections
                  for(int i = 0; i <
server.connections.size(); i++) {
                        ServerConnection c;
                        c =
(ServerConnection)server.connections.elementAt(i);
                        // if the connection thread isn't alive
anymore,
                        // remove it from the Vector and List.
                        if (!c.isAlive()) {

server.connections.removeElementAt(i);
                        server.connection_list.delItem(i);
                              i--;
                        }
                  }
            }

         }
        }
     }
```

Figure B.1: Concurrent Server Used to Provide Back End Support for REPI, Part 1 of 2

```java
import java.io.*;
import java.net.*;
import java.applet.*;

public class DBClient {
public   Socket socket;
public PrintWriter out;
public String Name;
public Reader reader;

public DBClient (String ServerName, int ServerPort) {
   try {   socket = new Socket(ServerName, ServerPort);
            reader = new Reader(this);
         out = new PrintWriter(socket.getOutputStream());
      }
        catch (IOException e) {System.err.println(e);}
      }

public String ProcessCommand(String InLine) {
//System.out.println("FROM DBCLIENT:"+InLine);
out.println(InLine);
out.flush();


synchronized(reader){reader.notify();reader.notifyOn=false;
}
  while(true) {
     if (reader.notifyOn) {break;}
  }

  //System.out.println("Reader RESULT:
"+reader.getResult());// removed by chetan

  return(reader.getResult());
}
}

class  Reader extends Thread {
protected DBClient client;
public String Result="original";
public  boolean notifyOn=true;

public Reader(DBClient c) {
  super("DBclient Reader");
  this.client = c;
  this.start();
}
```

```java
public synchronized void run() {
  String line="";

  BufferedReader in=null;

  try {
     in = new BufferedReader ( new InputStreamReader
(client.socket.getInputStream() ) );
    while(true) {
       try {if (notifyOn) {this.wait(); notifyOn=false;
Result="";}}
       catch (InterruptedException e){
     System.out.println("Caught an Interrupted Exception");
       }
       // prevent simultaneous access
       line = in.readLine();
  //System.out.println("I read from Server : " + line);
       if (line.equalsIgnoreCase("DONE")) {
         notifyOn=true;
        //break;
       } else
     {
     if (line == null) {
       System.out.println("Server closed connection.");
       break;
     } // if NOT null
     else {Result+=line+"\n";}
//   System.out.println("Read from server: "+Result);
     } // if NOT done..
    } //while loop
  }
  catch (IOException e) {System.out.println("Reader: " +
e);}
  finally {
    try {if (in != null) in.close();}
    catch (IOException e) {
      System.exit(0);
    }
  }
}

public String getResult() {
  return (Result);
  }
}
```
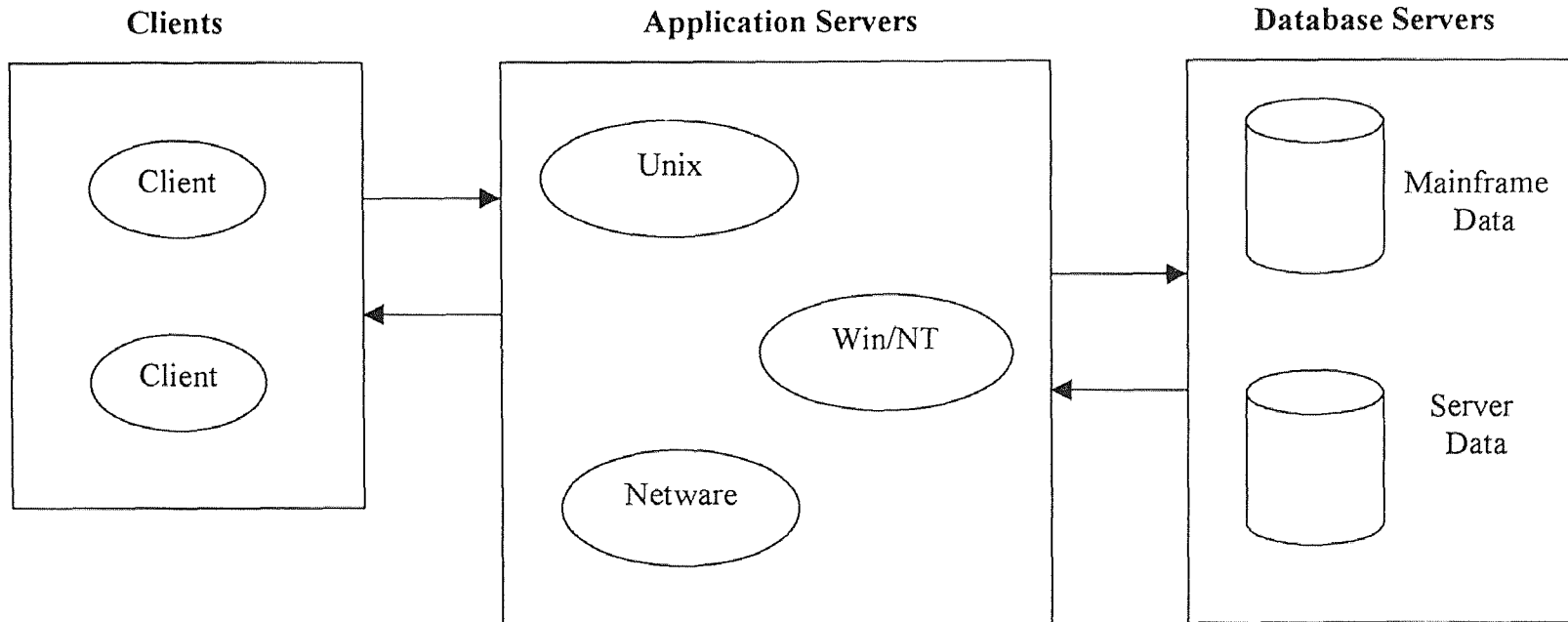
Figure B.2: Part of a Client Used to Provide Back End Support for REPI, Part 2 of 2

B.3 FUNCTION BASED CLIENT/SERVER ARCHITECTURE

Figure **B.3**: Function Based Three Tiered Client/Server Model Source: *Client/Server Architecture*. Alex Berson, 2nd Edition McGraw-Hill publication, 1996

APPENDIX C

USERS' TASKS FOR REPI: SOURCE CODE AND FRONT END

```java
//Title: Requirement Elicitation Process through Internet
//Version:
//Copyright:  Copyright (c) 1997
//Author:      Your Name
//Company:     Your Company
//Description:Your description
package User;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*; // used for bevelpanel

public class U_FF_1 extends Applet {
  boolean isStandalone = false;
  String firstName;
  String lastName;
  String workPhone;
  String email;
  String category;
  Checkbox tempBox;

  BorderLayout borderLayout1 = new BorderLayout();
  BevelPanel bevelPanel1 = new BevelPanel();
  TextField txtFirstName = new TextField();
  TextField txtLastName = new TextField();
  TextField txtWorkPhone = new TextField();
  TextField txtEmail = new TextField();
  Checkbox chbEndUser = new Checkbox();
  Checkbox chbCustomer = new Checkbox();
  Button btnSubmit = new Button();
  Button btnClear = new Button();
  Label label1 = new Label();
  Label lblFirst = new Label();
  Label lblLast = new Label();
  Label lblWorkPhone = new Label();
  Label lblEmail = new Label();
  XYLayout xYLayout1 = new XYLayout();
  //StatusBar statusBar1 = new StatusBar();
  Checkbox chbMgmt = new Checkbox();
  CheckboxGroup chgBox = new CheckboxGroup();
```

```java
   //Get a parameter value
   public String getParameter(String key, String def) {
     return isStandalone ? System.getProperty(key, def) :
       (getParameter(key) != null ? getParameter(key) :
def);
   }

   //Construct the applet
   public U_FF_1() {
   }
//Initialize the applet
   public void init() {
     try { firstName = this.getParameter("FirstName", ""); }
catch (Exception e) { e.printStackTrace(); }
     try { lastName = this.getParameter("LastName", ""); }
catch (Exception e) { e.printStackTrace(); }
     try { workPhone = this.getParameter("WorkPhone", ""); }
catch (Exception e) { e.printStackTrace(); }
     try { email = this.getParameter("E-mail", ""); } catch
(Exception e) { e.printStackTrace(); }
     try { jbInit(); } catch (Exception e) {
e.printStackTrace(); }
   }

   //Component initialization
   public void jbInit() throws Exception{
     this.setSize(new Dimension(582, 360));
     txtFirstName.setFont(new Font("Dialog", 0, 18));
     txtLastName.setFont(new Font("Dialog", 0, 18));
     txtWorkPhone.setFont(new Font("Dialog", 0, 18));
     txtEmail.setFont(new Font("Dialog", 0, 18));
     bevelPanel1.setLayout(xYLayout1);
     chbEndUser.setForeground(new Color(0, 0, 108));
     chbEndUser.setFont(new Font("Dialog", 1, 12));
     chbEndUser.setLabel("End user");
     chbEndUser.setCheckboxGroup(chgBox);
     chbCustomer.setForeground(new Color(0, 0, 108));
     chbCustomer.setFont(new Font("Dialog", 1, 12));
     chbCustomer.setLabel("Customer");
     chbCustomer.setCheckboxGroup(chgBox);
     btnSubmit.setBackground(Color.darkGray);
     btnSubmit.setForeground(Color.cyan);
     btnSubmit.setFont(new Font("Dialog", 1, 18));
     btnSubmit.setLabel("SUBMIT");
     btnSubmit.addActionListener(new
U_FF_1_btnSubmit_actionAdapter(this));
```

```
        btnClear.setForeground(Color.cyan);
        btnClear.setBackground(Color.darkGray);
        btnClear.setFont(new Font("SansSerif", 1, 18));
        btnClear.setLabel("CLEAR");
        btnClear.addActionListener(new
U_FF_1_btnClear_actionAdapter(this));
        label1.setForeground(new Color(173,0,173));
        label1.setFont(new Font("Dialog",1,18));
        label1.setAlignment(1);
        label1.setText("TASK 1: IDENTIFY DOMAIN EXPERTS");
        lblFirst.setForeground(new Color(155, 0, 108));
        lblFirst.setFont(new Font("Dialog", 1, 18));
        lblFirst.setAlignment(2);
        lblFirst.setText("First Name");
        lblLast.setForeground(new Color(155, 0, 108));
        lblLast.setFont(new Font("Dialog", 1, 18));
        lblLast.setAlignment(2);
        lblLast.setText("Last Name");
        lblWorkPhone.setForeground(new Color(155, 0, 108));
        lblWorkPhone.setFont(new Font("Dialog", 1, 18));
        lblWorkPhone.setAlignment(2);
        lblWorkPhone.setText("Work Phone");
        lblEmail.setForeground(new Color(147, 0, 108));
        lblEmail.setFont(new Font("Dialog", 1, 18));
        lblEmail.setAlignment(2);
        lblEmail.setText("E-Mail");
        //statusBar1.setAlignment(Label.CENTER);
        chbMgmt.setForeground(new Color(0, 0, 108));
        chbMgmt.setFont(new Font("Dialog", 1, 12));
        chbMgmt.setLabel("Management");
        chbMgmt.setCheckboxGroup(chgBox);
        this.setLayout(borderLayout1);
        this.add(bevelPanel1, BorderLayout.CENTER);
        bevelPanel1.add(label1, new XYConstraints(90,2,-1,30));
        bevelPanel1.add(txtFirstName, new XYConstraints(147,
51, 148, -1));
        bevelPanel1.add(txtLastName, new XYConstraints(425, 51,
148, -1));
        bevelPanel1.add(txtWorkPhone, new XYConstraints(147,
121, 148, -1));
        bevelPanel1.add(txtEmail, new XYConstraints(425, 121,
148, -1));
        bevelPanel1.add(chbEndUser, new XYConstraints(128, 196,
-1, -1));
        bevelPanel1.add(chbCustomer, new XYConstraints(252,
196, -1, -1));
```

```
    bevelPanel1.add(btnSubmit, new XYConstraints(150, 262,
97, 35));
    bevelPanel1.add(btnClear, new XYConstraints(359, 261,
94, 35));
    bevelPanel1.add(lblFirst, new XYConstraints(10, 52, -1,
32));
    bevelPanel1.add(lblLast, new XYConstraints(304, 49, -1,
32));
    bevelPanel1.add(lblWorkPhone, new XYConstraints(10,
124, -1, 32));
    bevelPanel1.add(lblEmail, new XYConstraints(340, 124, -
1, 32));
    //bevelPanel1.add(statusBar1, new XYConstraints(11,
316, 562, 39));
    bevelPanel1.add(chbMgmt, new XYConstraints(369, 195, -
1, -1));
    chgBox.setSelectedCheckbox(chbEndUser);
  }

  //Start the applet
  public void start() {
    firstName = "";
    lastName = "";
    workPhone = "";
    email = "";
    txtFirstName.setText("");
    txtLastName.setText("");
    txtWorkPhone.setText("");
    txtEmail.setText("");
    chbEndUser.setState(true);
    chbCustomer.setState(false);
    chbMgmt.setState(false);
  }

  //Stop the applet
  public void stop() {
  }

  //Destroy the applet
  public void destroy() {
  }

  //Get Applet information
  public String getAppletInfo() {
    return "Applet Information";
  }
```

```java
    //Get parameter info
    public String[][] getParameterInfo() {
      String pinfo[][] =
      {
        {"FirstName", "String", ""},
        {"LastName", "String", ""},
        {"WorkPhone", "String", ""},
        {"E-mail", "String", ""},
      };
      return pinfo;
    }


    //Main method
    static public void main(String[] args) {
      U_FF_1 applet = new U_FF_1();
      applet.isStandalone = true;
      DecoratedFrame frame = new DecoratedFrame();
      frame.setTitle("Applet Frame");
      frame.add(applet, BorderLayout.CENTER);
      applet.init();
      applet.start();
      frame.pack();
      Dimension d =
Toolkit.getDefaultToolkit().getScreenSize();
      frame.setLocation((d.width - frame.getSize().width) /
2, (d.height - frame.getSize().height) / 2);
      frame.setVisible(true);
    }

  void btnSubmit_actionPerformed(ActionEvent e) {
   /* String firstName = "";
     String lastname = "";
     String workPhone = "";
     String email = "";
     String category = "";
     Checkbox tempBox;*/

     firstName = txtFirstName.getText();
     lastName  = txtLastName.getText();
     workPhone = txtWorkPhone.getText();
     email     = txtEmail.getText();
     tempBox = chgBox.getSelectedCheckbox();

       if(tempBox == chbEndUser)
         category = "End User";
       else
       if(tempBox == chbCustomer)
```

```
         category = "Customer";
       else
         category = "Management";

    //statusBar1.setText("I got:"+firstName+"   " + lastName
+ "   " + workPhone + "   " + email + "   " + category);
  }
  void btnClear_actionPerformed(ActionEvent e) {
     firstName = "";
     lastName = "";
     workPhone = "";
     email = "";
     txtFirstName.setText("");
     txtLastName.setText("");
     txtWorkPhone.setText("");
     txtEmail.setText("");
     chbEndUser.setState(true);
   }
}

class U_FF_1_btnSubmit_actionAdapter implements
java.awt.event.ActionListener {
  U_FF_1 adaptee;

  U_FF_1_btnSubmit_actionAdapter(U_FF_1 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnSubmit_actionPerformed(e);
  }
}

class U_FF_1_btnClear_actionAdapter implements
java.awt.event.ActionListener {
  U_FF_1 adaptee;

  U_FF_1_btnClear_actionAdapter(U_FF_1 adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.btnClear_actionPerformed(e);
  }
}
```

Figure C.1: Source Code for "Task 1: Identify Domain Experts," Users' Fact Finding Phase for REPI, Part 1of 3

```
<HTML>
<TITLE>
</TITLE>
<BODY>
<APPLET
  CODEBASE = "."
  CODE     = "User.U_FF_1.class"
  NAME     = "TestApplet"
  WIDTH    = 400
  HEIGHT   = 300
  HSPACE   = 0
  VSPACE   = 0
  ALIGN    = Middle>
<PARAM NAME = "FirstName" VALUE = "">
<PARAM NAME = "LastName" VALUE = "">
<PARAM NAME = "WorkPhone" VALUE = "">
<PARAM NAME = "E-mail" VALUE = "">
</APPLET>
</BODY>
</HTML>
```

Figure C.2: Source Code for "Task 1: Identify Domain Experts," Users' Fact Finding Phase for REPI, Part 2 of 3

Figure C.3: Front End for "Task 1: Identify Domain Experts," Users' Fact Finding Phase for REPI, Part 3 of 3

