Jaypee University of Information Technology
Waknaghat, Distt. Solan (H.P.)
Learning Resource Center

CLASS NUM: BOOK NUM.:

ACCESSION NO.: SP08041/SP0812041

This book was issued is overdue due on the date stamped below. If the book is kept over due, a fine will be charged as per the library rules.

Due Date	Due Date	Due Date

Remote PC Administration Suite For Mobile Devices

Gautam Vashisht (081415)
Nishant Chawla (081421)
Bhavini Rai (081426)
Divyanshu Gogia(081458)

Under the Supervision of
Mr. Pradeep Kumar Gupta

Learning Resource Center
Acc. No.
JUIT, Waknaghat, Solan (H.P.)

Learning Resource Center
JUIT
Waknaghat, Solan (H.P.)

Submitted in partial fulfillment
of the requirements for the degree of
BACHELOR OF TECHNOLOGY (IT)

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY WAKNAGHAT
SOLAN, HIMACHAL PRADESH
INDIA
2012

TALBLE OF CONTENTS

विद्या तत्व ज्योतिसमः

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

SOLAN, HIMACHAL PRADESH

CERTIFICATE

This is to certify that the work entitled **"REMOTE PC ADMINISTRATION SUITE USING MOBILE DEVICES"**, submitted by **Gautam Vashisht**, **Nishant Chawla**, **Bhavini Rai** and **Divyanshu Gogia** in partial fulfillment for award of degree of Bachelor of Technology (Information Technology) of Jaypee University of Information Technology has been carried out in my supervision. This work has not been submitted partially or wholly to any other university or institution for award of this or any other degree programme.

[illegible] 30/5/12

Project Supervisor

Mr. Pradeep Kumar Gupta

Sr. Lecturer (CSE & IT), JUIT

ACKNOWLEDGEMENT

We are highly grateful to **Brig.(Retd.) S.P. Ghrera**, Associate Professor and Head, Dept. of Computer Science & Engineering and IT, for providing us the opportunity to work on the project.

With great pleasure we express our gratefulness to our guide and mentor **Mr. P.K.Gupta**, Senior Lecturer, Dept. of Computer Science & Engineering and IT, for his valuable and sustained guidance and careful supervision during the project.

We express our sincere thanks to all the faculty members of JAYPEE UNIVERSITY OF INFORMATION AND TECHNOLOGY who have helped us directly or in directly. Without their help and guidance we would not have been able to successfully complete our project.

Group Members:

Gautam Vashisht [illegible]

Nishant Chawla [illegible]

Bhavini Rai [illegible]

Divyanshu Gogia [illegible]

List of Figures

Abstract

Unhindered connectivity to a computer is the need of the hour. The ability to connect to a remote machine and control it will have infinite advantages. J2ME (JAVA 2 Micro Edition) is a platform created to enhance application development for mobile phones. By sending the required commands over an internet Connection and by using J2ME platform we have created an application that which can control any desktop computer or a server.

Purpose

The main aim of the application is to remotely access and control different applications on a static IP PC by connecting to it over a GPRS link from a J2ME enabled cell phone. The possible applications are:

- Launching Winamp , WordPad, games or other similar applications.
- Shutting down and restarting or logging off from your machine.
- Formatting hard drives.
- Running Internet Explorer with the required URL.
- Sending Messages to the static PC.
- Remote Desktop Connection from the Cell Phone.

In the present paper we propose a solution to control a computer by creating an all time functional link between the system and its administrator.

Motivation

Many applications like web hosting services, network servers, automated systems need to be monitored continuously. And to monitor them 24/7 by being physically present at the location is not viable. Therefore we propose to control such applications remotely by J2ME enabled mobile devices.

Advantages Of A Remote PC Controller

Adding a mobile remote control to an existing network can have many advantages as mentioned below:

- Faster, decentralized decision making.
- Increased responsiveness to customers.
- Increased sensitivity to market changes.
- Lowered commuting costs/time for staff.
- Increased productivity.

Target Enterprises (Commercial Viability Of the Idea)

- Virtual Classrooms
- Online Presentations
- Sales and Marketing Meetings
- Remote Support to Servers
- Remote Access/Office

Chapter 1

Introduction

1.1. Introduction To Java Platforms

There are three flavors of Java

–J2ME (Java2 -Micro Edition),

–J2SE (Java2-Standard Edition), and

–J2EE (Java 2-Enterprise Edition).

These three flavours of Java represent three distinct target market segments each of which has unique issues and requirements that need to be addressed somewhat differently.

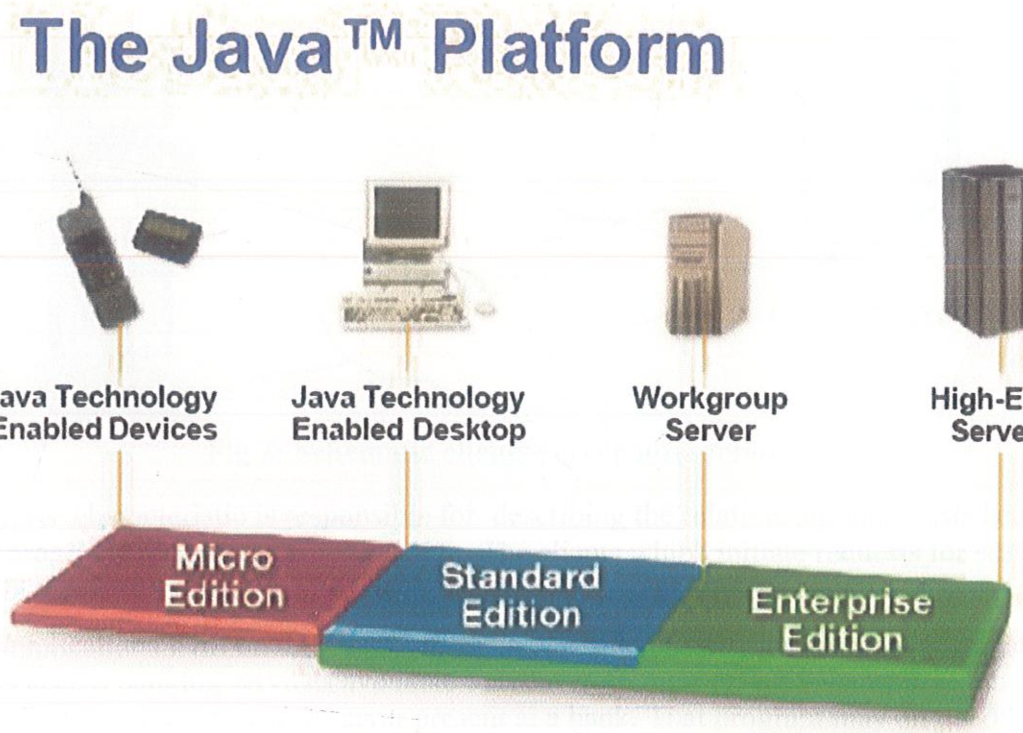


Fig 1: Java Platforms

Micro Edition addresses of market segment of small devices such as PDAs, cell phones and settopbox, which are typically constrained in terms of memory space and processing power.