```
//Title:    Requirement Elicitation Process through Internet
//Version:
//Copyright: Copyright (c) 1997
//Author:    Chetan patel
//Company:   NJIT
//Description:Masters' Student (CIS)
package User;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;

public class U_FF_2 extends Applet {
  XYLayout xYLayout1 = new XYLayout();
  boolean isStandalone = false;
  String descProblem;
  Label label1 = new Label();
  TextArea txaProblem = new TextArea();
  Button btnSubmit = new Button();
  Button btnClear = new Button();
  Button btnBrowse = new Button();

  //Get a parameter value
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) :
def);
  }

  //Construct the applet
  public U_FF_2() {
  }

  //Initialize the applet
  public void init() {
    try { descProblem = this.getParameter("DescProblem",
""); } catch (Exception e) { e.printStackTrace(); }
    try { jbInit(); } catch (Exception e) {
e.printStackTrace(); }
  }

  //Component initialization
  public void jbInit() throws Exception{
```

```
    xYLayout1.setWidth(512);
    xYLayout1.setHeight(326);
    label1.setForeground(new Color(170, 0, 170));
    label1.setFont(new Font("Dialog", 1, 18));
    label1.setAlignment(1);
    label1.setText("TASK 2: DESCRIBE THE PROBLEM");
    btnSubmit.setBackground(Color.darkGray);
    btnSubmit.setForeground(Color.cyan);
    btnSubmit.setFont(new Font("Dialog", 1, 18));
    btnSubmit.setLabel("SUBMIT");
    btnSubmit.addActionListener(new
U_FF_2_btnSubmit_actionAdapter(this));
    btnClear.setBackground(Color.darkGray);
    btnClear.setForeground(Color.cyan);
    btnClear.setFont(new Font("Dialog", 1, 18));
    btnClear.setLabel("CLEAR");
    btnClear.addActionListener(new
U_FF_2_btnClear_actionAdapter(this));
    btnBrowse.setForeground(new Color(155, 15, 0));
    btnBrowse.setFont(new Font("Dialog", 3, 18));
    btnBrowse.setLabel("IMPORT");
    btnBrowse.addActionListener(new
U_FF_2_btnBrowse_actionAdapter(this));
    this.setLayout(xYLayout1);
    this.add(label1, new XYConstraints(60, 9, 321, 28));
    this.add(txaProblem, new XYConstraints(22, 45, 385,
187));
    this.add(btnSubmit, new XYConstraints(25, 267, -1,
34));
    this.add(btnClear, new XYConstraints(325, 264, 85,
36));
    this.add(btnBrowse, new XYConstraints(417, 105, -1,
34));
  }

  //Start the applet
  public void start() {
  }

  //Stop the applet
  public void stop() {
  }

  //Destroy the applet
  public void destroy() {
  }
```

```java
//Get Applet information
public String getAppletInfo()
{
   return "Applet Information";
}

//Get parameter info
public String[][] getParameterInfo()
{
   String pinfo[][] =
   {
     //{"DescProblem", "String", ""},
     {"descProblem", "String", ""},
   };
   return pinfo;
}

void btnSubmit_actionPerformed(ActionEvent e) {
   btnSubmit.setEnabled(false);
   descProblem = txaProblem.getText();
}

void btnClear_actionPerformed(ActionEvent e) {
 descProblem = "";
 txaProblem.setText("");
 btnSubmit.setEnabled(true);
}

void btnBrowse_actionPerformed(ActionEvent e) {
   openFile();
}

void openFile(){
 FileDialog fileDlg = new FileDialog ( new Frame());
 fileDlg.setMode(FileDialog.LOAD);
 fileDlg.setVisible(true);

   if(fileDlg.getFile() != null){
     try{
       File file = new File (fileDlg.getDirectory() +
fileDlg.getFile());
       int size = (int)file.length();
       int chars_read = 0;
       char [] data = new char[size];
       FileReader in = new FileReader(file);

         while(in.ready())
```

```
            chars_read += in.read(data, chars_read, size -
chars_read);

        in.close();
        txaProblem.setText(new String(data, 0,
chars_read));
      }
      catch(IOException e){
        txaProblem.setText("ERROR OPENING " +
fileDlg.getDirectory() + fileDlg.getFile());
      }
    }// if
  } // openFile

}

class U_FF_2_btnSubmit_actionAdapter implements
java.awt.event.ActionListener {
  U_FF_2 adaptee;

  U_FF_2_btnSubmit_actionAdapter(U_FF_2 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnSubmit_actionPerformed(e);
  }
}

class U_FF_2_btnClear_actionAdapter implements
java.awt.event.ActionListener {
  U_FF_2 adaptee;

  U_FF_2_btnClear_actionAdapter(U_FF_2 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnClear_actionPerformed(e);
  }
}

class U_FF_2_btnBrowse_actionAdapter implements
java.awt.event.ActionListener {

  U_FF_2 adaptee;
```

```
U_FF_2_btnBrowse_actionAdapter(U_FF_2 adaptee) {
  this.adaptee = adaptee;
}

public void actionPerformed(ActionEvent e) {
  adaptee.btnBrowse_actionPerformed(e);
}
}
```
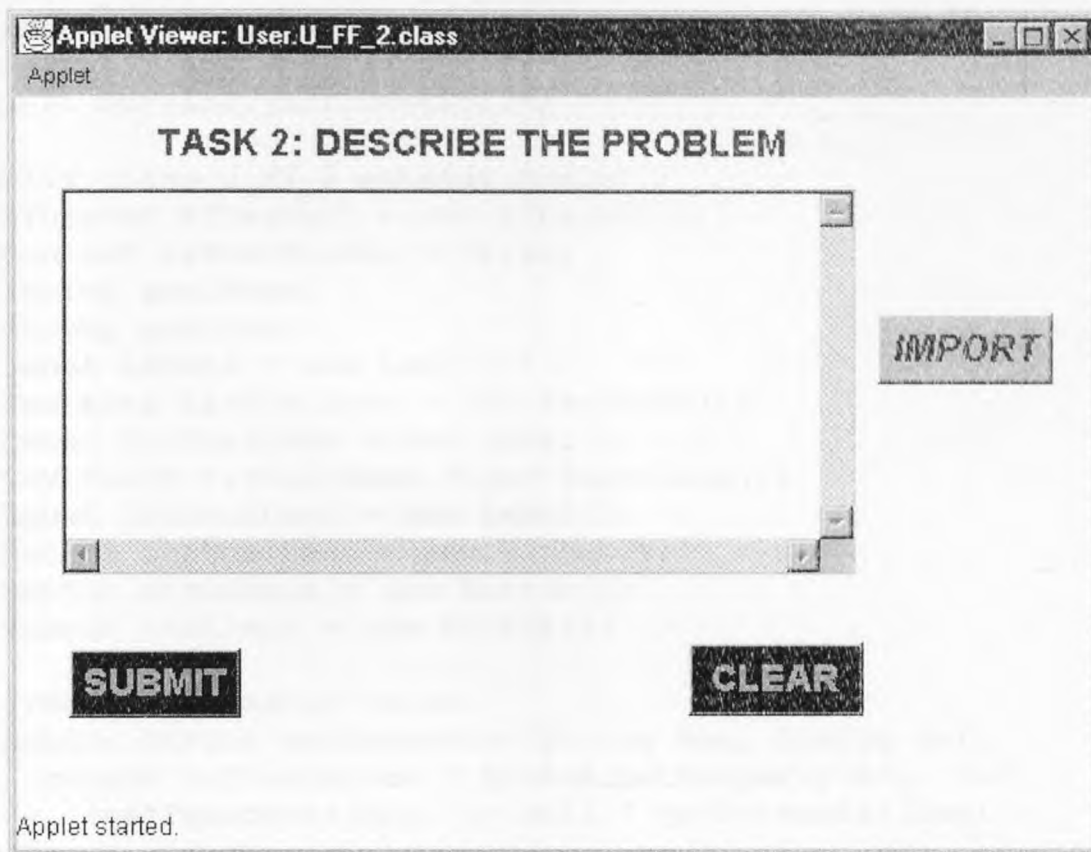
Figure C.4: Source Code for "Task 2: Describe the Problem." Users' Fact Finding Phase for REPI, Part 1 of 3

```
<HTML>
<TITLE>
HTML Test Page
</TITLE>
<BODY>
<APPLET
  CODEBASE = "."
  CODE     = "User.U_FF_2.class"
  NAME     = "TestApplet"
  WIDTH    = 520
  HEIGHT   = 400
  HSPACE   = 0
  VSPACE   = 0
  ALIGN    = Middle>
<PARAM NAME = "DescProblem" VALUE = "">
</APPLET>
</BODY>
</HTML>
```

Figure C.5: Source Code for "Task 2: Describe the Problem," Users' Fact Finding Phase for REPI, Part 2 of 3

Figure C.6: Front End for "Task 2: Describe the Problem," Users' Fact Finding Phase for REPI, Part 3 of 3

```
//Title:   Requirement Elicitation Process through Internet
//Version:
//Copyright:  Copyright (c) 1997
//Author:      Chetan Patel
//Company:     NJIT
//Description:Masters' Student (CIS)
package User;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;

public class U_FF_3 extends Applet {
  XYLayout xYLayout1 = new XYLayout();
  boolean isStandalone = false;
  String goalName;
  String goalDesc;
  Label label1 = new Label();
  TextArea txaGoalDesc = new TextArea();
  Label lblGoalname = new Label();
  TextField txfGoalName = new TextField();
  Label lblGoalDesc = new Label();
  Button btnGoalDesc = new Button();
  Button btnSubmit = new Button();
  Button btnClear = new Button();

  //Get a parameter value
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) :
def);
  }

  //Construct the applet
  public U_FF_3() {
  }

  //Initialize the applet
  public void init() {
    try { goalName = this.getParameter("GoalName", ""); }
catch (Exception e) { e.printStackTrace(); }
    try { goalDesc = this.getParameter("GoalDesc", ""); }
catch (Exception e) { e.printStackTrace(); }
```

```
      try { jbInit(); } catch (Exception e) {
e.printStackTrace(); }
  }


  //Component initialization
  public void jbInit() throws Exception{
    this.setSize(new Dimension(532, 400));
    xYLayout1.setWidth(532);
    xYLayout1.setHeight(400);
    label1.setForeground(new Color(170, 0, 170));
    label1.setFont(new Font("Dialog", 1, 18));
    label1.setAlignment(1);
    label1.setText("TASK 3:  DEFINE THE GOAL");
    lblGoalname.setForeground(new Color(155, 0, 108));
    lblGoalname.setFont(new Font("Dialog", 1, 12));
    lblGoalname.setAlignment(2);
    lblGoalname.setText("PROJECT GOAL NAME");
    txfGoalName.setColumns(60);
    lblGoalDesc.setForeground(new Color(155, 0, 108));
    lblGoalDesc.setFont(new Font("Dialog", 1, 12));
    lblGoalDesc.setText("PROJECT GOAL DESCRIPTION");
    btnSubmit.setBackground(Color.darkGray);
    btnSubmit.setForeground(Color.cyan);
    btnSubmit.setFont(new Font("Dialog", 1, 18));
    btnSubmit.setLabel("SUBMIT");
    btnSubmit.addActionListener(new
U_FF_3_btnSubmit_actionAdapter(this));
    btnClear.setForeground(Color.cyan);
    btnClear.setBackground(Color.darkGray);
    btnClear.setFont(new Font("Dialog", 1, 18));
    btnClear.setLabel("CLEAR");
    btnClear.addActionListener(new
U_FF_3_btnClear_actionAdapter(this));
    btnGoalDesc.setForeground(new Color(155, 15, 0));
    btnGoalDesc.setFont(new Font("Dialog", 3, 18));
    btnGoalDesc.setLabel("IMPORT");
    btnGoalDesc.addActionListener(new
U_FF_3_btnGoalDesc_actionAdapter(this));
    this.setLayout(xYLayout1);
    this.add(label1, new XYConstraints(102, 2, 289, 26));
    this.add(txaGoalDesc, new XYConstraints(22, 147, 413,
159));
    this.add(lblGoalname, new XYConstraints(8, 55, -1,
16));
    this.add(txfGoalName, new XYConstraints(164, 52, 368, -
1));
```

```java
    this.add(lblGoalDesc, new XYConstraints(133, 121, -1,
24));
    this.add(btnGoalDesc, new XYConstraints(440, 202, -1,
35));
    this.add(btnSubmit, new XYConstraints(22, 330, 85,
31));
    this.add(btnClear, new XYConstraints(359, 334, 79,
30));
  }

  //Start the applet
  public void start() { }

  //Stop the applet
  public void stop() {}

  //Destroy the applet
  public void destroy() { }

  //Get Applet information
  public String getAppletInfo() {
    return "Applet Information";
  }

  //Get parameter info
  public String[][] getParameterInfo() {
    String pinfo[][] =
    {
      {"GoalName", "String", ""},
      {"GoalDesc", "String", ""},
    };
    return pinfo;
  }

  void btnGoalDesc_actionPerformed(ActionEvent e) {
    openFile();
  }
  void openFile(){
   FileDialog fileDlg = new FileDialog ( new Frame());
   fileDlg.setMode(FileDialog.LOAD);
   fileDlg.setVisible(true);

    if(fileDlg.getFile() != null){
      try{
        File file = new File (fileDlg.getDirectory() +
fileDlg.getFile());
        int size = (int)file.length();
```

```
        int chars_read = 0;
        char [] data = new char[size];
        FileReader in = new FileReader(file);

        while(in.ready())
            chars_read += in.read(data, chars_read, size -
chars_read);

        in.close();
        txaGoalDesc.setText(new String(data, 0,
chars_read));
        }
      catch(IOException e){
        txaGoalDesc.setText("ERROR OPENING " +
fileDlg.getDirectory() + fileDlg.getFile());
        }
    }// if
  } // openFile

  void btnSubmit_actionPerformed(ActionEvent e) {
    goalName = txfGoalName.getText();
    goalDesc = txaGoalDesc.getText();
    btnSubmit.setEnabled(true);
  }

  void btnClear_actionPerformed(ActionEvent e) {
    goalName = "";
    goalDesc = "";
    txfGoalName.setText("");
    txaGoalDesc.setText("");
    btnSubmit.setEnabled(true);
  }

}

class U_FF_3_btnGoalDesc_actionAdapter implements
java.awt.event.ActionListener {
  U_FF_3 adaptee;

  U_FF_3_btnGoalDesc_actionAdapter(U_FF_3 adaptee) {
    this.adaptee = adaptee;
  }
  public void actionPerformed(ActionEvent e) {
    adaptee.btnGoalDesc_actionPerformed(e);
  }
}
```

```
class U_FF_3_btnSubmit_actionAdapter implements
java.awt.event.ActionListener{
  U_FF_3 adaptee;

  U_FF_3_btnSubmit_actionAdapter(U_FF_3 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnSubmit_actionPerformed(e);
  }
}

class U_FF_3_btnClear_actionAdapter implements
java.awt.event.ActionListener {
  U_FF_3 adaptee;

  U_FF_3_btnClear_actionAdapter(U_FF_3 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnClear_actionPerformed(e);
  }
}
```

Figure C.7: Source Code for "Task 3: Define the Goal," Users' Fact Finding Phase for REPI, Part 1 of 3