Standard Edition represents the Java that we all know and love, a Java for desktop and workgroup server environments that require a full feature functionalities including rich graphical user interface.

Enterprise Edition covers the Java platform for developing and deploying enterprise quality applications which are typically transactional, reliable and secure.

1.2. Problem Statement

- We intend to build a Remote PC Administration Suite using J2ME which will access a static computer from a mobile device.
- Mobile will be able to access the screen of computer system.
- It will control the keyboard of static PC and will use to type anything on the static PC.
- It will have the mouse control function like clicking and moving the cursor.

1.3. Client Server Architecture

The **client/server model** is a type of a computing model which acts as distributed application. It partitions tasks or workloads among the providers of a resource or service, called servers, and service requesters, called clients. The clients and servers can be seen communicating over a computer network on some different hardware, however both client and as well as the server reside in a single system. The host is basically a server machine that is running one or more server programs which share their resources as well as services with clients. A client requests for the server's service function.

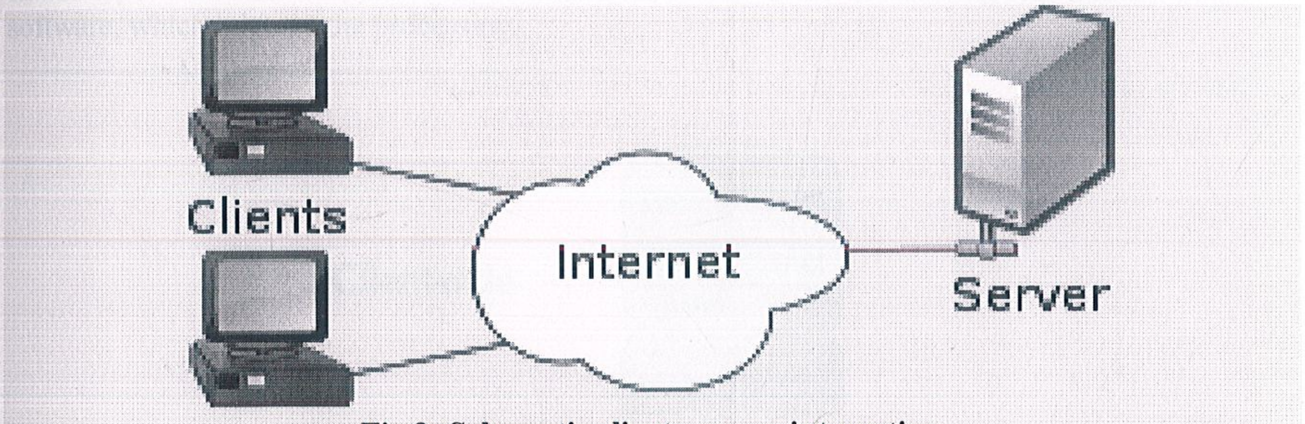


Fig 2: Schematic clients-server interaction

The *client/server* characteristic is responsible for describing the relationship that exists between the cooperating programs residing in an application. The clients which initiate requests for services are provided with the service function.

Example of applications built on the client server model: email exchange, web access and database access. Users access banking services available to them from their PC's by using a web browser client forwarding a request to a web server present at a bank. That program may on the other hand forwards the request to its existing database client program, which sends a request to a database server at another bank computer to retrieve the account information. The balance is returned to the bank database client, which in turn serves it back to the web browser client, displaying the results to the user. The client–server model has become one of the central ideas of network computing. Many business applications being written today use the client–server model, as do the Internet's main application protocols, such as HTTP, SMTP, Telnet, and DNS.

The interaction between client and server is often described using sequence diagrams. The Unified Modeling Language has support for sequence diagrams.

Specific types of clients include web browsers, email clients, and online chat clients.

1.4 Methodology

1.4.1. Basics

Remote Desktop Administration

- View a desktop remotely
- Interact with a remote desktop
- Make the remote desktop occupy the entire local screen
- Open a connection to a user's desktop, asking the user for permission before actually connecting
- Open a connection to a user's desktop without having the user be aware of your connection
- Browse for a specific user and connect to that user's desktop
- Browse the network for machines to connect to

1.4.2. Architecture (2 tier)

In this type of architecture, the client-side software is responsible for sending requests to server-side software, which initiates the processing.

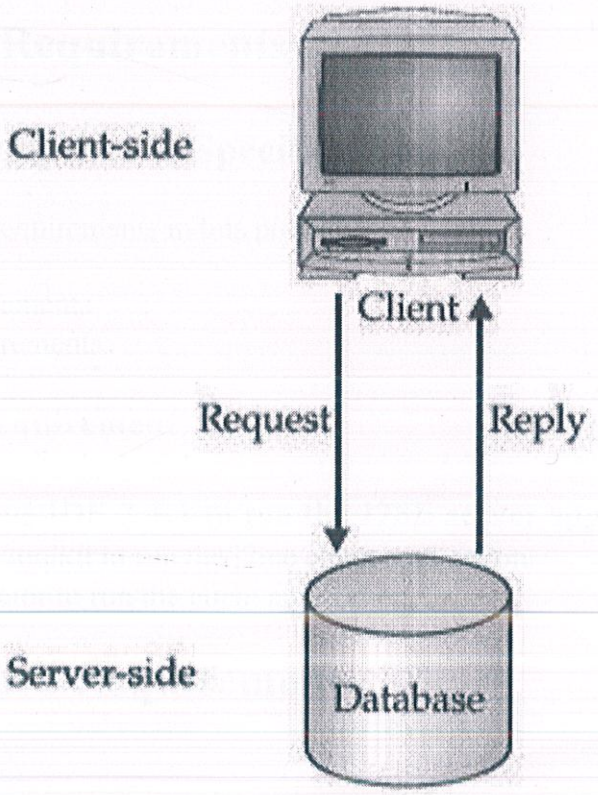


Fig 3: Client Server Architecture
Source: McGrawHill Osborne -Java J2ME The Complete Reference

1.5. Aims & Objectives

- This application is designed to provide the user Access a PC from his mobile device.
- This project is to enable the mobile users to control his PC information at their finger tips.
- The application must consider the limitations and disadvantages of mobile devices.
- The application must work properly without facing any hindarnce on all targeted devices, and in matters of power supply and network support.

1.6. Scope

Commercial Viability Of the Idea :

i) Virtual classrooms,

ii) Online presentations,

iii) Remote support to servers,

iv)Remote access to office workstation.

1.7. Resources Requirements

1.7.1. Types Description & Specifications

There are two types of requirements in this project:

- Software Requirements
- Hardware Requirements

1.7.1.1. Software Requirement

- We need Netbeans IDE 7.1.1 to run the J2SE server application.
- We need wireless toolkit to run the j2me client application.
- We need an emulator to run the client application.

1.7.1.2. Hardware Requirement

Computer system with

- Min 512 MB RAM
- Internet Connectivity
- Min 1Gb of free hard disk space

Mobile device which is enabled to work on Java platform.

Chapter 2

Moblie Application Development

2.1. What is J2ME?

The users of mobile and other small computing devices generally develop great expextations regarding the performance of these devices. The consumers of such devices demand multiple features like quick response time, full-featured applications in a small computing device and compatibility with companion services,.

This gives an additional responsibilty to developers as they need to rethink the way they tend to build computer systems. What the developers need to do is harness the power of existing front-end and back-end software found on business computers. Besides this thet need to transfer this power onto small, mobile, and wireless computing devices. J2ME allows the occurrence of this transformation with minimal changes, here it safely assumes that the design of the applications is scalable thereby, allowing it to be custom fitted.

An abstract version of the Java API and Java Virtual Machine is J2ME. J2ME has been designed as such that it operates in the new breed of embedded computers and microcomputers within the sparse resources available.

A notable benefit of using J2ME is that when it comes to compatibility it is compatible with all Java-enabled devices. A device is to be Java-enable if it runs the Java Virtual Machine. For instance, Ericsson, Motorola, Nextel, Nokia, Panasonic, and RIM all have Java-enabled devices.Besides this, J2ME also maintains the powerful security features that are found in the Java language enabling wireless and small computing devices to access resources that are present within an organization's firewall.

2.1.1 J2ME Configurations

J2ME has two types of configurations. They are **Connected Limited Device Configuration (CLDC)** and the **Connected Device Configuration (CDC)**. The CLDC is basically designed for small computing devices which are 16 bit or 32 bit and have limited amounts of memory. The available memory in CLDC devices are usually between 160KB and 512KB and they are battery powered. They generally do not have a user interface and use an inconsistent, small-bandwidth network wireless connection. CLDC devices make use of the KJava Virtual Machine (KVM) implementation, which is in reality, a stripped-down version of the JVM.

CLDC devices include pagers, personal digital assistants, cell phones, dedicated terminals, and all consumer devices that have memory of 128KB and 512KB. CDC devices on the other hand use a 32-bit architecture, memory availability of atleast two megabytes , and a complete functional JVM is implemented. CDC devices include digital set-top boxes, navigation systems, point-of-sale terminals, home appliances, and smart phones.

2.1.2 J2ME Profiles

A profile usually contains Java classes that allow implementation of features for a particular small computing device. We have defined seven profiles. These are the Foundation Profile, Game Profile, Mobile Information Device Profile, PDA Profile, Personal Profile, Personal Basis Profile, and RMI Profile.

- The Foundation Profile contains core Java classes and hence is used along with the CDC configuration and is the core for nearly all other profiles used with the CDC configuration
- The Game Profile is also used for developing game applications and contains the necessary classes that required for developing the same.
- The Mobile Information Device Profile (MIDP) contains all the classes that are responsible for providing local storage, a user interface, and networking capabilities to an application which is running on a mobile computing device for instance Palm OS devices. MIDP is used along with all the wireless Java applications.
- The PDAProfile (PDAP) is used along with the CLDC configuration. It stores all those classes that utilize sophisticated resources found on personal digital assistants. The features being reffered here are better displays and larger memory.
- The Personal Profile makes use of the CDC configuration and the Foundation Profile and handles the classes which implement a complex user interface. The core classes are provided by the Foundation Profile , and the Personal Profiles are responsible for providing classes to implement a sophisticated user interface. A sophisticated user interface is a user interface which is capable of displaying multiple windows at a time.
- The Personal Basis Profile is very much similar to the Personal Profile. It is used with the CDC configuration and the Foundation Profile.The Personal Basis Profile provides classes to implement a simple user interface. A simple user interface is a user interface that is capable of displaying one window at a time.
- The RMI Profile is used with the CDC configuration and the Foundation Profile. It provides Remote Method Invocation classes to the core classes that are contained in the Foundation Profile.

2.1.3 CLDC

The Connected Limited Device Configuration (CLDC) is the basic building block on which the J2ME profiles for small devices, such as cell phones, pagers, and low-end PDAs, are built. These devices are characterized by their limited memory resources and processing power, which make it impossible for them to host a fully featured Java platform. CLDC specifies a minimal set of Java packages and classes and a reduced functionality Java virtual machine that can be implemented within the resource constraints imposed by such small devices

2.1.3.1 The CLDC Java Virtual Machine

The hardware and software limitations imposed by the devices at which CLDC is targeted make it impractical to support either a full Java virtual machine or a complete set of J2SE core classes. Running a simple "Hello, world" application on the Windows platform requires around 16 MB of memory to be allocated. Contrast this with the minimum platform requirements for CLDC, which call for:

• 128 KB of ROM, flash or battery-backed memory for persistent storage of the Java VM and the class libraries that make up the CLDC platform.

• 32 KB (or more) of volatile memory to be available for runtime allocation.

This memory is used to satisfy the dynamic requirements of Java applications, which include class loading and the allocation of heap space for objects and the stack. In order to support a Java runtime environment with such limited resources, CLDC defines reduced requirements for the virtual machine, the language itself, and the core libraries.

Other than the memory requirements, CLDC makes few assumptions about its host platform. It does not, for example, assume that the device will have any kind of display or user input mechanism such as a keyboard or a mouse, and it does not require any kind of local storage for application data. These issues are all assumed to be addressed individually by each device vendor. As far as the software environment is concerned, CLDC assumes only that the host device has some kind of operating system that can execute and manage the virtual machine.