```
<HTML>
<TITLE>
HTML Test Page
</TITLE>
<BODY>
<APPLET
  CODEBASE = "."
  CODE     = "User.U_FF_3.class"
  NAME     = "TestApplet"
  WIDTH    = 550
  HEIGHT   = 400
  HSPACE   = 0
  VSPACE   = 0
  ALIGN    = Middle>
<PARAM NAME = "GoalName" VALUE = "">
<PARAM NAME = "GoalDesc" VALUE = "">
```
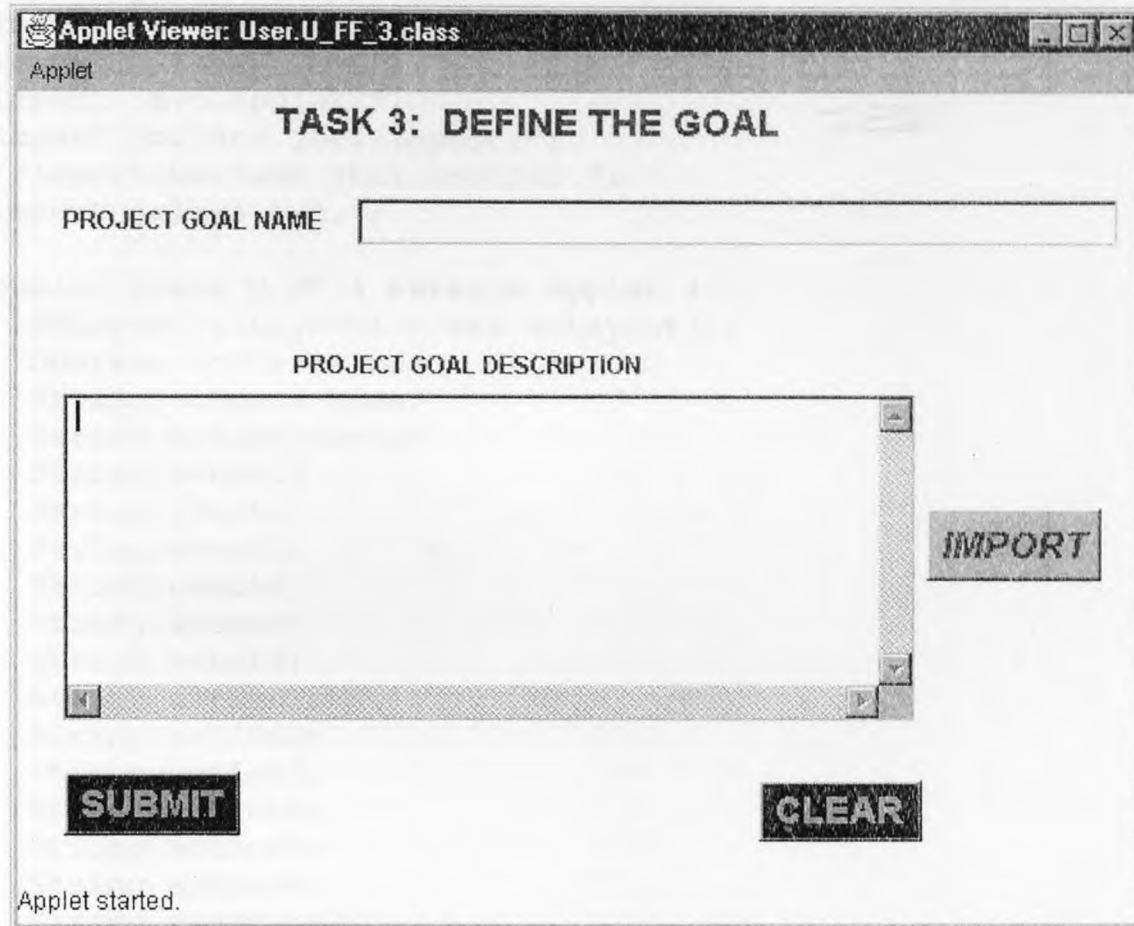
```
</APPLET>
</BODY>
</HTML>
```

Figure C.8: Source Code for "Task 3: Define the Goal," Users' Fact Finding Phase for REPI, Part 2 of 3

Figure C.9: Front End for "Task 3: Define the Goal," Users' Fact Finding Phase for REPI, Part 3 of 3

```java
//Title:      Requirement Elicitation Process through
Internet
//Version:
//Copyright:  Copyright (c) 1997
//Author:     Chetan Patel
//Company:    NJIT
//Description:Masters' Student (CIS)
package User;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
//import borland.jbcl.control.*;
import jclass.bwt.*;

public class U_FF_4 extends Applet {
  XYLayout xYLayout1 = new XYLayout();
  boolean isStandalone = false;
  String scenarioName;
  String scenarioDesc;
  String event1;
  String event2;
  String event3;
  String event4;
  String event5;
  String event6;
  String action1;
  String action2;
  String action3;
  String action4;
  String action5;
  String action6;
  String reaction1;
  String reaction2;
  String reaction3;
  String reaction4;
  String reaction5;
  String reaction6;
  Label label1 = new Label();
  Label lblScenarionName = new Label();
  TextField txfScenarioName = new TextField();
  TextArea txaScenarioDesc = new TextArea();
  Button btnScenarioDesc = new Button();
  Label lblScenarioDesc = new Label();
  JCSeparator jCSeparator1 = new JCSeparator();
```

```
  Label lblEvent = new Label();
  Label lblAction = new Label();
  Label lblReaction = new Label();
  TextField txfEvent1 = new TextField();
  TextField txfAction1 = new TextField();
  TextField txfReaction1 = new TextField();
  JCSeparator jCSeparator2 = new JCSeparator();
  JCSeparator jCSeparator3 = new JCSeparator();
  JCSeparator jCSeparator4 = new JCSeparator();
  JCSeparator jCSeparator5 = new JCSeparator();
  JCSeparator jCSeparator6 = new JCSeparator();
  JCSeparator jCSeparator7 = new JCSeparator();
  JCSeparator jCSeparator8 = new JCSeparator();
  TextField txfEvent2 = new TextField();
  TextField txfAction2 = new TextField();
  TextField txfReaction2 = new TextField();
  TextField txfEvent3 = new TextField();
  TextField txfAction3 = new TextField();
  TextField txfReaction3 = new TextField();
  TextField txfEvent4 = new TextField();
  TextField txfAction4 = new TextField();
  TextField txfReaction4 = new TextField();
  TextField txfEvent5 = new TextField();
  TextField txfAction5 = new TextField();
  TextField txfReaction5 = new TextField();
  TextField txfEvent6 = new TextField();
  TextField txfAction6 = new TextField();
  TextField txfReaction6 = new TextField();
  JCSeparator jCSeparator9 = new
JCSeparator(JCSeparator.VERTICAL);
  JCSeparator jCSeparator10 = new
JCSeparator(JCSeparator.VERTICAL);
  JCSeparator jCSeparator11 = new
JCSeparator(JCSeparator.VERTICAL);
  JCSeparator jCSeparator12 = new
JCSeparator(JCSeparator.VERTICAL);
  Button btnSubmit = new Button();
  Button btnClear = new Button();

  //Get a parameter value
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) :
def);
  }

  //Construct the applet
```

```java
    public U_FF_4() {
    }

    //Initialize the applet
    public void init() {
        try { scenarioName = this.getParameter("ScenarioName",
""); } catch (Exception e) { e.printStackTrace(); }
        try { scenarioDesc = this.getParameter("ScenationDesc",
""); } catch (Exception e) { e.printStackTrace(); }
        try { event1 = this.getParameter("Event1", ""); } catch
(Exception e) { e.printStackTrace(); }
        try { event2 = this.getParameter("Event2", ""); } catch
(Exception e) { e.printStackTrace(); }
        try { event3 = this.getParameter("Event3", ""); } catch
(Exception e) { e.printStackTrace(); }
        try { event4 = this.getParameter("Event4", ""); } catch
(Exception e) { e.printStackTrace(); }
        try { event5 = this.getParameter("Event5", ""); } catch
(Exception e) { e.printStackTrace(); }
        try { event6 = this.getParameter("Event6", ""); } catch
(Exception e) { e.printStackTrace(); }
        try { action1 = this.getParameter("Action1", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { action2 = this.getParameter("Action2", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { action3 = this.getParameter("Action3", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { action4 = this.getParameter("Action4", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { action5 = this.getParameter("Action5", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { action6 = this.getParameter("Action6", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { reaction1 = this.getParameter("Reaction1", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { reaction2 = this.getParameter("Reaction2", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { reaction3 = this.getParameter("Reaction3", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { reaction4 = this.getParameter("Reaction4", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { reaction5 = this.getParameter("Reaction5", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { reaction6 = this.getParameter("Reaction6", ""); }
catch (Exception e) { e.printStackTrace(); }
        try { jbInit(); } catch (Exception e) {
e.printStackTrace(); }
```

```java
    }

    //Component initialization
    public void jbInit() throws Exception{
        this.setSize(new Dimension(600, 621));
        xYLayout1.setWidth(600);
        xYLayout1.setHeight(621);
        label1.setForeground(new Color(178, 0, 178));
        label1.setFont(new Font("Dialog", 1, 18));
        label1.setAlignment(1);
        label1.setText("TASK 4:   LIST MISSION SCENARIO");
        lblScenarionName.setForeground(new Color(155, 0, 108));
        lblScenarionName.setFont(new Font("Dialog", 1, 12));
        lblScenarionName.setText("SCENARIO NAME");
        txfScenarioName.setColumns(60);
        btnScenarioDesc.setForeground(new Color(155, 15, 0));
        btnScenarioDesc.setFont(new Font("Dialog", 3, 18));
        btnScenarioDesc.setLabel("IMPORT");
        btnScenarioDesc.addActionListener(new
U_FF_4_btnScenarioDesc_actionAdapter(this));
        btnSubmit.setBackground(Color.darkGray);
        btnSubmit.setForeground(Color.cyan);
        btnSubmit.setFont(new Font("Dialog", 1, 18));
        btnSubmit.setLabel("SUBMIT");
        btnSubmit.addActionListener(new
U_FF_4_btnSubmit_actionAdapter(this));
        btnClear.setFont(new Font("Dialog", 1, 18));
        btnClear.setForeground(Color.cyan);
        btnClear.setBackground(Color.darkGray);
        btnClear.setLabel("CLEAR");
        btnClear.addActionListener(new
U_FF_4_btnClear_actionAdapter(this));
        lblScenarioDesc.setForeground(new Color(155, 0, 108));
        lblScenarioDesc.setFont(new Font("Dialog", 1, 15));
        lblScenarioDesc.setText("GENERAL SCENARIO
DESCRIPTION");
        lblEvent.setForeground(new Color(155, 0, 108));
        lblEvent.setFont(new Font("Dialog", 1, 12));
        lblEvent.setAlignment(1);
        lblEvent.setText("EVENT");
        lblAction.setForeground(new Color(155, 0, 108));
        lblAction.setFont(new Font("Dialog", 1, 12));
        lblAction.setText("ACTION");
        lblReaction.setForeground(new Color(155, 0, 108));
        lblReaction.setFont(new Font("Dialog", 1, 12));
        lblReaction.setAlignment(1);
        lblReaction.setText("REACTION");
```

```
    jCSeparator2.setSize(new Dimension(580,3));

jCSeparator9.setOrientation(jclass.bwt.BWTEnum.VERTICAL);

jCSeparator10.setOrientation(jclass.bwt.BWTEnum.VERTICAL);

jCSeparator11.setOrientation(jclass.bwt.BWTEnum.VERTICAL);

jCSeparator12.setOrientation(jclass.bwt.BWTEnum.VERTICAL);
    lblAction.setAlignment(1);
    lblScenarionName.setAlignment(2);
    this.setLayout(xYLayout1);
    this.add(label1, new XYConstraints(105, 2, -1, 32));
    this.add(lblScenarionName, new XYConstraints(17, 55, -
1, 22));
    this.add(txfScenarioName, new XYConstraints(142, 54,
310, -1));
    this.add(txaScenarioDesc, new XYConstraints(35, 125,
420, 100));
    this.add(btnScenarioDesc, new XYConstraints(472, 148, -
1, 35));
    this.add(lblScenarioDesc, new XYConstraints(107, 107, -
1, 16));
    this.add(jCSeparator1, new XYConstraints(0, 244, 582,
5));
    this.add(lblEvent, new XYConstraints(55, 254, 46, 19));
    this.add(lblAction, new XYConstraints(261, 253, 53,
18));
    this.add(lblReaction, new XYConstraints(442, 255, 67,
19));
    this.add(txfEvent1, new XYConstraints(14, 293, 148, -
1));
    this.add(txfAction1, new XYConstraints(199, 294, 177, -
1));
    this.add(txfReaction1, new XYConstraints(407, 294, 158,
-1));
    this.add(jCSeparator2, new XYConstraints(0, 278, 580,
2));
    this.add(jCSeparator3, new XYConstraints(0, 325, 581,
2));
    this.add(jCSeparator4, new XYConstraints(0, 371, 583,
3));
    this.add(jCSeparator5, new XYConstraints(2, 417, 580,
2));
    this.add(jCSeparator6, new XYConstraints(-1, 452, 583,
3));
```

```
    this.add(jCSeparator7, new XYConstraints(3, 498, 580, -
1));
    this.add(jCSeparator8, new XYConstraints(1, 540, 582,
5));
    this.add(txfEvent2, new XYConstraints(15, 337, 146, -
1));
    this.add(txfAction2, new XYConstraints(197, 340, 180, -
1));
    this.add(txfReaction2, new XYConstraints(409, 339, 157,
-2));
    this.add(txfEvent3, new XYConstraints(15, 384, 147, -
1));
    this.add(txfAction3, new XYConstraints(199, 386, 179, -
1));
    this.add(txfReaction3, new XYConstraints(411, 385, 157,
-2));
    this.add(txfEvent4, new XYConstraints(16, 424, 146, -
1));
    this.add(txfAction4, new XYConstraints(200, 426, 177, -
1));
    this.add(txfReaction4, new XYConstraints(412, 427, 158,
-1));
    this.add(txfEvent5, new XYConstraints(16, 466, 146, -
1));
    this.add(txfAction5, new XYConstraints(201, 467, 176, -
1));
    this.add(txfReaction5, new XYConstraints(413, 467, 157,
-1));
    this.add(txfEvent6, new XYConstraints(17, 510, 146, -
1));
    this.add(txfAction6, new XYConstraints(201, 510, 179, -
1));
    this.add(txfReaction6, new XYConstraints(413, 509, 158,
-1));
    this.add(jCSeparator10, new XYConstraints(176, 245, 4,
299));
    this.add(jCSeparator11, new XYConstraints(392, 245, 4,
295));
    this.add(jCSeparator9, new XYConstraints(1, 246, 3,
300));
    this.add(jCSeparator12, new XYConstraints(581, 247, 2,
296));
    this.add(btnSubmit, new XYConstraints(56, 563, -1, -
1));
    this.add(btnClear, new XYConstraints(445, 560, 75,
29));
    }
```

```java
//Start the applet
public void start() { }

//Stop the applet
public void stop() { }

//Destroy the applet
public void destroy() { }

//Get Applet information
public String getAppletInfo() {
   return "Applet Information"; }

//Get parameter info
public String[][] getParameterInfo() {
   String pinfo[][] =
   {
     {"ScenarioName", "String", ""},
     {"ScenationDesc", "String", ""},
     {"Event1", "String", ""},
     {"Event2", "String", ""},
     {"Event3", "String", ""},
     {"Event4", "String", ""},
     {"Event5", "String", ""},
     {"Event6", "String", ""},
     {"Action1", "String", ""},
     {"Action2", "String", ""},
     {"Action3", "String", ""},
     {"Action4", "String", ""},
     {"Action5", "String", ""},
     {"Action6", "String", ""},
     {"Reaction1", "String", ""},
     {"Reaction2", "String", ""},
     {"Reaction3", "String", ""},
     {"Reaction4", "String", ""},
     {"Reaction5", "String", ""},
     {"Reaction6", "String", ""},
   };
   return pinfo;
}

void btnScenarioDesc_actionPerformed(ActionEvent e) {
   openFile();
}

void openFile(){
```

```
   FileDialog fileDlg = new FileDialog ( new Frame());
   fileDlg.setMode(FileDialog.LOAD);
   fileDlg.setVisible(true);

  if(fileDlg.getFile() != null){
    try{
       File file = new File (fileDlg.getDirectory() +
fileDlg.getFile());
       int size = (int)file.length();
       int chars_read = 0;
       char [] data = new char[size];
       FileReader in = new FileReader(file);

         while(in.ready())
           chars_read += in.read(data, chars_read, size-
chars_read);

       in.close();
       txaScenarioDesc.setText(new String(data, 0,
chars_read));
      }
    catch(IOException e){
       txaScenarioDesc.setText("ERROR OPENING " +
fileDlg.getDirectory() + fileDlg.getFile());
      }
    }// if
  } // openFile

  void btnSubmit_actionPerformed(ActionEvent e) {
    scenarioName = txfScenarioName.getText();
    scenarioDesc = txaScenarioDesc.getText();
    event1 = txfEvent1.getText();
    event2 = txfEvent2.getText();
    event3 = txfEvent3.getText();
    event4 = txfEvent4.getText();
    event5 = txfEvent5.getText();
    event6 = txfEvent6.getText();
    action1 = txfAction1.getText();
    action2 = txfAction2.getText();
    action3 = txfAction3.getText();
    action4 = txfAction4.getText();
    action5 = txfAction5.getText();
    action6 = txfAction6.getText();
    reaction1 = txfReaction1.getText();
    reaction2 = txfReaction2.getText();
    reaction3 = txfReaction3.getText();
    reaction4 = txfReaction4.getText();
```