2.1.3.2 The CLDC Class Libraries

CLDC addresses a wide range of platforms that do not have sufficient memory resources to support the full range of packages and classes provided by J2SE. Because CLDC is a configuration rather than a profile, it cannot have any optional features. Therefore, the packages and classes that it specifies must have a small enough footprint that they can be hosted by devices that meet only the minimum requirements of the CLDC specification. The CLDC class library is very small -- it is composed of a package containing functionality that is specific to J2ME (called javax.microedition.io), along with a selection of classes from the following packages in the core J2SE platform:

- java.io
- java.lang
- java.util

All J2ME configurations and profiles include packages or classes from J2SE. When J2ME incorporates software interfaces from J2SE, it must follow several rules:

• The names of the packages or classes must be the same, wherever possible. It would not be acceptable, for example, to completely reimplement the java.lang package in a package called javax.microedition.lang if the API in the java.lang package can be used.

• The semantics of classes and methods that are carried over into J2ME must be identical to those with the same name in J2SE.
• It is not possible to add public or protected fields or methods to a class that is shared between J2SE and J2ME.

Because of these rules, J2ME packages and classes will always be a subset of the packages and classes of the same name in J2SE, and the J2ME behavior will not be surprising to developers familiar with J2SE. Furthermore, J2ME configurations and profiles are not allowed to add extra functionality in packages and classes that they share with J2SE, so upward compatibility from J2ME to J2SE is preserved.

2.1.4 The Mobile Information Device Profile and MIDlets

MIDP is based on CLDC and KVM and is that version of the Java platform which is aimed at small devices, mostly cell phones and two-way pagers.

The software that implements MIDP runs in the KVM supplied by CLDC and provides additional services for the benefit of application code written using MIDP APIs. MIDP applications are called *MIDlets*. MIDlets can directly use both MIDP facilities and the APIs. MIDlets do not access the host platform's underlying operating system and cannot do so without becoming nonportable. Because the KVM does not support JNI, the only way for a MIDP application to access native platform facilities directly is by linking native code into a customized version of the virtual machine.

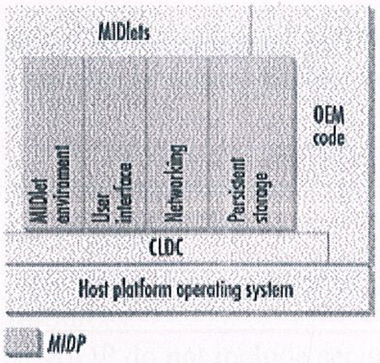


Fig 4: The Mobile Information Device Profile
Source: J2ME in a nutshell (O'Reilly JAVA)

2.1.4.1 The MIDP Java Platform

The Java platform available to MIDlets is that which has been provided to it by CLDC. The MIDP specification places the following mentioned requirements on the core libraries:

• MIDlets are managed in an execution environment just like applets. This environment is slightly different from that of a Java application. The foremost entry point to a MIDlet is not the main( ) method of its MIDlet class. The exit( ) methods present and available in both the System and Runtime classes are therefore expected to throw a SecurityException if they are invoked.

• MIDP devices are expexted to set the microedition.locale property so that it reflects the locale in which the device is supposed to operate. The locale names are formed in a different way in case of MIDP than from those used by J2SE, as the language and country components instead of being separated by an underscore character are separated by a hyphen. For instance : en-US on a MIDP device, meanwhile a J2SE developer would be expecting the locale name to be present in the form en_US.

• The system property microedition.profiles is supposed to contain at least the value MIDP- 1.0.

2.1.4.2 MIDlets and MIDlet Suites

MIDlets are Java applications running on MIDP devices.

A MIDlet suite is composed of a group of related MIDlets. All the MIDlets that exist in a suite are packaged together and installed onto a device. Though they are installed as a single entity they can be uninstalled only as a group. The MIDlets in a suite share not only the static but also the runtime resources of their host environment, as follows:

•All active MIDlets from a MIDlet suite run in a single Java VM if at the runtime the device supports concurrent running of more than one MIDlet . All MIDlets that belong to the same suite therefore share the same instances of all Java classes. This clearly means that data is allowed to be shared between MIDlets.

• At the MIDlet suite level, storage on the device is managed . MIDlets can access not only their own persistent data but also of other MIDlets in the same suite.

2.1.4.3 MIDlet Security

The Java security model that is used in J2SE not only powerful but also flexible. However it is expensive when we talk about memory resources.
This is the reason why neither CLDC nor MIDP do not include security checking of API calls that is available in J2SE.

This implies that a MIDlet appears to be more of a potential threat than an applet would to a browser user, for a mobile device user. This is because the MIDlet is not cotrolled by the Java applet "sandbox".Therefore when installing MIDlets a mobile device owner needs to be cautious and.
This however is not possible as the user is not completely sure at the time of writing of who is providing a MIDlet. At present, there is very limited security available and existing as of now against malicious MIDlets. In the existing scenario no MIDlet APIs are available that allow any access to information that is already presentnon the device, such as name, and telephone number list or place of work.
A MIDlet is allowed to store information on a device however that storage is private to that particular MIDlet and its suite, so the MIDlet can harm only the data that is present in it.

2.1.4.4 MIDlet Execution Environment and Lifecycle

The abstract base of class javax.microedition.midlet.MIDlet is used for picking up the MIDlets. This base class contains methods which are called by the MIDP platform to not only control the lifecycle of a MIDlet, but also control methods which the MIDlet itself uses to ask for any change in its state. It is important that a MIDlet must contain a public default constructor. A public default constructor is a constructor having no arguments. This is supposed to be developer supplied if there exists initialization that has to be performed. Also the Java compiler inserts an empty default constructor when there are no explicit constructors present. A skeleton MIDlet class somewhat looks like:

```
public class MyMIDlet extends MIDlet {
// Optional constructor
MyMIDlet( ) {
}
protected void startApp( ) throws MIDletStateChangedException {
}
protected void pauseApp( ) {
}
protected void destroyApp(boolean unconditional)
throws MIDletStateChangedException {
}
}
```

At any given time, a MIDlet is in one of three states: Paused, Active, or Destroyed. A state diagram that shows how these states are related.