```
    reaction5 = txfReaction5.getText();
    reaction6 = txfReaction6.getText();
    btnSubmit.setEnabled(false);
}

void btnClear_actionPerformed(ActionEvent e) {

  event1 = "";
  event2 = "";
  event3 = "";
  event4 = "";
  event5 = "";
  event6 = "";
  action1 = "";
  action2 = "";
  action3 = "";
  action4 = "";
  action5 = "";
  action6 = "";
  reaction1 = "";
  reaction2 = "";
  reaction3 = "";
  reaction4 = "";
  reaction5 = "";
  reaction6 = "";

  txfScenarioName.setText("");
  txaScenarioDesc.setText("");

  txfEvent1.setText("");
  txfEvent2.setText("");
  txfEvent3.setText("");
  txfEvent4.setText("");
  txfEvent5.setText("");
  txfEvent6.setText("");

  txfAction1.setText("");
  txfAction2.setText("");
  txfAction3.setText("");
  txfAction4.setText("");
  txfAction5.setText("");
  txfAction6.setText("");

  txfReaction1.setText("");
  txfReaction2.setText("");
  txfReaction3.setText("");
  txfReaction4.setText("");
```

```
      txfReaction5.setText("");
      txfReaction6.setText("");
      btnSubmit.setEnabled(true);
   }

}

class U_FF_4_btnScenarioDesc_actionAdapter implements
java.awt.event.ActionListener {
   U_FF_4 adaptee;

   U_FF_4_btnScenarioDesc_actionAdapter(U_FF_4 adaptee) {
      this.adaptee = adaptee;
   }

   public void actionPerformed(ActionEvent e) {
      adaptee.btnScenarioDesc_actionPerformed(e);
   }
}

class U_FF_4_btnSubmit_actionAdapter implements
java.awt.event.ActionListener {
   U_FF_4 adaptee;

   U_FF_4_btnSubmit_actionAdapter(U_FF_4 adaptee) {
      this.adaptee = adaptee;
   }

   public void actionPerformed(ActionEvent e) {
      adaptee.btnSubmit_actionPerformed(e);
   }
}

class U_FF_4_btnClear_actionAdapter implements
java.awt.event.ActionListener {
   U_FF_4 adaptee;

   U_FF_4_btnClear_actionAdapter(U_FF_4 adaptee) {
      this.adaptee = adaptee;
   }
   public void actionPerformed(ActionEvent e) {
      adaptee.btnClear_actionPerformed(e);
   }
}
```

Figure C.10: Source Code for "Task 4: List Mission Scenario," Users' Fact Finding Phase for REPI, Part 1 of 3

```
<HTML>
<TITLE>
HTML Test Page
</TITLE>
<BODY>
<APPLET
  CODEBASE  = "."
  CODE      = "User.U_FF_4.class"
  NAME      = "TestApplet"
  WIDTH     = 600
  HEIGHT    = 621
  HSPACE    = 0
  VSPACE    = 0
  ALIGN     = Middle>
<PARAM NAME = "ScenarioName" VALUE = "">
<PARAM NAME = "ScenationDesc" VALUE = "">
<PARAM NAME = "Event1" VALUE = "">
<PARAM NAME = "Event2" VALUE = "">
<PARAM NAME = "Event3" VALUE = "">
<PARAM NAME = "Event4" VALUE = "">
<PARAM NAME = "Event5" VALUE = "">
<PARAM NAME = "Event6" VALUE = "">
<PARAM NAME = "Action1" VALUE = "">
<PARAM NAME = "Action2" VALUE = "">
<PARAM NAME = "Action3" VALUE = "">
<PARAM NAME = "Action4" VALUE = "">
<PARAM NAME = "Action5" VALUE = "">
<PARAM NAME = "Action6" VALUE = "">
<PARAM NAME = "Reaction1" VALUE = "">
<PARAM NAME = "Reaction2" VALUE = "">
<PARAM NAME = "Reaction3" VALUE = "">
<PARAM NAME = "Reaction4" VALUE = "">
<PARAM NAME = "Reaction5" VALUE = "">
<PARAM NAME = "Reaction6" VALUE = "">
</APPLET>
</BODY>
</HTML>
```

Figure C.11: Source Code for "Task 4: List Mission Scenario," Users' Fact Finding Phase for REPI, Part 2 of 3

Figure C.12: Front End for "Task 4: List Mission Scenario," Users' Fact Finding Phase for REPI, Part 3 of 3

```
//Title:     Requirement Elicitation Process through Internet
//Version:
//Copyright:  Copyright (c) 1997
//Author:     Chetan Patel
//Company:    NJIT
//Description:CIS Student (Masters')
//package Developer;
package User;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;
import jclass.bwt.*;

public class U_GC_2 extends Applet {
   XYLayout xYLayout1 = new XYLayout();
   boolean isStandalone = false;
   String reqTitle;
   String category;
   String reqDesc;
   Label lblReqID = new Label();
   Label lblReqTitle = new Label();
   Label lblCategory = new Label();
   Label lblPick = new Label();
   Label lblOr = new Label();
   Label lblType = new Label();
   Choice chcReqId = new Choice();
   TextField txfReqTitle = new TextField();
   Choice chcCategory = new Choice();
   TextField txfCategory = new TextField();
   Label lblCompliance = new Label();
   Choice chcCompliance = new Choice();
   Label lblStatus = new Label();
   Choice chcStatus = new Choice();
   Label lblReqDesc = new Label();
   TextArea txaReqDesc = new TextArea();
   Button btnReqDesc = new Button();
   Label lblReqType = new Label();
   Label lblVerify = new Label();
   CheckboxGroup chgReqType = new CheckboxGroup();
   Checkbox chbFunctional = new Checkbox();
   Checkbox chbNonFunctional = new Checkbox();
   Checkbox chbInterface = new Checkbox();
   Checkbox chbDesign = new Checkbox();
```

```
CheckboxGroup chgVerify = new CheckboxGroup();
Checkbox chbInspection = new Checkbox();
Checkbox chbAnalysis = new Checkbox();
Checkbox chbDemo = new Checkbox();
Checkbox chbTest = new Checkbox();
Choice chcNonFunctional = new Choice();
Choice chcInterface = new Choice();
JCSeparator jCSeparator1 = new JCSeparator();
JCSeparator jCSeparator2 = new JCSeparator();
JCSeparator jCSeparator3 = new JCSeparator();
JCSeparator jCSeparator4 = new JCSeparator();
JCSeparator jCSeparator5 = new JCSeparator();
JCSeparator jCSeparator6 = new JCSeparator();
JCSeparator jCSeparator7 = new JCSeparator();
JCSeparator jCSeparator8 = new JCSeparator();
JCSeparator jCSeparator9 = new JCSeparator();
Button btnSubmit = new Button();
Button btnClear = new Button();
Label label1 = new Label();

//Get a parameter value
public String getParameter(String key, String def) {
   return isStandalone ? System.getProperty(key, def) :
     (getParameter(key) != null ? getParameter(key) :
def);
 }

//Construct the applet
public U_GC_2() {
 }

//Initialize the applet
public void init() {
   try { reqTitle = this.getParameter("ReqTitle", ""); }
catch (Exception e) { e.printStackTrace(); }
   try { category = this.getParameter("Category", ""); }
catch (Exception e) { e.printStackTrace(); }
   try { reqDesc = this.getParameter("Requirement", ""); }
catch (Exception e) { e.printStackTrace(); }
   try { jbInit(); } catch (Exception e) {
e.printStackTrace(); }
 }

//Component initialization
public void jbInit() throws Exception{
   this.setSize(new Dimension(617, 500));
   xYLayout1.setWidth(617);
```

```
xYLayout1.setHeight(500);
lblReqID.setForeground(new Color(155, 0, 108));
lblReqID.setFont(new Font("Dialog", 1, 12));
lblReqID.setAlignment(1);
lblReqID.setText("REQ. ID");
lblReqTitle.setForeground(new Color(155, 0, 108));
lblReqTitle.setFont(new Font("Dialog", 1, 12));
lblReqTitle.setAlignment(1);
lblReqTitle.setText("REQ. TITLE");
lblCategory.setForeground(new Color(155, 0, 108));
lblCategory.setFont(new Font("Dialog", 1, 12));
lblCategory.setAlignment(2);
lblCategory.setText("CATEGORY");
lblPick.setForeground(new Color(155, 0, 108));
lblPick.setFont(new Font("Dialog", 1, 12));
lblPick.setAlignment(1);
lblPick.setText("PICK ONE");
lblOr.setForeground(Color.blue);
lblOr.setFont(new Font("Dialog", 1, 12));
lblOr.setAlignment(1);
lblOr.setText("OR");
lblType.setForeground(new Color(178, 0, 108));
lblType.setFont(new Font("Dialog", 1, 12));
lblType.setAlignment(1);
lblType.setText("TYPE ONE");
chcReqId.setFont(new Font("Dialog", 1, 12));
lblCompliance.setForeground(new Color(155, 0, 108));
lblCompliance.setFont(new Font("Dialog", 1, 12));
lblCompliance.setAlignment(2);
lblCompliance.setText("COMPLIACE LEVEL");
lblStatus.setForeground(new Color(155, 0, 108));
lblStatus.setFont(new Font("Dialog", 1, 12));
lblStatus.setAlignment(2);
lblStatus.setText("CURRENT STATUS");
lblReqDesc.setForeground(new Color(155, 0, 108));
lblReqDesc.setFont(new Font("Dialog", 1, 12));
lblReqDesc.setAlignment(1);
lblReqDesc.setText("DESCRIBE THE REQUIREMENT");
chbFunctional.setForeground(new Color(0, 0, 108));
chbFunctional.setFont(new Font("Dialog", 1, 12));
chbFunctional.setLabel("Functional");
chbFunctional.setCheckboxGroup(chgReqType);
chbNonFunctional.setForeground(new Color(0, 0, 108));
chbNonFunctional.setFont(new Font("Dialog", 1, 12));
chbNonFunctional.setLabel("Non-Functional");
chbNonFunctional.setCheckboxGroup(chgReqType);
chbInterface.setForeground(new Color(0, 0, 108));
```

```
chbInterface.setFont(new Font("Dialog", 1, 12));
chbInterface.setLabel("Interface");
chbInterface.setCheckboxGroup(chgReqType);
chbDesign.setForeground(new Color(0, 0, 108));
chbDesign.setFont(new Font("Dialog", 1, 12));
chbDesign.setLabel("Design Criteria");
chbDesign.setCheckboxGroup(chgReqType);
//chgVerify.setCurrent(chbInspection);
//chgVerify.setSelectedCheckbox(chbInspection);
chbInspection.setForeground(new Color(0, 0, 108));
chbInspection.setFont(new Font("Dialog", 1, 12));
chbInspection.setLabel("Inspection");
chbInspection.setCheckboxGroup(chgVerify);
chbAnalysis.setForeground(new Color(0, 0, 108));
chbAnalysis.setFont(new Font("Dialog", 1, 12));
chbAnalysis.setLabel("Analysis");
chbAnalysis.setCheckboxGroup(chgVerify);
chbDemo.setForeground(new Color(0, 0, 108));
chbDemo.setFont(new Font("Dialog", 1, 12));
chbDemo.setLabel("Demonstration");
chbDemo.setCheckboxGroup(chgVerify);
chbTest.setForeground(new Color(0, 0, 108));
chbTest.setFont(new Font("Dialog", 1, 12));
chbTest.setLabel("Test");
chbTest.setCheckboxGroup(chgVerify);

jCSeparator3.setOrientation(jclass.bwt.BWTEnum.VERTICAL);

jCSeparator4.setOrientation(jclass.bwt.BWTEnum.VERTICAL);

jCSeparator5.setOrientation(jclass.bwt.BWTEnum.VERTICAL);

jCSeparator6.setOrientation(jclass.bwt.BWTEnum.VERTICAL);

jCSeparator7.setOrientation(jclass.bwt.BWTEnum.VERTICAL);

jCSeparator9.setOrientation(jclass.bwt.BWTEnum.VERTICAL);
    btnSubmit.setBackground(Color.darkGray);
    btnSubmit.setForeground(Color.cyan);
    btnSubmit.setFont(new Font("Dialog", 1, 18));
    btnSubmit.setLabel("SUBMIT");
    btnSubmit.addActionListener(new
U_GC_2_btnSubmit_actionAdapter(this));
    btnClear.setBackground(Color.darkGray);
    btnClear.setForeground(Color.cyan);
    btnClear.setFont(new Font("Dialog", 1, 18));
    btnClear.setLabel("CLEAR");
```

```
label1.setForeground(new Color(178, 0, 178));
label1.setFont(new Font("Dialog", 1, 18));
label1.setAlignment(1);
label1.setText("TASK 2 :  ADD REQUIREMENT");
btnClear.addActionListener(new
U_GC_2_btnClear_actionAdapter(this));
btnReqDesc.setForeground(new Color(155, 15, 0));
btnReqDesc.setFont(new Font("Dialog", 3, 18));
btnReqDesc.setLabel("BROWSE");
btnReqDesc.addActionListener(new
U_GC_2_btnReqDesc_actionAdapter(this));
lblReqType.setForeground(new Color(155, 0, 108));
lblReqType.setFont(new Font("Dialog", 1, 12));
lblReqType.setText("REQ. TYPE");
lblVerify.setForeground(new Color(155, 0, 108));
lblVerify.setFont(new Font("Dialog", 1, 12));
lblVerify.setText("VERIFIED BY");
//chgReqType.setCurrent(chbFunction);
chgReqType.setSelectedCheckbox(chbFunctional);
chgVerify.setSelectedCheckbox(chbInspection);
this.setLayout(xYLayout1);
this.add(lblReqID, new XYConstraints(10, 42, 70, 17));
this.add(lblReqTitle, new XYConstraints(130, 42, 119,
20));
this.add(lblCategory, new XYConstraints(252, 64, -1, -
1));
this.add(lblPick, new XYConstraints(362, 45, 75, 16));
this.add(lblOr, new XYConstraints(469, 50, 24, 13));
this.add(lblType, new XYConstraints(515, 41, -1, 18));
this.add(chcReqId, new XYConstraints(12, 65, 55, 17));
// by me
chcReqId.addItem("UR1");
chcReqId.addItem("UR2");
chcReqId.addItem("UR3");
chcReqId.addItem("UR4");
chcReqId.addItem("UR5");
chcReqId.addItem("UR6");
chcReqId.addItem("UR7");
chcReqId.addItem("UR8");

chcCategory.addItem("");
chcCategory.addItem("Accuracy");
chcCategory.addItem("Development Process");
chcCategory.addItem("Decision Milestones");
chcCategory.addItem("Critical Info Needs");
chcCategory.addItem("Liability Issues");
```

```java
chcCompliance.addItem("Mandatory");
chcCompliance.addItem("Goal");
chcCompliance.addItem("Objective");
chcCompliance.addItem("Optional");

chcStatus.addItem("To Be Determined");
chcStatus.addItem("To Be Reviewed");
chcStatus.addItem("Defined");
chcStatus.addItem("Approved");
chcStatus.addItem("Verified");
chcStatus.addItem("Defected");

chcNonFunctional.addItem("");
chcNonFunctional.addItem("Performance");
chcNonFunctional.addItem("Security");
chcNonFunctional.addItem("Maintainability");
chcNonFunctional.addItem("Portability");
chcNonFunctional.addItem("Extensibility");

chcInterface.addItem("");
chcInterface.addItem("User");
chcInterface.addItem("Software");
chcInterface.addItem("Communication");
chcInterface.addItem("Hardware");
chcInterface.addItem("External");

// by me
this.add(txfReqTitle, new XYConstraints(87, 65, 160, -
1));
    this.add(chcCategory, new XYConstraints(341, 64, 126, -
1));
    this.add(txfCategory, new XYConstraints(505, 64, 102, -
1));
    this.add(lblCompliance, new XYConstraints(10, 113, -1,
20));
    this.add(chcCompliance, new XYConstraints(160, 110,
146, 25));
    this.add(lblStatus, new XYConstraints(290, 113, -1,
21));
    this.add(chcStatus, new XYConstraints(433, 112, -1, -
1));
    this.add(lblReqDesc, new XYConstraints(17, 169, 480,
20));
    this.add(txaReqDesc, new XYConstraints(16, 192, 486,
76));
    this.add(btnReqDesc, new XYConstraints(513, 208, 87,
32));
```

```
    this.add(lblReqType, new XYConstraints(28, 319, 70,
20));
    this.add(lblVerify, new XYConstraints(26, 380, -1,
18));
    this.add(chbFunctional, new XYConstraints(119, 310, -1,
21));
    this.add(chbNonFunctional, new XYConstraints(216, 311,
-1, 22));
    this.add(chbInterface, new XYConstraints(359, 310, 107,
-1));
    this.add(chbDesign, new XYConstraints(490, 308, 105, -
1));
    this.add(chbInspection, new XYConstraints(121, 381, 80,
20));
    this.add(chbAnalysis, new XYConstraints(216, 381, 100,
19));
    this.add(chbDemo, new XYConstraints(360, 380, 107,
21));
    this.add(chbTest, new XYConstraints(491, 380, 102,
21));
    this.add(chcNonFunctional, new XYConstraints(214, 336,
133, 20));
    this.add(chcInterface, new XYConstraints(360, 335, 120,
22));
    this.add(jCSeparator1, new XYConstraints(18, 365, 584,
7));
    this.add(jCSeparator2, new XYConstraints(16, 297, 584,
3));
    this.add(jCSeparator3, new XYConstraints(17, 299, 3,
109));
    this.add(jCSeparator4, new XYConstraints(598, 298, 3,
110));
    this.add(jCSeparator5, new XYConstraints(482, 297, 3,
112));
    this.add(jCSeparator6, new XYConstraints(349, 298, 3,
109));
    this.add(jCSeparator7, new XYConstraints(205, 298, 3,
113));
    this.add(jCSeparator8, new XYConstraints(18, 407, 580,
3));
    this.add(btnSubmit, new XYConstraints(43, 435, -1,
28));
    this.add(btnClear, new XYConstraints(472, 438, 72,
28));
    this.add(jCSeparator9, new XYConstraints(106, 298, 2,
109));
    this.add(label1, new XYConstraints(163, 0, -1, 25));
```

```java
}

//Start the applet
public void start() { }

//Stop the applet
public void stop() { }

//Destroy the applet
public void destroy() { }

//Get Applet information
public String getAppletInfo() {
  return "Applet Information";
}

//Get parameter info
public String[][] getParameterInfo() {
  String pinfo[][] =  {
    {"ReqTitle", "String", "Requirement Title"},
    {"Category", "String", ""},
    {"Requirement", "String", "Requirement Description"},
  };
  return pinfo;
}

void btnReqDesc_actionPerformed(ActionEvent e) {
  openFile(); }

void openFile(){
  FileDialog fileDlg = new FileDialog ( new Frame());
  fileDlg.setMode(FileDialog.LOAD);
  fileDlg.setVisible(true);

  if(fileDlg.getFile() != null){
    try{
      File file = new File (fileDlg.getDirectory() +
fileDlg.getFile());
      int size = (int)file.length();
      int chars_read = 0;
      char [] data = new char[size];
      FileReader in = new FileReader(file);

      while(in.ready())
        chars_read += in.read(data, chars_read, size-
chars_read);
```

```
        in.close();
        txaReqDesc.setText(new String(data, 0,
chars_read));
      }
    catch(IOException e){
        txaReqDesc.setText("ERROR OPENING " +
fileDlg.getDirectory() + fileDlg.getFile());
      }
    }// if
  } // openFile

  void btnSubmit_actionPerformed(ActionEvent e) {
    String reqId = "";
    String reqTitle = "";
    String category = "";
    String complianceLevel = "";
    String currentStatus = "";
    String reqDesc = "";
    String reqType = "";
    String verifiedBy = "";
    Checkbox tempBox;

      reqId   = chcReqId.getSelectedItem();
      reqTitle = txfReqTitle.getText();
      category = chcCategory.getSelectedItem();
      reqDesc  = txaReqDesc.getText();
        if(category == "")
         category = txfCategory.getText();
      complianceLevel = chcCompliance.getSelectedItem();
      currentStatus = chcStatus.getSelectedItem();
      tempBox = chgReqType.getSelectedCheckbox();
        if(tempBox == chbFunctional)
          reqType = "Functional";
        else
        if(tempBox == chbNonFunctional)
          reqType = chcNonFunctional.getSelectedItem();
        else
        if(tempBox == chbInterface)
          reqType = chcInterface.getSelectedItem();
        else
        if(tempBox == chbDesign)
          reqType = "Design COnstraint";

      tempBox = chgVerify.getSelectedCheckbox();

        if(tempBox == chbInspection)
          verifiedBy = "Inspection";
```

```
          else
          if (tempBox == chbAnalysis)
            verifiedBy = "Analysis";
          else
          if (tempBox == chbDemo)
            verifiedBy = "Demonstration";
          else
           verifiedBy = "Test";
    }

    void btnClear_actionPerformed(ActionEvent e) {
      chcReqId.select(0);
      txfReqTitle.setText("");
      chcCategory.select(0);
      txfCategory.setText("");
      chcCompliance.select(0);
      chcStatus.select(0);
      txaReqDesc.setText("");
      chgReqType.setSelectedCheckbox(chbFunctional);
      chgVerify.setSelectedCheckbox(chbInspection);

    }

    void button1_actionPerformed(ActionEvent e) { }

} // class U_GC_2
class U_GC_2_btnReqDesc_actionAdapter implements
java.awt.event.ActionListener{
  U_GC_2 adaptee;

  U_GC_2_btnReqDesc_actionAdapter(U_GC_2 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnReqDesc_actionPerformed(e);
  }
}

class U_GC_2_btnSubmit_actionAdapter implements
java.awt.event.ActionListener {
  U_GC_2 adaptee;

  U_GC_2_btnSubmit_actionAdapter(U_GC_2 adaptee) {
    this.adaptee = adaptee;
  }
```

```
   public void actionPerformed(ActionEvent e) {
     adaptee.btnSubmit_actionPerformed(e);
   }
}

class U_GC_2_btnClear_actionAdapter implements
java.awt.event.ActionListener {
  U_GC_2 adaptee;

  U_GC_2_btnClear_actionAdapter(U_GC_2 adaptee) {
    this.adaptee = adaptee;
  }
public void actionPerformed(ActionEvent e) {
    adaptee.btnClear_actionPerformed(e);
  }
}
```

Figure C.13: Source Code for "Task 2: Add Requirement," Users' Gathering and Classification Phase for REPI, Part 1 of 3

```
<HTML>
<TITLE>
HTML Test Page
</TITLE>
<BODY>
<APPLET
  CODEBASE = "."
  CODE      = "User.U_GC_2.class"
  NAME      = "TestApplet"
  WIDTH     = 600
  HEIGHT    = 500
  HSPACE    = 0
  VSPACE    = 0
  ALIGN     = Middle>
<PARAM NAME = "ReqTitle" VALUE = "">
<PARAM NAME = "Category" VALUE = "">
<PARAM NAME = "Requirement" VALUE = "">
</APPLET>
</BODY>
</HTML>
```

Figure C.14: Source Code for "Task 2: Add Requirement," Users' Gathering and Classification Phase for REPI, Part 2 of 3

Figure C.15: Front End for "Task 2: Add Requirement," Users' Gathering and Classification Phase for REPI, Part 3 of 3

Figure C.16: Front End for "Task 1: Perform Abstraction," Users' Evaluation and Rationalization Phase

Figure C.17: Front End for "Task 2: Capture Rationale," Users' Evaluation and Rationalization Phase for REPI

APPENDIX D

DEVELOPERS' TASKS FOR REPI: SOURCE CODE AND FRONT END

```java
//Title:     Requirement Elicitation Process through Internet
//Version:
//Copyright: Copyright (c) 1997
//Author:        Your Name
//Company:       Your Company
//Description:MS Student (CIS)
package Developer;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*; // needed for bevelpanel

public class D_FF_1 extends Applet {
  boolean isStandalone = false;
  String firstName;
  String lastName;
  String workPhone;
  String email;
  boolean appExpert = false;
  boolean devExpert = false;
  BorderLayout borderLayout1 = new BorderLayout();
  BevelPanel bevelPanel1 = new BevelPanel();
  Label label1 = new Label();
  TextField txtFirstName = new TextField();
  TextField txtLastName = new TextField();
  TextField txtWorkPhone = new TextField();
  TextField txtEmail = new TextField();
  Checkbox chkAppExpert = new Checkbox();
  Checkbox chkDevExpert = new Checkbox();
  Button btnSubmit = new Button();
  Button btnClear = new Button();
  Label lblFirst = new Label();
  Label lblLast = new Label();
  Label lblWorkPhone = new Label();
  Label lblEmail = new Label();
  XYLayout xyLayout1 = new XYLayout();
  //StatusBar statusBar1 = new StatusBar();

  //Get a parameter value
  public String getParameter(String key, String def) {
```

```java
      return isStandalone ? System.getProperty(key, def) :
         (getParameter(key) != null ? getParameter(key) :
def);
   }

   //Construct the applet
   public D_FF_1() {
   }

   //Initialize the applet
   public void init() {
      try { firstName = this.getParameter("FirstName", ""); }
catch (Exception e) { e.printStackTrace(); }
      try { lastName = this.getParameter("LastName", ""); }
catch (Exception e) { e.printStackTrace(); }
      try { workPhone = this.getParameter("WorkPhone", ""); }
catch (Exception e) { e.printStackTrace(); }
      try { email = this.getParameter("E-mail", ""); } catch
(Exception e) { e.printStackTrace(); }
      try { jbInit(); } catch (Exception e) {
e.printStackTrace(); }
   }

   //Component initialization
   public void jbInit() throws Exception{
      this.setSize(new Dimension(582, 360));
      label1.setFont(new Font("Dialog",1,18));
      label1.setText("TASK 1: IDENTIFY DOMAIN EXPERTS");
      txtFirstName.setFont(new Font("Dialog", 0, 18);
      txtLastName.setFont(new Font("Dialog", 0, 18));
      txtWorkPhone.setFont(new Font("Dialog", 0, 18));
      txtEmail.setFont(new Font("Dialog", 0, 18));
      appExpert = false;
      devExpert = false;
      this.setLayout(xyLayout1);
      bevelPanel1.setLayout(xyLayout1);
      chkAppExpert.setFont(new Font("Dialog", 1, 12);
      chkAppExpert.setLabel("Application Expert");
      chkDevExpert.setFont(new Font("Dialog", 1, 12));
      chkDevExpert.setLabel("Developer Expert");
      btnSubmit.setFont(new Font("SansSerif", 1, 18));
      btnSubmit.setForeground(Color.cyan);
      btnSubmit.setBackground(Color.darkGray);
      btnSubmit.setLabel("SUBMIT");
      btnSubmit.addActionListener(new
D_FF_1_btnSubmit_actionAdapter(this));
      btnClear.setFont(new Font("SansSerif", 1, 18));
```

```
    btnClear.setLabel("CLEAR");
    btnClear.setForeground(Color.cyan);
    btnClear.setBackground(Color.darkGray);
    btnClear.addActionListener(new
D_FF_1_btnClear_actionAdapter(this));
    lblFirst.setFont(new Font("SansSerif", 1, 18));
    lblFirst.setAlignment(2);
    lblFirst.setText("First Name");
    lblLast.setFont(new Font("SansSerif", 1, 18));
    lblLast.setAlignment(2);
    lblLast.setText("Last Name");
    lblWorkPhone.setFont(new Font("SansSerif", 1, 18));
    lblWorkPhone.setAlignment(2);
    lblWorkPhone.setText("Work Phone");
    lblEmail.setFont(new Font("SansSerif", 1, 18));
    lblEmail.setAlignment(2);
    lblEmail.setText("E-Mail");
    //statusBar1.setAlignment(Label.CENTER);
    this.setLayout(borderLayout1);
    //bevelPanel1.setLayout(xylayout1);
    this.add(bevelPanel1, BorderLayout.CENTER);
    bevelPanel1.add(label1, new XYConstraints(90,2,-1, -
1));
    bevelPanel1.add(txtFirstName, new XYConstraints(140,
51, 141, 35));
    bevelPanel1.add(txtLastName, new XYConstraints(425, 47,
137, 37));
    bevelPanel1.add(txtWorkPhone, new XYConstraints(137,
121, 149, 34));
    bevelPanel1.add(txtEmail, new XYConstraints(424, 120,
139, 35));
    bevelPanel1.add(chkAppExpert, new XYConstraints(147,
201, -1, -1));
    bevelPanel1.add(chkDevExpert, new XYConstraints(355,
199, -1, -1));
    bevelPanel1.add(btnSubmit, new XYConstraints(150, 262,
97, 35));
    bevelPanel1.add(btnClear, new XYConstraints(359, 261,
94, 35));
    bevelPanel1.add(lblFirst, new XYConstraints(24, 52,
104, 32));
    bevelPanel1.add(lblLast, new XYConstraints(314, 49, 99,
32));
    bevelPanel1.add(lblWorkPhone, new XYConstraints(17,
124, 109, 27));
    bevelPanel1.add(lblEmail, new XYConstraints(318, 124,
98, -1));
```

```java
        //bevelPanel1.add(statusBar1, new XYConstraints(11,
316, 562, 39));
    }

    //Start the applet
    public void start() {
        firstName = "";
        lastName = "";
        workPhone = "";
        email = "";
        txtFirstName.setText("");
        txtLastName.setText("");
        txtWorkPhone.setText("");
        txtEmail.setText("");
        chkAppExpert.setState(false);
        chkDevExpert.setState(false);
    }

    //Stop the applet
    public void stop() {
    }

    //Destroy the applet
    public void destroy() {
    }

    //Get Applet information
    public String getAppletInfo() {
      return "Applet Information";
    }

    //Get parameter info
    public String[][] getParameterInfo() {
      String pinfo[][] =
      {
        {"FirstName", "String", ""},
        {"LastName", "String", ""},
        {"WorkPhone", "String", ""},
        {"E-mail", "String", ""},
      };
      return pinfo;
    }


    //Main method
    static public void main(String[] args) {
      D_FF_1 applet = new D_FF_1();
```

```
    applet.isStandalone = true;
    DecoratedFrame frame = new DecoratedFrame();
    frame.setTitle("Applet Frame");
    frame.add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    frame.pack();
    Dimension d =
Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((d.width - frame.getSize().width) /
2, (d.height - frame.getSize().height) / 2);
    frame.setVisible(true);
  }

  void btnSubmit_actionPerformed(ActionEvent e) {
    firstName = txtFirstName.getText();
    lastName  = txtLastName.getText();
    workPhone = txtWorkPhone.getText();
    email     = txtEmail.getText();
    appExpert = chkAppExpert.getState();
    devExpert = chkAppExpert.getState();
    btnSubmit.setEnabled(false);
    //statusBar1.setText("I got :" + firstName + "   " +
lastName + "   " + workPhone + "   " + email + "   " +
((appExpert==true) ? "AppExpert" :"")+(devExpert==true? "
DevExpert" : ""));

  }


  void btnClear_actionPerformed(ActionEvent e) {
    firstName = "";
    lastName = "";
    workPhone = "";
    email = "";
    txtFirstName.setText("");
    txtLastName.setText("");
    txtWorkPhone.setText("");
    txtEmail.setText("");
    chkAppExpert.setState(false);
    chkDevExpert.setState(false);
    btnSubmit.setEnabled(true);
    //statusBar1.setText("");
  }
}
```

```
class D_FF_1_btnSubmit_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_1 adaptee;

  D_FF_1_btnSubmit_actionAdapter(D_FF_1 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnSubmit_actionPerformed(e);
  }
}


class D_FF_1_btnClear_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_1 adaptee;