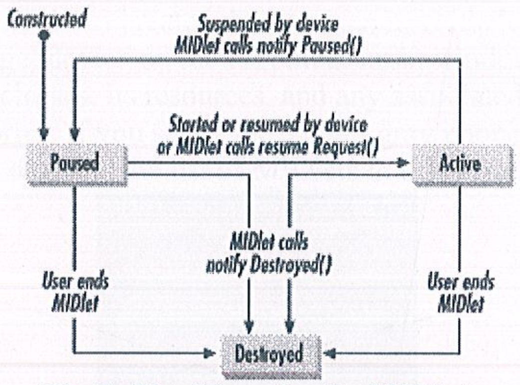


Fig 5: The lifecycle of a MIDlet
Source: J2ME in a nutshell (O'Reilly JAVA)

Initially when a MIDlet is loaded, it is supposed to be in the Paused state. When the MIDlet instance is created, it begins with the invoking all the instance initializers which is followed by invoking its public no-argument constructor. If during the period of execution of its constructor, the MIDlet somehow throws an exception, the MIDlet will be easily destroyed. The state of the MIDlet changes from Paused to Active, if the MIDlet fails to throw an exception and is thereafter scheduled for execution at some later time. This initiates a method called startApp( ). This method has been declared by the MIDlet class as follows:

```
protected void startApp( ) throws MIDletStateChangeException;
```

Since this method is abstract which implies that it must be implemented it in the MIDlet. The fact that this method is protected highlights the fact that either a MIDlet class will be used to call it or another class javax.microedition.midlet package will be used to call it. The MIDlet lifecycle methods can be called from a class in this package available called Scheduler, this when we are talking about the reference implementation. Although there is hardly anything that is present in the MIDP specification that will require this class to be used.
The startApp( ) method is defined as public by the MIDlet developers, however it should not be declared as protected.

The startApp( ) method may complete normally, in which case the MIDlet is allowed to run, or it may inform the MIDP platform that the MIDlet does not want to run at this point. There are several ways to achieve the latter:

• If the startApp( ) method detects If an error condition is detected by the startApp() method, that will prevent the method from completing, which however may not exist later altogether, it should supposedly throw an exception named MIDletStateChangeException.

• The notifyDestroyed( ) method is called when the startApp( ) method detects an error condition from which recovery is very difficult.

• In the case, a method invoked by the MIDlet does not throw an exception a MIDletStateChangeException is thrown. The MIDlet in that case is destroyed by calling its destroyApp( ) method after it is safely assumed that a fatal error has occured.

2.1.4.5 Running a MIDlet

At this stage, the JAR file has not been created, but you can nevertheless test the MIDlet suite by selecting an appropriate target device on the KToolbar main window and pressing the Run button. This loads the MIDlet classes, its resources, and any associated libraries from the *classes*, *res*, and *lib* subdirectories. If you select the default gray phone and press the Run button, the emulator starts and displays the list of MIDlets in this suite.

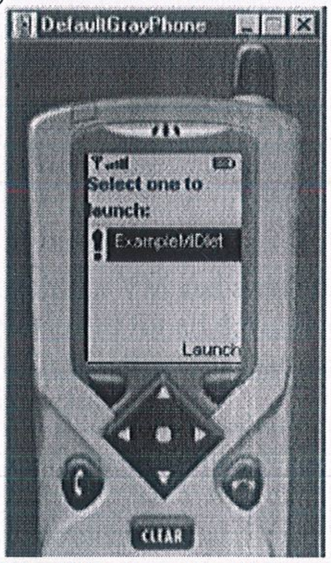


Fig 6: The Wireless Toolkit emulator
Source: J2ME in a nutshell (O'Reilly JAVA)

When the MIDlet suite is loaded, the device's application management software displays a list of the MIDlets that it contains and allows you to select the one you want to run. In this case, even though the suite contains only one MIDlet, the list is still displayed. Given the current lack of security for MIDlets imported from external sources, it would be dangerous for the device to run a MIDlet automatically, and, by giving the device user the chance to choose a MIDlet, it allows him the opportunity to decide not to run any of the MIDlets if, for any reason, they are thought to be a security risk or otherwise unsuitable. It is not obvious, though, on what basis such a decision would be made, since the user will see only the MIDlet names at this stage, but requiring the user to confirm that a MIDlet should be run transfers the ultimate responsibility to the user. In this case, the device displays the MIDlet name and its icon (the exclamation mark) as taken from the MIDlet-1 attribute in the manifest file. The device is not obliged to display an icon, and it may use its own icon in preference to the one specified in the manifest.

When we run the MIDlet suite this way, the Wireless Toolkit compiles the source code with the option set to save debugging information in the class files, and it does not create a JAR file. If you want to create a JAR, you can do so by selecting the Package item from the Project menu. This rebuilds all the class files without debugging enabled, which reduces the size of the class files, a measure intended to keep the time required to download the JAR to a cell phone or PDA as small as possible. It also extracts the content of any JARs or ZIP files it finds in the *lib* subdirectory and includes them in the MIDlet JAR, after running the preverifier over any class files that it finds in these archives. The JAR can be used, along with the JAD file, to distribute the MIDlet suite for installation into a device over a network.

2.1.5 The CDC

The CDC is targeted at devices that have a minimum of 2 MB of memory available to be used by the Java VM and its class libraries. As with CLDC, most devices probably have the VM and the core class libraries in ROM or Flash memory, but they also require RAM for application classes (unless the application is embedded and hence also included in the ROM) and the Java heap.

CDC devices typically have a 32-bit processor and a network connection, which may be intermittent or permanent, often directly to the Internet or a TCP/IP-based intranet. This contrasts to the CLDC environment, which is often hosted by slower 16-bit processors, and which has only a relatively low-bandwidth, nonpermanent connection to a network that cannot be assumed to support TCP/IP.

Like CLDC, the CDC specification requires a VM and a set of class libraries represent the minimal subset of the Java 2 platform required for all devices to which this configuration is targeted. Devices built to target specific applications or markets require additional software facilities that are provided by CDC's associated profiles.

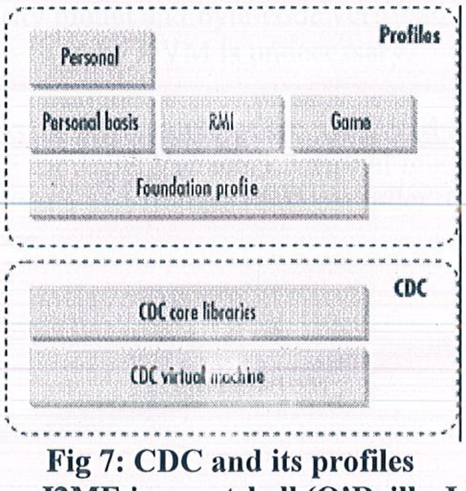


Fig 7: CDC and its profiles
Source: J2ME in a nutshell (O'Reilly JAVA)

2.1.5.1 The CDC Virtual Machine

Because CDC devices are much more capable than those targeted by CLDC, they can support a full Java VM. In fact, any VM provided as part of a CDC implementation must provide all the features described in the second edition of the Java Virtual Machine specification.

The CDC reference implementation contains the source code for the CVM and the core CDC Java class libraries. The reference implementation can be compiled for Linux (strictly speaking, only Red Hat Linux Version 6.2 is supported) and VxWorks, a real-time operating system. However, CVM is designed to be highly portable, and the download includes documentation that covers the details of the porting layer for those who need to implement it for a different platform. Perhaps somewhat surprisingly, Sun does not provide a version of CVM for PocketPC platforms such as the Compaq iPAQ range of PDAs, which would be an ideal host for a Java 2 programming environment. Third party support for these devices is almost certain to appear, however, when the GUI-based profiles
become available.

CVM uses the same ROMizing feature used by KVM to reduce VM startup time and minimize memory usage by building a prelinked set of Java classes directly into the VM. The reference implementation produces a CVM prelinked with most of the core CDC classes and, optionally, some the classes in the Foundation Profile.

Since CVM is a full virtual machine, the VM and the core libraries include many features that are not available in the KVM, including the following:

- Floating-point byte codes and data types
- Native code execution using the Java Native Interface
- Weak references
- Reflection
- Object serialization
- Developer-defined class loaders
- Java Virtual Machine Debugging Interface (JVMDI) support

The availability of JVMDI means that it is possible to connect a debugger to the CVM without the use of the debug proxy agent required by the KVM. The CDC platform also incorporates the full Java 2 security model and byte-code verification, which means that the off-device preverification process used by KVM is unnecessary.

Despite the fact that the CVM has all the features of the J2SE VM as defined by the JVM specification, it is *not* the same as the J2SE Version 1.3 virtual machine. In particular, it does not have hotspot technology or even a just-in-time (JIT) compiler. CVM is strictly a byte-code interpreter, albeit an optimized one.

2.1.5.2 CDC Class Libraries

Unlike CLDC, a class included in CDC is unchanged from its J2SE counterpart, unless it has deprecated APIs. Because there is no legacy CDC application code to support, there is no requirement for backward compatibility, and, therefore, the opportunity has been taken to remove APIs that are deprecated in J2SE Version 1.3, whenever there is an alternative available. In general, however, working with CDC or a CDC-based profile is much closer to using a full J2SE Version 1.3 platform than CLDC,

The CDC specification includes a minimal set of core Java classes that provide the common functionality required by every CDC platform. According to the specification, the core libraries represent little more than the minimum needed to support a Java VM. They include classes from the following packages:

java.io
java.lang
java.lang.ref
java.lang.reflect
java.math
java.net
java.security
java.security.cert
java.text
java.util
java.util.jar
java.util.zip
javax.microedition.io

Learning Resource Center
Acc. No. SP08041
SP08120411
JUIT, Waknaghat, Solan (H.P.)

The following paragraphs briefly cover the differences between the CDC packages and their J2SE counterparts.

The java.io package
Most of the J2SE classes in this package are included in CDC, with the exception of some of the less commonly used Reader and Writer subclasses, as well as LineNumberInputStream and StringBufferInputStream, both of which are deprecated in J2SE.

The java.lang package
In this package, only the Compiler class and UnknownException have been omitted.

The java.lang.ref package
Complete.

The java.lang.reflect package
Complete.

The java.math package
This package contains only two classes in J2SE. The CDC version includes BigInteger but excludes BigDecimal.

The java.net package
CDC provides the classes necessary to support datagrams (i.e., the UDP protocol), but it does not support sockets (i.e., TCP) or HTTP and therefore omits classes that relate to these two features. URL-based operations can be used, provided they do not rely on HTTP or sockets. This means, for example, that file and jar-based URLs are allowed, but http URLs are not.

The java.security package
Only those parts of the java.security package that deal with handling fine-grain security for Java classes is included, together with minimal support for creating and checking message digests.

The java.security.cert package
Contains only the Certificate class and two certificate-related exception classes. This package is of limited use because it does not include any concrete certificate implementations (such as X509Certificate).

The java.text package
The CDC java.text package provides support for locale-specific formatting, parsing of numbers and dates, and formatting of error messages. Classes that support advanced locale-sensitive collation and attributed character strings are omitted.

The java.util package
This useful package is almost complete in CDC. The only omissions are classes that relate to event handling (such as Observer and EventObject) and timers. Unlike CLDC, CDC includes both the JDK 1.1 and Java 2 collection frameworks.

The java.util.jar package
This package is complete, apart from the JarOutputStream class, which means that it is possible to read but not create a JAR file. This distinction is possible because, although the VM has to be able to load Java classes and other resources from a JAR file, it never needs to write to one.

The java.util.zip package
This package contains the classes that are necessary for the VM to read from a compressed or uncompressed ZIP file, but it omits the classes that allow writing or provide streams that handle compression and decompression of data for the benefit of applications. Compressed ZIP files are supported by virtue of the inclusion of the Inflater class.

The javax.microedition.io package
This package is provided for upward compatibility with applications written for CLDC. It contains the classes and interfaces that make up the Generic Connection Framework and includes support for datagrams. Interestingly, the StreamConnection and StreamConnectionNotifier classes, which are intended for support of TCPbased sockets, are included, even though the java.net package excludes socket support, and a CDC implementation is not required to allow socket communication. Furthermore, in the reference implementation, it is possible to connect using a GCF socket URL. The HttpConnection class is, however, not included.

2.1.5.3 CDC Profiles

At the time of writing, CDC has only one profile, the Foundation Profile, for which a reference implementation is available. Another, the RMI profile, has been specified, but an implementation has not yet been released. Three others are still in the process of being specified. This section provides an overview of the Foundation Profile and touches briefly on the remaining profiles, which are currently of little practical use because there are no implementations available.

The Foundation Profile

Most of the CDC profiles are based on the Foundation Profile, which adds to the minimal facilities of the CDC core libraries in much the same way that MIDP extends CLDC. This profile fills many of the gaps in the basic CDC class libraries by supplying most of the omitted classes from the packages that CDC supports; it also adds many of the other J2SE packages that are not included by CDC. The most important omissions from the Foundation Profile are the user interface classes, which are not required on all devices and which are instead provided by the Personal Basis and Personal profiles that are layered on top of the Foundation Profile.