  D_FF_1_btnClear_actionAdapter(D_FF_1 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnClear_actionPerformed(e);
  }
}
```

Figure D.1: Source Code for "Task 1: Identify Domain Experts," Developers' Fact Finding Phase for REPI, Part 1 of 3
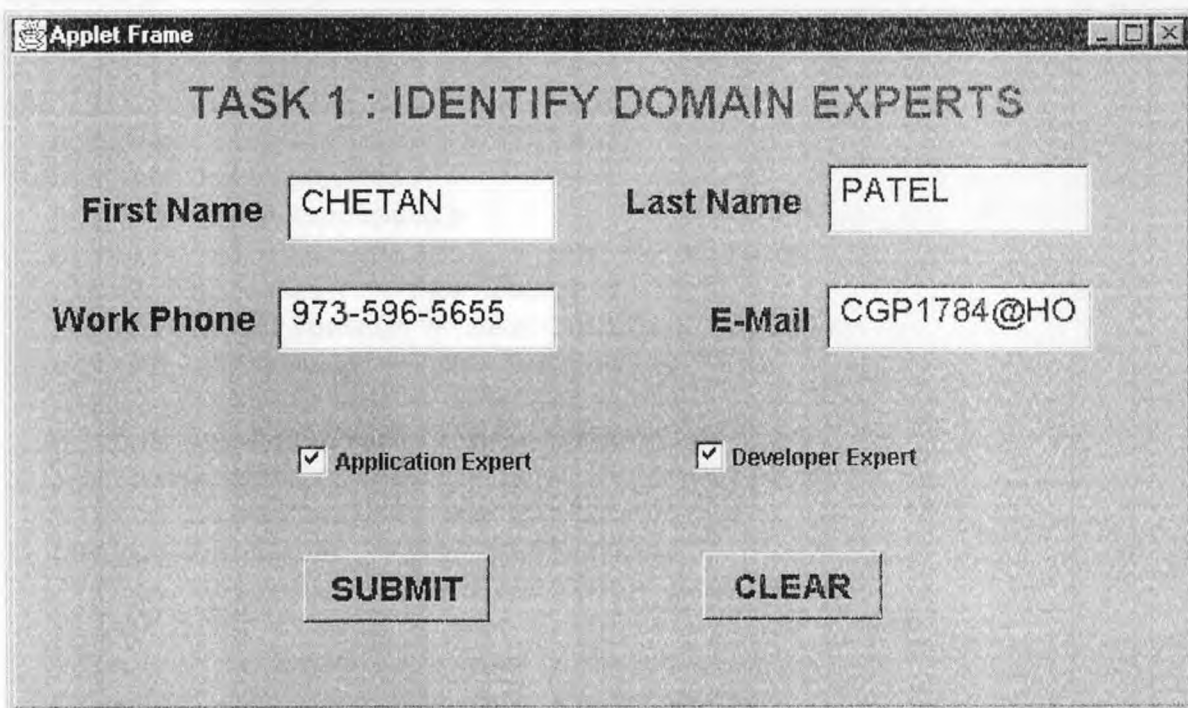
```
<HTML>
<TITLE>
Identify Domain Experts
</TITLE>
<BODY>
<APPLET
  CODEBASE = "."
  CODE     = "Developer.D_ff_1.class"
  NAME     = "TestApplet"
  WIDTH    = 400
  HEIGHT   = 300
  HSPACE   = 0
  VSPACE   = 0
  ALIGN    = Middle>
  <PARAM NAME = "FirstName" VALUE = "">
```

```
<PARAM NAME = "LastName" VALUE = "">
<PARAM NAME = "WorkPhone" VALUE = "">
<PARAM NAME = "E-mail" VALUE = "">
</APPLET>
</BODY>
</HTML>
```

Figure D.2: Source Code for "Task 1: Identify Domain Experts," Developers' Fact Finding Phase for REPI, Part 2 of 3

Figure D.3: Front End for "Task 1: Identify Domain Experts," Developers' Fact Finding Phase for REPI, Part 3 of 3

```
//Title:    Requirement Elicitation Process through Internet
//Version:
//Copyright: Copyright (c) 1997
//Author:       Your Name
//Company:      Your Company
//Description:Master's Student

package Developer;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;

public class D_FF_2 extends Applet {
  boolean isStandalone = false;
  String domainModel;
  String archtechModel;
  BevelPanel bevelPanel1 = new BevelPanel();
  Label lblDomain = new Label();
  TextArea txtDomain = new TextArea(3,80);
  Button btnDomain = new Button();
  Label lblArchtech = new Label();
  Button btnAtchtech = new Button();
  TextArea txtArchtech = new TextArea(3,80);
  Button btnSubmit = new Button();
  Button btnClear = new Button();
  //Filer filer1 = new Filer(new Frame());
  FileDialog filer1 = new FileDialog(new Frame());
  XYLayout xYLayout1 = new XYLayout();
  XYLayout xYLayout2 = new XYLayout();
  Label label1 = new Label();
  //Get a parameter value
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) :
def);
  }

  //Construct the applet
  public D_FF_2() {
  }

  //Initialize the applet
  public void init() {
```

```
    try { domainModel = this.getParameter("DomainModel",
""); } catch (Exception e) { e.printStackTrace(); }
    try { archtechModel =
this.getParameter("ArchtechModel", ""); } catch (Exception
e) { e.printStackTrace(); }
    try { jbInit(); } catch (Exception e) {
e.printStackTrace(); }
  }


  //Component initialization
  public void jbInit() throws Exception{
    this.setSize(new Dimension(550, 400));
    bevelPanel1.setLayout(xYLayout2);
    btnDomain.setForeground(new Color(155, 15, 0));
    btnDomain.setFont(new Font("Dialog", 3, 18));
    btnDomain.setSize(new Dimension(96, 30));
    btnDomain.setLabel("IMPORT");
    btnDomain.addActionListener(new
D_FF_2_btnDomain_actionAdapter(this));
    lblArchtech.setForeground(new Color(155, 0, 108));
    lblArchtech.setFont(new Font("Dialog", 1, 18));
    lblArchtech.setSize(new Dimension(340, 35));
    lblArchtech.setAlignment(1);
    lblArchtech.setText("IDENTIFY ARCHITECTURAL MODEL");
    btnAtchtech.setForeground(new Color(155, 15, 0));
    btnAtchtech.setFont(new Font("Dialog", 3, 18));
    btnAtchtech.setLabel("IMPORT");
    btnAtchtech.addActionListener(new
D_FF_2_btnAtchtech_actionAdapter(this));
    btnSubmit.setBackground(Color.darkGray);
    btnSubmit.setForeground(Color.cyan);
    btnSubmit.setFont(new Font("Dialog", 1, 18));
    btnSubmit.setLabel("SUBMIT");
    btnSubmit.addActionListener(new
D_FF_2_btnSubmit_actionAdapter(this));
    btnClear.setBackground(Color.darkGray);
    btnClear.setForeground(Color.cyan);
    btnClear.setFont(new Font("Dialog", 1, 18));
    btnClear.setLabel("CLEAR");
    filer1.setTitle("Open File");
    xYLayout1.setHeight(497);
    label1.setForeground(new Color(155, 0, 177));
    label1.setFont(new Font("Dialog", 1, 24));
    label1.setText("TASK 2: IDENTIFY DOMAIN MODELS");
    label1.setAlignment(1);
    xYLayout1.setWidth(550);
```

```
    btnClear.addActionListener(new
D_FF_2_btnClear_actionAdapter(this));
    lblDomain.setForeground(new Color(155, 0, 108));
    lblDomain.setFont(new Font("Dialog", 1, 18));
    lblDomain.setLocation(new Point(38, 20));
    lblDomain.setAlignment(1);
    lblDomain.setText("IDENTIFY DOMAIN MODEL");
    this.setLayout(xYLayout1);
    this.add(bevelPanel1, new XYConstraints(0, 0, 550,
499));
    bevelPanel1.add(lblDomain, new XYConstraints(52, 68,
367, -1));
    bevelPanel1.add(txtDomain, new XYConstraints(38, 107,
411, 124));
    bevelPanel1.add(btnDomain, new XYConstraints(456, 161,
85, 31));
    bevelPanel1.add(lblArchtech, new XYConstraints(6, 258,
503, -1));
    bevelPanel1.add(btnAtchtech, new XYConstraints(457,
335, 85, 31));
    bevelPanel1.add(txtArchtech, new XYConstraints(34, 290,
410, 128));
    bevelPanel1.add(btnSubmit, new XYConstraints(58, 433,
84, 37));
    bevelPanel1.add(btnClear, new XYConstraints(330, 434, -
1, 37));
    bevelPanel1.add(label1, new XYConstraints(37, 21, 452,
30));
  }

  //Start the applet
  public void start() {
  }

  //Stop the applet
  public void stop() {
  }

  //Destroy the applet
  public void destroy() {
  }

  //Get Applet information
  public String getAppletInfo() {
    return "Applet Information";
  }
  //Get parameter info
```

```java
  public String[][] getParameterInfo() {
    String pinfo[][] =
    {
      {"DomainModel", "String", ""},
      {"ArchtechModel", "String", ""},
    };
    return pinfo;
  }

  //Main method
  static public void main(String[] args) {
    D_FF_2 applet = new D_FF_2();
    applet.isStandalone = true;
    DecoratedFrame frame = new DecoratedFrame();
    frame.setTitle("Applet Frame");
    frame.add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    frame.pack();
    Dimension d =
Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((d.width - frame.getSize().width) /
2, (d.height - frame.getSize().height) / 2);
    frame.setVisible(true);
  }

  void btnDomain_actionPerformed(ActionEvent e) {
    openFile(txtDomain);
  }

  void btnAtchtech_actionPerformed(ActionEvent e) {
    openFile(txtArchtech);
  }

  void btnSubmit_actionPerformed(ActionEvent e) {
    domainModel = txtDomain.getText();
    archtechModel = txtArchtech.getText();
    btnSubmit.setEnabled(false);
  }

  void btnClear_actionPerformed(ActionEvent e) {
    txtDomain.setText("");
    txtArchtech.setText("");
    domainModel = "";
    archtechModel ="";
    btnSubmit.setEnabled(true);
  }
```

```
   void openFile(TextArea t){
       filer1.setMode(FileDialog.LOAD);
       filer1.setVisible(true);
       String fileName = "";

       if(filer1.getFile() != null){
         try{
             fileName = filer1.getDirectory() +
filer1.getFile();
             File file = new File(fileName);
             int size = (int)file.length();
             char[] data = new char[size];
             int chars_read = 0;
             FileReader in = new FileReader(file);

               while(in.ready()){
                 chars_read += in.read(data ,chars_read, size
- chars_read);
                 }
             in.close();
             t.setText(new String (data, 0, chars_read));

         }  //try
         catch(IOException e){
             t.setText("Error Opening " + fileName);
         }
       } //if
   }

} // class main ends

class D_FF_2_btnDomain_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_2 adaptee;

  D_FF_2_btnDomain_actionAdapter(D_FF_2 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnDomain_actionPerformed(e);
  }
}
class D_FF_2_btnAtchtech_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_2 adaptee;
```

```
  D_FF_2_btnAtchtech_actionAdapter(D_FF_2 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnAtchtech_actionPerformed(e);
  }
}

class D_FF_2_btnSubmit_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_2 adaptee;

  D_FF_2_btnSubmit_actionAdapter(D_FF_2 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnSubmit_actionPerformed(e);
  }
}

class D_FF_2_btnClear_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_2 adaptee;

  D_FF_2_btnClear_actionAdapter(D_FF_2 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnClear_actionPerformed(e);
  }
}
```

Figure D.4: Source Code for "Task 2: Identify Domain Models," Developers' Fact Finding Phase for REPI, Part 1 of 3

```
<HTML>
<TITLE>
  Test Applet
</TITLE>
<BODY>
<APPLET
  CODEBASE = "."
  CODE     = "Developer.D_FF_2.class"
  NAME     = "TestApplet"
  WIDTH    = 400
  HEIGHT   = 300
  HSPACE   = 0
  VSPACE   = 0
  ALIGN    = Middle>
<PARAM NAME = "DomainModel" VALUE = "">
<PARAM NAME = "ArchtechModel" VALUE = "">
</APPLET>
</BODY>
</HTML>
```

Figure D.5: Source Code for "Task 2: Identify Domain Models," Developers' Fact Finding Phase for REPI, Part 2 of 3

Figure D.6: Front End for "Task 2: Identify Domain Models," Developers' Fact Finding Phase for REPI, Part 3 of 3

```
//Title:    Requirement Elicitation Process through Internet
//Version:
//Copyright: Copyright (c) 1997
//Author:    Chetan Patel
//Company:   NJIT
//Description:MS Student

package Developer;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;

public class D_FF_3 extends Applet {
  boolean isStandalone = false;
  String surveyName;
  String surveyInfo;
  XYLayout xYLayout1 = new XYLayout();
  TextField txfSurveyName = new TextField();
  Button btnBrowse = new Button();
  TextArea txaSurveyInfo = new TextArea();
  Button btnSubmit = new Button();
  Button btnClear = new Button();
  Label label1 = new Label();
  Label lblSurveyName = new Label();
  Label lblSurveyInfo = new Label();

  //Get a parameter value
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) :
def);
  }

  //Construct the applet
  public D_FF_3() {
  }

  //Initialize the applet
  public void init() {
    try { surveyName = this.getParameter("SurveyName", "");
} catch (Exception e) { e.printStackTrace(); }
    try { surveyInfo = this.getParameter("SurveyInfo", "");
} catch (Exception e) { e.printStackTrace(); }
```

```
    try { jbInit(); } catch (Exception e) {
e.printStackTrace(); }
  }

  //Component initialization
  public void jbInit() throws Exception{
    xYLayout1.setHeight(430);
    label1.setForeground(new Color(173,0,173));
    label1.setFont(new Font("Dialog",1,18));
    label1.setText("TASK 3: CONDUCT TECHNOLOGICAL SURVEY");
    btnBrowse.setForeground(new Color(155, 15, 0));
//    btnBrowse.setFont(new Font("Dialog", 1, 18));
    btnBrowse.setFont(new Font("Dialog", 3, 18));
    btnBrowse.setLabel("IMPORT");
    btnBrowse.addActionListener(new
D_FF_3_btnBrowse_actionAdapter(this));
    btnSubmit.setBackground(Color.darkGray);
    btnSubmit.setForeground(Color.cyan);
    btnSubmit.setFont(new Font("Dialog", 1, 18));
    btnSubmit.setLabel("SUBMIT");
    btnSubmit.addActionListener(new
D_FF_3_btnSubmit_actionAdapter(this));
    btnClear.setBackground(Color.darkGray);
    btnClear.setForeground(Color.cyan);
    btnClear.setFont(new Font("Dialog", 1, 18));
    btnClear.setLabel("CLEAR");
    btnClear.addActionListener(new
D_FF_3_btnClear_actionAdapter(this));

    lblSurveyInfo .setFont(new Font("Dialog",1,12));
    lblSurveyInfo .setForeground(new Color(155,0,108));
    lblSurveyInfo .setAlignment(1);
    lblSurveyInfo .setText("ENTER SURVERY INFORMATION");
    lblSurveyName.setForeground(new Color(155, 0, 108));
    lblSurveyName.setFont(new Font("Dialog", 1, 12));
    lblSurveyName.setAlignment(1);
    lblSurveyName.setText("TECHNOLOGICAL SURVEY NAME");
    lblSurveyInfo.setForeground(new Color(155, 0, 108));
    lblSurveyInfo.setFont(new Font("Dialog", 1, 12));
    lblSurveyInfo.setAlignment(1);
    lblSurveyInfo.setText("ENTER SURVEY INFORMATION");
    txfSurveyName.setFont(new Font("Dialog", 0, 12));
    txfSurveyName.setText("Enter Technical Survey Name in
One Line");
    txfSurveyName.setColumns(80);
    txaSurveyInfo.setColumns(80);
    txaSurveyInfo.setFont(new Font("Dialog", 0, 12));
```

```
    txaSurveyInfo.setText("Enter Survey Information");
    txaSurveyInfo.setRows(5);
    this.setFont(new Font("Dialog", 1, 18));
    this.setSize(new Dimension(500, 400));
    this.addContainerListener(new
D_FF_3_this_containerAdapter(this));
    xYLayout1.setWidth(500);
    this.setLayout(xYLayout1);
    this.add(label1, new XYConstraints(34, 12, -1, -1));
    this.add(txfSurveyName, new XYConstraints(16, 90, 350,
-1));
    this.add(btnBrowse, new XYConstraints(387, 223, -1,
33));
    this.add(lblSurveyInfo, new XYConstraints(34, 146, 324,
-1));
    this.add(txaSurveyInfo, new XYConstraints(18, 175, 349,
135));
    this.add(btnSubmit, new XYConstraints(23, 340, 84,
32));
    this.add(btnClear, new XYConstraints(274, 338, -1,
32));
    this.add(lblSurveyName, new XYConstraints(104, 51, -1,
33));
  }

  //Start the applet
  public void start() {
  }

  //Stop the applet
  public void stop() {
  }

  //Destroy the applet
  public void destroy() {
  }

  //Get Applet information
  public String getAppletInfo() {
    return "Applet Information";
  }

  //Get parameter info
  public String[][] getParameterInfo() {
    String pinfo[][] =
    {
      {"SurveryName", "String", ""},
```