The packages in the Foundation Profile include all the classes from their J2SE counterparts. The following packages are provided:

java.io (but not LineNumberInputStream and StringBufferInputStream, which are deprecated in J2SE)
java.lang
java.lang.ref
java.lang.reflect
java.math
java.net
java.security
java.security.acl
java.security.cert
java.security.interfaces
java.security.spec
java.text
java.util
java.util.jar
java.util.zip

The Foundation Profile also supports all of the javax.microedition.io package, including HTTP connections.

The RMI Profile

The RMI profile adds a subset of the J2SE Remote Method Invocation facility on top of the Foundation Profile. Since CDC devices are typically used in the role of the RMI client, only the client RMI functionality is included in this profile. At the time of writing, the RMI profile is available only in the form of a specification. There is, as yet, no reference implementation.

2.1.6 KVM

In order to provide Java-level debugging facilities, hooks must be supplied by the Java VM so that a debugger can perform tasks such as placing breakpoints, inspecting and modifying objects, and arranging to be notified when a debugging-related event occurs within the VM. The Java 2 platform includes an architecture, called the Java Platform Debugger Archicture (JPDA), that defines the debugging features that must be provided by a VM and the way in which they can be accessed by a debugger.

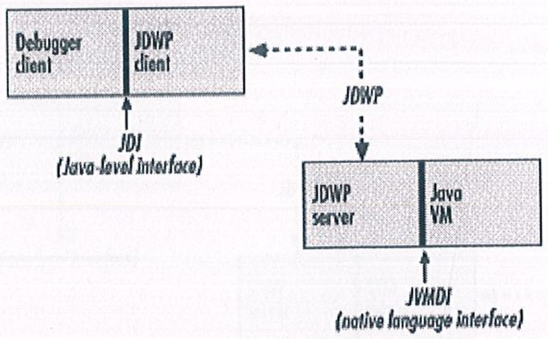


Fig 8: The Java 2 Platform Debugger Architecture
Source: J2ME in a nutshell (O'Reilly JAVA)

2.1.6.1 The JPDA

In the JPDA, the debugger interacts with the Java VM using a well-defined protocol called the *Java Debug Wire Protocol* (JDWP). This protocol specifies messages that are passed from a JDWP client to a JDWP server to request that operations be performed on the target VM, corresponding to debugging commands issued by the user. It also defines events that can be transmitted in the opposite direction to notify the debugger of state changes within the VM.

The architecture separates the debugger and the JVM from the details of the wire-level protocol by inserting an insulating layer on each side of the JDWP; this layer takes care of mapping the protocol messages to and from the programming interfaces required by the debugger and provided by the VM. In order to make it possible to accomodate different VM or debugger implementations without requiring each of them to provide their own JDWP implementation, two internal APIs are defined:

The Java Debug Interface (JDI)
The JDI is a Java-level interface that exposes the services of a JDWP client to a debugger. Typically, the debugger is a GUI program written by a third party vendor, but it could provide a command-line interface (such as that provided by the jdb command in the SDK). Debuggers using this interface can be assured that they will

work with any JVM written to conform to the JPDA.

The Java Virtual Machine Debug Interface (JVMDI)
JVMDI is the interface exposed by the JVM itself to allow operations received by the JDWP to be performed and to report VM state changes to the JDWP server. Unlike JDI, JVMDI is a native language interface because it requires low-level access to the virtual machine.

2.1.6.2 The KVM Implementation of the JPDA

The CLDC specification does not place any requirements for debugging support within the VM, but a practical VM implementation needs to provide some kind of debugging capability. The KVM has debugging support, but resource constraints make it impossible to fully implement the server side of the JDWP protocol and the hooks within the KVM itself. Instead, this functionality is divided between the VM and another process called the *KVM debug proxy* (or KDP).

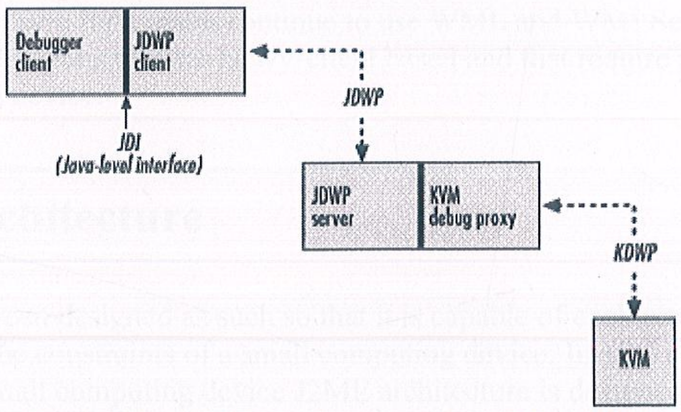


Fig 9: The KVM implementation of the JPDA
Source: J2ME in a nutshell (O'Reilly JAVA)

The function of the debug proxy is to implement features of the JDWP that are too resource intensive
to be placed within the KVM process itself. Normally, the debug proxy is not run on the same device as the KVM itself, so it does not require device resources. Instead, the debug proxy might be executed on a desktop system and communicate with the KVM using a specially designed variant of JDWP called the KVM Debug Wire Protocol (KDWP), carried over a socket connection. The definition of the KDWP can be found in the *KVM Debug Wire Protocol Specification*, which is included with the CLDC reference implementation.

2.1.7 J2ME and Wireless Devices

The sharp increase in use of mobile communications products such as cell phones has brought about a demand for applications that can easily run on those devices. Corporations along with the consumers desire and wish of bringing about an expantion in mobile communications devices from voice communications to applications that are traditionally found on laptops and computers.

The Wireless Application Protocol or the WAP forum has set out to create all together different standards for wireless technology. The WAP forum created theWAP standard which describes mobile communications device standards. TheWAP standard is basically nothing but an enhancement of HTML and XML. The Wireless Markup Language specification is an element of this standard. It comprises largely of HTML and XML and is used worldwide by developers all across to create documents that can be displayed with the help of a microbrowser. A *microbrows* operates on a mobile communications device and is a dimunitive web browser.Sophisticated applications require the device to process information that are beyond the capabilities of theWAP specification. This is important for the devices that have been designed for mobile communications devices. J2ME is responsible for providing the standard that is required to fill this gap. For instance, a sales representative wishing to check the list of available flights and hotel accommodations, will first purchase an airline ticket, secondly he\she will book the hotel as expected, which will be followed by sending the itinerary to a client, this while being present in a taxi in traffic.