```
        {"SurveryInfo", "String", ""},
    };
    return pinfo;
}


//Main method
static public void main(String[] args) {
    D_FF_3 applet = new D_FF_3();
    applet.isStandalone = true;
    DecoratedFrame frame = new DecoratedFrame();
    frame.setTitle("Applet Frame");
    frame.add(applet, BorderLayout.CENTER);
    applet.init();
    applet.start();
    frame.pack();
    Dimension d =
Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation((d.width - frame.getSize().width) /
2, (d.height - frame.getSize().height) / 2);
    frame.setVisible(true);
}

void btnBrowse_actionPerformed(ActionEvent e) {
    openFile();
}

void openFile(){
    FileDialog fileDlg = new FileDialog(new Frame());
    fileDlg.setMode(FileDialog.LOAD);
    fileDlg.setVisible(true);

    if(fileDlg.getFile() != null){
      try{
        File file = new File(fileDlg.getDirectory() +
fileDlg.getFile());
            int size  = (int)file.length();
            char [] data = new char[size];
            int chars_read = 0;
            FileReader in = new FileReader(file);

            while(in.ready()){
              chars_read += in.read(data, chars_read, size -
chars_read);
            }
            in.close();
            txaSurveyInfo.setText(new String (data, 0,
chars_read));
```

```
        }
        catch(IOException e){
          txaSurveyInfo.setText( "Error Opening " +
fileDlg.getDirectory() + fileDlg.getFile() );
        }

    }
}

class D_FF_3_btnBrowse_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_3 adaptee;

  D_FF_3_btnBrowse_actionAdapter(D_FF_3 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnBrowse_actionPerformed(e);
  }
}

  void this_componentAdded(ContainerEvent e) {

  }

  void btnSubmit_actionPerformed(ActionEvent e) {
    /* D_FF_2 second = new D_FF_2();
    second.start();*/
    surveyName = txfSurveyName.getText();
    surveyInfo = txaSurveyInfo.getText();
  }

  void btnClear_actionPerformed(ActionEvent e) {
    txfSurveyName.setText("Enter Technical Survey Name in
One Line");
    txaSurveyInfo.setText("Enter Survey Information");
    surveyInfo = null;
    surveyName = null;
  }
}

 class D_FF_3_this_containerAdapter extends
 java.awt.event.ContainerAdapter {
   D_FF_3 adaptee;

   D_FF_3_this_containerAdapter(D_FF_3 adaptee) {
```

```
      this.adaptee = adaptee;
   }

   public void componentAdded(ContainerEvent e) {
      adaptee.this_componentAdded(e);
   }
}

class D_FF_3_btnSubmit_actionAdapter implements
java.awt.event.ActionListener {
   D_FF_3 adaptee;

   D_FF_3_btnSubmit_actionAdapter(D_FF_3 adaptee) {
      this.adaptee = adaptee;
   }

   public void actionPerformed(ActionEvent e) {
      adaptee.btnSubmit_actionPerformed(e);
   }
}

class D_FF_3_btnClear_actionAdapter implements
java.awt.event.ActionListener {
   D_FF_3 adaptee;

   D_FF_3_btnClear_actionAdapter(D_FF_3 adaptee) {
      this.adaptee = adaptee;
   }

   public void actionPerformed(ActionEvent e) {
      adaptee.btnClear_actionPerformed(e);
   }
}
```

Figure D.7: Source Code for "Task 3: Conduct Technological Survey," Developers' Fact Finding Phase for REPI, Part 1 of 3

```
<HTML>
<TITLE>
HTML Test Page
</TITLE>
<BODY>
<APPLET
   CODEBASE = "."
   CODE     = "Developer.D_FF_3.class"
   NAME     = "TestApplet"
   WIDTH    = 455
   HEIGHT   = 400
   HSPACE   = 0
   VSPACE   = 0
   ALIGN    = Middle>
<PARAM NAME = "SurveyName" VALUE = "">
<PARAM NAME = "SurveyInfo" VALUE = "">
</APPLET>
</BODY>
</HTML>
```

Figure D.8: Source Code for "Task 3: Conduct Technological Survey," Developers' Fact Finding Phase for REPI, Part 2 of 3

Figure D.9: Front End for "Task 3: Conduct Technological Survey," Developers' Fact Finding Phase for REPI, Part 3 of 3

```
//Title:      Requirement Elicitation Process through
Internet
//Version:
//Copyright:  Copyright (c) 1997
//Author:     Chetan Patel
//Company:    NJIT
//Description: CIS student
package Developer;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import borland.jbcl.layout.*;
import borland.jbcl.control.*;

public class D_FF_4 extends Applet {
  boolean isStandalone = false;
  String constraintName;
  String constraintInfo;
  String constraintAssesment;
  short constName;
  Label label1 = new Label();
  Label lblConstName = new Label();
  Choice chcConstName = new Choice();
  Label lblConstInfo = new Label();
  TextArea txaConstInfo = new TextArea();
  Label lblConstAsses = new Label();
  TextArea txaConstAsses = new TextArea();
  Button btnSubmit = new Button();
  Button btnClear = new Button();
  Button btnConstInfo = new Button();
  Button btnConstAsses = new Button();
  FileDialog fileDlg = new FileDialog(new Frame());
  XYLayout xYLayout1 = new XYLayout();

  //Get a parameter value
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) :
def);
  }

  //Construct the applet
  public D_FF_4() { }

  //Initialize the applet
```

```
  public void init() {
    try { constraintName =
this.getParameter("ConstraintName", "PC Based"); } catch
(Exception e) { e.printStackTrace(); }
    try { constraintInfo =
this.getParameter("ConstraintInfo", ""); } catch (Exception
e) { e.printStackTrace(); }
    try { constraintAssesment =
this.getParameter("ConstraintAssesment", ""); } catch
(Exception e) { e.printStackTrace(); }
    try { constName = (short)
Integer.parseInt(this.getParameter("ConstName", "0")); }
catch (Exception e) { e.printStackTrace(); }
    try { jbInit(); } catch (Exception e) {
e.printStackTrace(); } }
//Component initialization
  public void jbInit() throws Exception{
    btnSubmit.setForeground(Color.cyan);
    btnSubmit.setBackground(Color.darkGray);
    btnSubmit.setFont(new Font("Dialog", 1, 18));
    btnSubmit.setLabel("SUBMIT");
    btnSubmit.addActionListener(new
D_FF_4_btnSubmit_actionAdapter(this));
    btnClear.setForeground(Color.cyan);
    btnClear.setBackground(Color.darkGray);
    btnClear.setFont(new Font("Dialog", 1, 18));
    btnClear.setLabel("CLEAR");
    btnClear.addActionListener(new
D_FF_4_btnClear_actionAdapter(this));
    btnConstInfo.setForeground(new Color(155, 15, 0));
    btnConstInfo.setFont(new Font("Dialog", 3, 18));
    btnConstInfo.setLabel("BROWSE");
    btnConstInfo.addActionListener(new
D_FF_4_btnConstInfo_actionAdapter(this));
    btnConstAsses.setForeground(new Color(155, 15, 0));
    btnConstAsses.setFont(new Font("Dialog", 3, 18));
    btnConstAsses.setLabel("BROWSE");
    xYLayout1.setHeight(482);
    xYLayout1.setWidth(500);
    btnConstAsses.addActionListener(new
D_FF_4_btnConstAsses_actionAdapter(this));
    this.setSize(new Dimension(500, 400));
    this.addContainerListener(new
D_FF_4_this_containerAdapter(this));
    chcConstName.addItem("PC Based");
    chcConstName.addItem("Deterministic Model");
    chcConstName.addItem("Community Input");
```

```
    chcConstName.addItem("Presentation Information");
    chcConstName.addItem("Predictability");
    chcConstName.select(0);
    label1.setForeground(new Color(173,0,173));
    label1.setFont(new Font("Dialog", 1, 24));
    label1.setAlignment(1);
    label1.setText("TASK 4 : ASSESS CONSTRAINTS");
    lblConstName.setForeground(new Color(155, 0, 108));
    lblConstName.setFont(new Font("Dialog", 1, 12));
    lblConstName.setAlignment(2);
    lblConstName.setText("CONSTRAINT NAME");
    lblConstInfo.setForeground(new Color(155, 0, 108));
    lblConstInfo.setFont(new Font("Dialog", 1, 18));
    lblConstInfo.setAlignment(1);
    lblConstInfo.setText("CONSTRAINT INFORMATION");
    txaConstInfo.setRows(3);
    txaConstInfo.setColumns(80);
    lblConstAsses.setForeground(new Color(155, 0, 108));
    lblConstAsses.setFont(new Font("Dialog", 1, 18));
    lblConstAsses.setAlignment(1);
    lblConstAsses.setText("CONSTRAINT ASSESMENT  ");
    this.setLayout(xYLayout1);
    this.add(label1, new XYConstraints(38, 11, 410, 28));
    this.add(lblConstName, new XYConstraints(46, 68, 174,
25));
    this.add(chcConstName, new XYConstraints(244, 62, 155,
31));
    this.add(lblConstInfo, new XYConstraints(40, 122, 326,
-1));
    this.add(txaConstInfo, new XYConstraints(22, 158, 360,
84));
    this.add(lblConstAsses, new XYConstraints(33, 276, 332,
29));
    this.add(txaConstAsses, new XYConstraints(20, 310, 360,
86));
    this.add(btnSubmit, new XYConstraints(38, 415, 76,
37));
    this.add(btnClear, new XYConstraints(292, 415, -1,
40));
    this.add(btnConstInfo, new XYConstraints(399, 182, 88,
35));
    this.add(btnConstAsses, new XYConstraints(397, 334, 88,
34));
    }

  //Start the applet
  public void start() {
```

```
}

  //Stop the applet
  public void stop() {
}

  //Destroy the applet
  public void destroy() {
 }

  //Get Applet information
  public String getAppletInfo() {
    return "Applet Information";
  }

  //Get parameter info
  public String[][] getParameterInfo() {
    String pinfo[][] = {
      {"ConstraintName", "String", ""},
      {"ConstraintInfo", "String", ""},
      {"ConstraintAssesment", "String", ""},
      {"ConstName", "short", "This cant be removed later
on"},
    };
    return pinfo; }

  void btnConstInfo_actionPerformed(ActionEvent e) {
    openFile(txaConstInfo);
 }

  void btnConstAsses_actionPerformed(ActionEvent e) {
   openFile(txaConstAsses);
}

  void openFile(TextArea t){
    fileDlg.setMode(FileDialog.LOAD);
    fileDlg.setVisible(true);

    if(fileDlg.getFile() != null){
     try{
       File file = new File (fileDlg.getDirectory() +
fileDlg.getFile());
       int size = (int)file.length();
       char [] data = new char[size];
       int chars_read = 0;
       FileReader in = new FileReader(file);
```

```
        while(in.ready()){
          chars_read += in.read (data, chars_read, size -
chars_read);
          }
        in.close();
        t.setText(new String (data, 0, chars_read));
        }
      catch(IOException e){
          t.setText("ERROR Opening : " +
fileDlg.getDirectory() + fileDlg.getFile());
        }
      } // try
    } // if

    void btnSubmit_actionPerformed(ActionEvent e) {
      constraintName = chcConstName.getSelectedItem();
      constraintInfo = txaConstInfo.getText();
      constraintAssesment = txaConstAsses.getText();
      btnSubmit.setEnabled(false);
  }

void btnClear_actionPerformed(ActionEvent e) {
    constraintName = "";
    constraintInfo = "";
    constraintAssesment = "";

    chcConstName.select(0);
    txaConstInfo.setText("");
    txaConstAsses.setText("");
    btnSubmit.setEnabled(true);
  }
}

class D_FF_4_btnConstInfo_actionAdapter implements
java.awt.event.ActionListener{
  D_FF_4 adaptee;

  D_FF_4_btnConstInfo_actionAdapter(D_FF_4 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnConstInfo_actionPerformed(e);
  }
}
```

```
class D_FF_4_btnConstAsses_actionAdapter implements
java.awt.event.ActionListener{
  D_FF_4 adaptee;

  D_FF_4_btnConstAsses_actionAdapter(D_FF_4 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnConstAsses_actionPerformed(e);
  }
}


class D_FF_4_btnSubmit_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_4 adaptee;

  D_FF_4_btnSubmit_actionAdapter(D_FF_4 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.btnSubmit_actionPerformed(e);
  }
}


class D_FF_4_this_containerAdapter extends
java.awt.event.ContainerAdapter {
  D_FF_4 adaptee;

  D_FF_4_this_containerAdapter(D_FF_4 adaptee) {
    this.adaptee = adaptee;
  }
}


class D_FF_4_btnClear_actionAdapter implements
java.awt.event.ActionListener {
  D_FF_4 adaptee;

  D_FF_4_btnClear_actionAdapter(D_FF_4 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
```

```
        adaptee.btnClear_actionPerformed(e);
    }
}
```

Figure D.10: Source Code for "Task 4: Assess Constraints," Developers' Fact Finding Phase for REPI, Part 1 of 3

```
<HTML>
<TITLE>
HTML Test Page
</TITLE>
<BODY>
<APPLET
  CODEBASE = "."
  CODE     = "Developer.D_FF_4.class"
  NAME     = "TestApplet"
  WIDTH    = 400
  HEIGHT   = 300
  HSPACE   = 0
  VSPACE   = 0
  ALIGN    = Middle>
<PARAM NAME = "ConstraintName" VALUE = "PC Based">
<PARAM NAME = "ConstraintInfo" VALUE = "">
<PARAM NAME = "ConstraintAssesment" VALUE = "">
<PARAM NAME = "ConstName" VALUE = "0">
</APPLET>
</BODY>
</HTML>
```

Figure D.11: Source Code for "Task 4: Assess Constraints," Developers' Fact Finding Phase for REPI, Part 2 of 3
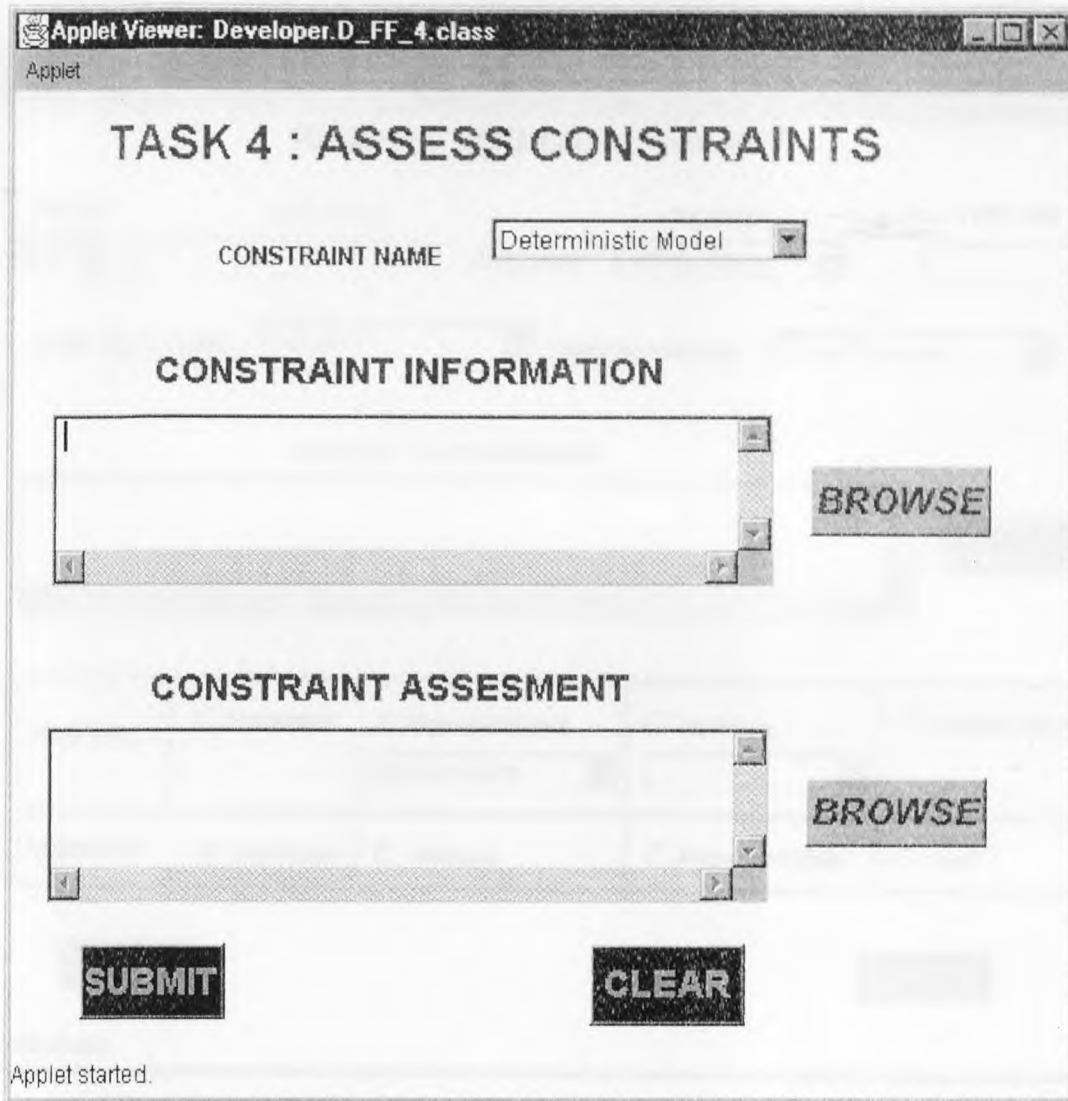
Figure D.12: Front End for "Task 4: Assess Constraints," Developers' Fact Finding Phase for REPI, Part 3 of 3

Figure D.13: Front End for "Task 3: Add Requirement," Developers' Gathering and Classification Phase for REPI
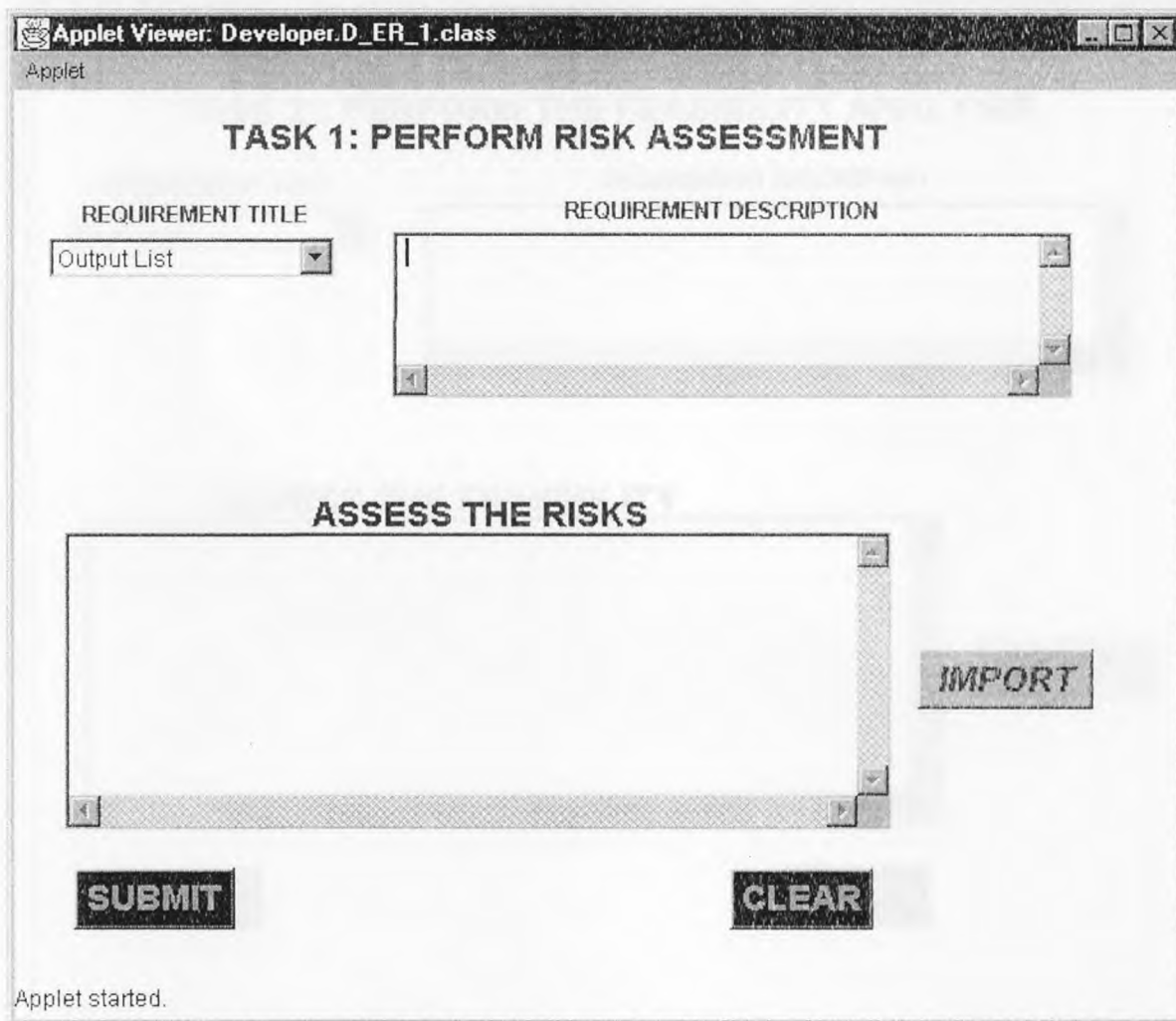
Figure D.14: Front End for "Task 1: Perform Risk Assessment," Developers' Evaluation and Rationalization Phase for REPI

Figure D.15: Front End for "Task 2: Perform the Feasibility Analysis," Developers' Evaluation and Rationalization Phase for REPI

Figure D.16: Front End for "Task 3: Cost/Benefit Analysis," Developers' Evaluation and Rationalization Phase for REPI

Figure D.17: Front End for "Task 1: Prioritize Requirements," Developers' Prioritize and Planning Phase for REPI
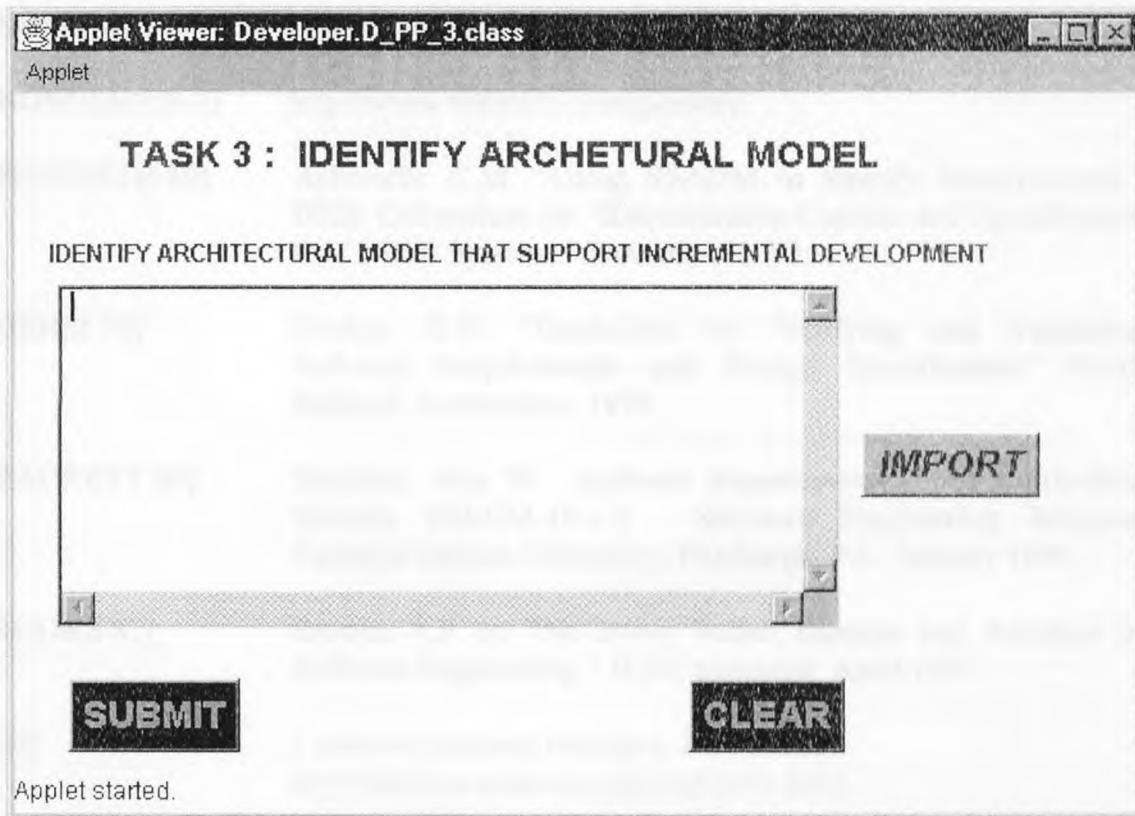
Figure D.18: Front End for "Task 3: Identify Architectural Model," Developers' Prioritize and Planning Phase for REPI

REFERENCES

[AL-RAWAS 96]   Al-Rawas, Amer and Easterbrook, Steve. "Communication Problems in Requirements Engineering: A Field Study." Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering. 1996.

[ANONYMOUS 1]   http://www.javasoft.com/white_paper.
                  Javasoft, Sun Microsystems Inc.

[ANONYMOUS 2]   http://www.sigma1.com/dev/dev_cswhite_paper.html.

[ANONYMOUS 3]   http://www.webcom.com/glossary.

[ASHWORTH 89]   Ashworth, C.M. "Using SSADM to Specify Requirements." IEEE Colloquium on "Requirements Capture and Specification for Critical Systems." November 1989.

[BOEHM 79]   Boehm, B.W. "Guidelines for Verifying and Validating Software Requirements and Design Specification." North Holland, Amsterdam. 1979.

[BRACKETT 90]   Brackett, John W. *Software Requirements*. SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. January 1990.

[BROOKS 87]   Brooks, F.P. Jr. "No Silver Bullet: Essence and Accident of Software Engineering." IEEE computer. April 1997.

[CGI]   *Common Gateway Interface: Introduction.*
                  http://hoohoo.ncsa.uiuc.edu/cgi/intro.html.

[CHECKLAND 89]   Checkland, Peter. *An Application of Soft Systems Methodology. Rational Analysis for a Problematic World.* John Wiley & Sons. 1989.

[CHRISTEL 92]   Christel, Michael G and Kyo C. Kang. "Issues in Requirements Elicitation." Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. September 1992.

[CONGRESS 90]   U.S. Congressional Subcommittee on Investigations and Oversight. "Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation." Technical Report 4/90 Staff Study, U.S. 101st Congress, Washington DC, DC. April 1990.

[DAVIS 93]   Davis, Alan M. *Software Requirements: Objects. Functions and States.* P T R Prentice Hall, Englewood Cliffs, NJ. 1993.

[DEEPAK 98]   Deepak Pandit. "Applications of Internet Technology for Requirements Elicitation." A Master's Thesis, New Jersey Institute of Technology, Newark, NJ. January 1998.

[DOD 91]   U.S. Department of Defense. "Software Technology Plan: Volume II Plan of Action." August 1991.

[GIRGENSOHN 96]   Girgensohn, Andreas. "Experiences in Developing Collaborative Applications Using the World Wide Web 'Shell'." *Hypertext 96: The Seventh ACM Conference on Hypertext.* 1996.

[GOSLING 95]   Gosling, James and McGilton, Henry. "The Java Language Environment." A White Paper. Sun Microsystems, Inc. 1995.

[HARWELL 93]   Harwell, Richard, et al. "What is a Requirement?" Published in the *Proceedings of the Third International Symposium of the INCOSE.*
                    http://internet-plaza.net/incose/workgrps/rwg/what_is.html.
                    1993.

[HERRMANN 96]   Herrmann, Eric. *Teach Yourself CGI Programming with Perl in a Week.* Sams.net, Indianapolis, Indiana. 1996.

[HOFMANN 93]   Hofmann, Hubert H. "Requirement Engineering- A survey of Methods and Tools." University of Zurich, Zurich. May 1993.

[IEEE 90a]   Institute of Electrical and Electronics Engineers. IEEE standard glossary of Software Engineering Terminology. IEEE standard 610.12-1990.

[IEEE 90b]   Institute of Electrical and Electronics Engineers. IEEE standard glossary of Software Engineering Terminology. IEEE standard 830-1984.

[JAMSA 96]   Jamsa, Kris. *Java Now!* Jamsa Press, 1996.

[JOHN 96]   John Gallaugher and Suresh Ramanathan. "The Critical Choice of Client Server Architecture: A Comparison of Two and Three Tier Systems." 1996.

[KAR 96] Kar, Pradip and Bailey, Michelle. "Characteristics of Good Requirements." Paper given at the 6th INCOSE Symposium. http://gate1.tlmworks.com/cai/incose.html. 1996.

[KEIL 95] Keil, Mark and Carmel, Erran. "Customer-Developer Links in Software Development." *Communications of the ACM*. Volume 38, Number 5. May 1995.

[KRAMER 96] Kramer, Douglas. "The Java Platform." A White Paper. Sun Microsystems, Inc. 1996.

[KEVIN 93] Kevin Hughes. "How does the Web work?" Honolulu Community College. October 1993.

[LARRY 98] Larry. "The Internet." IEEE Spectrum magazine. January 1998

[LEINER 97] Leiner, Barry M., Cerf, Vinton G., et al. *A Brief History of the Internet*. http://info-isoc.org/internet-history/. Feburary 1997.

[LEITE 87] Leite, J.C. "A Survey on Requirements Analysis." A technical report RTP-071, Department of Information and Computer Science, University of California, Irvine. 1987.

[LINDEN 96] Linden, Peter van der. *Just Java*. Sun Microsystems Inc. 1996.

[MAX 95] Dolgicer, Max: "When it's time for a TP Monitor; Client/Server Today." 1995.

[MCDERMID 89] McDermid, J.A. "Requirement Analysis : Problems the STARTS approach." Institution of Electrical Engineers. November 1989.

[MILLER 93] Miller, M. Greg and Tanik, Murat M. *Multimedia Applications in Software Engineering*. Technical Report 93-CSE-50. Southern Methodist University, Dallas, Texas. November 1993.

[MULLERY 89] Mullery, G.P. "Method Engineering: Methods via Methodology." IEEE Digest No. 138. November 1989.

[OCKER 95] Ocker, Rosalie, Hiltz, Starr Roxanne, et al. "The effects of distributed group support and process structuring on software requirements." *Journal of Management Information Systems*. Winter 95/96, Volume 12, Issue 3. 1995.

[ORACLE 96]   "Oracle Intranet Strategy." An Oracle White Paper. http://www.oracle.com/promotions/intranet/html/intranet_wp.html. Oracle Corporation, Redwood Shores, CA. July 1996.

[PATRICK 97]   Patrick, Naughton. *The Java Handbook.* McGraw Hill Publication. 1997.

[PLAYLE 96]   Playle, Greg and Schroeder, Charles. "Software Requirements Elicitation: Problems, Tools, and Techniques." http://www.stsc.hill.af.mil/crosstalk/1996/dec/xt96d12e.html. 1996.

[PRATIK 96]   Patel Patrik, Carl Moss, *Java Database Connectivity with JDBC,* Corrolios publication. 1996

[RAGHAVAN 94]   Raghavan, Sridhar, Zelesnik, Gregory, and Ford, Gary. *Lecture Notes on Requirements Elicitation.* Report CMU/SEI-94-EM-10. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 1994.

[RICHMOND 97]   Richmond, Alan and Richmond, Lucy. "HyperText Transfer Protocol." http://WWW.Stars.com/Internet/Protocols/HTTP/article.html. The Web Developer's Virtual Library. 1997.

[SAGE 90]   Sage, Andrew P., and Palmer, James D. *Software Systems Engineering.* John Wiley & Sons, New York. 1990.

[SHIFFMAN 97]   Shiffman, Hank. *Making Sense of Java.* http://reality.sgi.com/employees/shiffman_engr/Java-QA.html. Silicon Graphics, Inc. 1997.

[SOMMERVILLE 95]   Ian Sommerville, *Software Engineering,* Fifth Edition, Addison-Wesley Publishers Ltd. 1995.

[SOUTHWLLL 87]   Southwell, K., James, K., Clarke, B.A., Andrew,B., Ashworth, C., Norris, M., and Patel, V. "Requirement Definition and Design." The STARTS Guide, Second Edition. 1987.

[STARTS 87]   STARTS Guide, "Requirements Definition and Design." Manchester. 1987.

[STEP 91]   Software Test & Evaluation Panel (STEP), Requirements Definition Implementation Team. Operational requirements for Automated Capabilities, Draft Pamphlet (Draft PAM). April 1991.

[UMAR 97]   Umar Amjad, *Application (re)engineering, Building Web-Based Applications and Dealing with Legacies.* Bell Communications Research (Bellcore), Picataway, New Jersey. 1997.

[VONDRAK 97]   Vondrak, Cory. "Java." *Software Technology Review.* http://www.sei.cmu.edu/technology/str/descriptions/java_body. htm Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA. 1997.