J2ME applications that are referred to as a *MIDlet* are capable of running on practically all types of mobile communications device that successfully implement a JVM and MIDP. This fact greatly aspires and convinces developers to invest more time and money in building these type of applications.
All the applications that have light client continue to use WML and WMLScript. Developers intend to turn to J2ME for applications that are heavy-client based and that require processing on the mobile communications device.

2.1.8 J2ME Architecture

J2ME architecture has been designed as such so that it is capable of enableing an application to be scaled based solely on the constraints of a small computing device. Instead of replacing the operating system of a small computing device J2ME architecture is designed in layers that are located above the native operating system, and are collectively referred to as the Connected Limited Device Configuration (CLDC). The CLDC provides the run-time environment for small computing devices. CLDC is installed on top of the operating system.

The J2ME architecture is composed of three software layers. Java Virtual Machine (JVM) is the first configuration layer , that directly interacts with the native operating system. All the interactions between the profile and the JVM are handled by the configuration layer. Profile layer which is the second layer, consists of the minimum set of application programming interfaces (APIs) for the small computing device. Mobile Information Device Profile (MIDP) is the third layer in the architecture. The MIDP layer has access to CLDC libraries and MIDP libraries.It also contains Java APIs for user network connections, persistence storage.

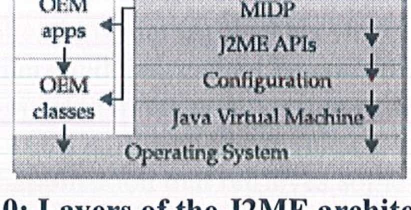


Fig 10: Layers of the J2ME architecture
Source: McGrawHill Osborne -Java J2ME The Complete Reference

2.1.9 Emulator: The J2ME Wireless Toolkit Emulator

The *emulator* command provides the execution environment and application management software for the J2ME Wireless Toolkit. Its functionality and command-line interface are both very similar to those of *midp*, but it supports the use of device skins together with a configuration file, so different devices can be emulated without the need to modify any code. Although the emulator can be used from the command line, it is most frequently accessed indirectly via the KToolBar interface provided by the Wireless Toolkit.

2.1.9.1 Options

The operation of the *emulator* command is determined by the options supplied to it. There are three different modes of operation:

- Displaying information using the *-help*, *-version*, and *-Xquery* options. Here, the *classname* argument is not required, and the command exits after printing the required information.
- Running a MIDlet from the local system or by loading from a network server, but without installing it. This mode of operation uses the *-classpath* option together with a class name or the *-Xdescriptor* option, which may or may not be accompanied by a class name.
- Using the emulator's application management software to install, run, list, or delete MIDlet suites. This mode of operation uses the *-Xjam* option.

2.2 Socket Programming

Sockets are the lowest level of network communication that most programmers encounter, although real enthusiasts might choose to delve into the murky details of transport and network layers -- and some even survive the experience! Because the socket API is so simple, widely known, and universally available, it is often used as the basis for distributed applications involving one or more clients talking to a single server, exchanging information using a very basic application-level protocol. In this situation, the use of a higher-level abstract such as RMI, CORBA, or one of the Java Enterprise products would not be justified. All this notwithstanding, CLDC does not require the provision of a socket interface to the network, and neither does MIDP. Part of the reason is that sockets are usually used in connection with Internet protocols such as TCP/IP, but many mobile devices do not have a direct connection to the Internet, and, therefore, the device's host software almost certainly does not include a TCP/IP protocol stack. Making sockets part of MIDP would have required manufacturers to add this software to their devices (which has an associated cost) or necessitated its inclusion in the MIDP reference implementation, which is not economically possible on many platforms because of the memory requirements.

At the present time, therefore, applications that use sockets work on some devices, such as PDAs with modems, but not on others and thus cannot be considered portable. However, because sockets are likely to be supported in the next version of MIDP, we'll take advantage of the socket implementation in the CLDC 1.0 reference release to illustrate how sockets fit into the GCF by showing a simple application that retrieves some data from a web server.

2.2.1 Client Sockets

The steps required to open a socket connection to a web server and read some data from it are as follows:

1. Build the appropriate name string and invoke the Connector open( ) method.
2. Get an output stream and use it to send a request message to the server.
3. Open an input stream and read the response.
4. Close both streams and the socket.

The naming scheme for sockets uses the fixed string "socket://" followed by the server name and port, separated by a colon. Here's how you might open a socket to a web server given the server's name and a string containing its port number (usually 80) in variables called server and port, respectively:

```
StreamConnection socket;
try {
String name = "socket://" + server + ":" + port;
socket = (StreamConnection)Connector.open(name,
Connector.READ_WRITE);
} catch (Exception ex) {
// Handle failure to connect here...
}
```

2.2.2 Server Sockets

The programming model for server sockets differs in several ways from that of client sockets. First, the name that you give to the Connector open( ) method contains the port that you want the server to listen on, but it does not specify the hostname. A server implicitly listens on the host it is running on, so there is no need to give a hostname; the protocol implementation uses this fact to distinguish a request to create a server socket from a request for a client socket. To listen on port 80, for example, you would use the following name:

socket://:80

The biggest difference with server sockets is that the Connector open( ) method doesn't return a StreamConnection object that you can use to send and receive data. This is because a server differs from a client in two important ways:

• When a server is started, it isn't connected to a client at all. Instead, it needs to register a port to listen on and then wait for a client to connect to that port.

• In general, a server supports many clients, either one after another or in parallel.Therefore, it needs several different sockets, one for each client that it communicates with.

Chapter 3

Project Planning

3.1. UML Diagrams

The Unified Modeling Language (UML) is a language that specifies, visualizies, constructs, and documents of software systems, for the purpose of business modeling and other non-software systems. The best engineering practices that have seen success in the field modeling of large and complex systems have been presented by the UML. The UML is responsible for developing object oriented software as well as the software development process. The UML makes use of the graphical notations to express and depict the design and structure of various software projects.

Goals of UML

The basic goals in the design of the UML are:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

3.1.1. Use Case Diagram

Use case diagrams are important for visualizing, specifying, and documenting the behavior of an element.

- *A diagram that depicts a set of use cases by defining the actors involved and the* relationships that exist between them is called a Use Case diagram.
- Use case diagrams commonly contain
  - Use cases
  - Actors
  - Dependency, generalization, and association relationships

Description of specific Use Case Diagram:

- **Actors** are *CLIENT and SERVER.*
- **Use cases** are *Connect, Enter IP Address, Enter Port No., Screen Sharing, Mouse, Click , Keyboard, Send Message.*
- **Dependency or Relationship** is *<<include>>.*

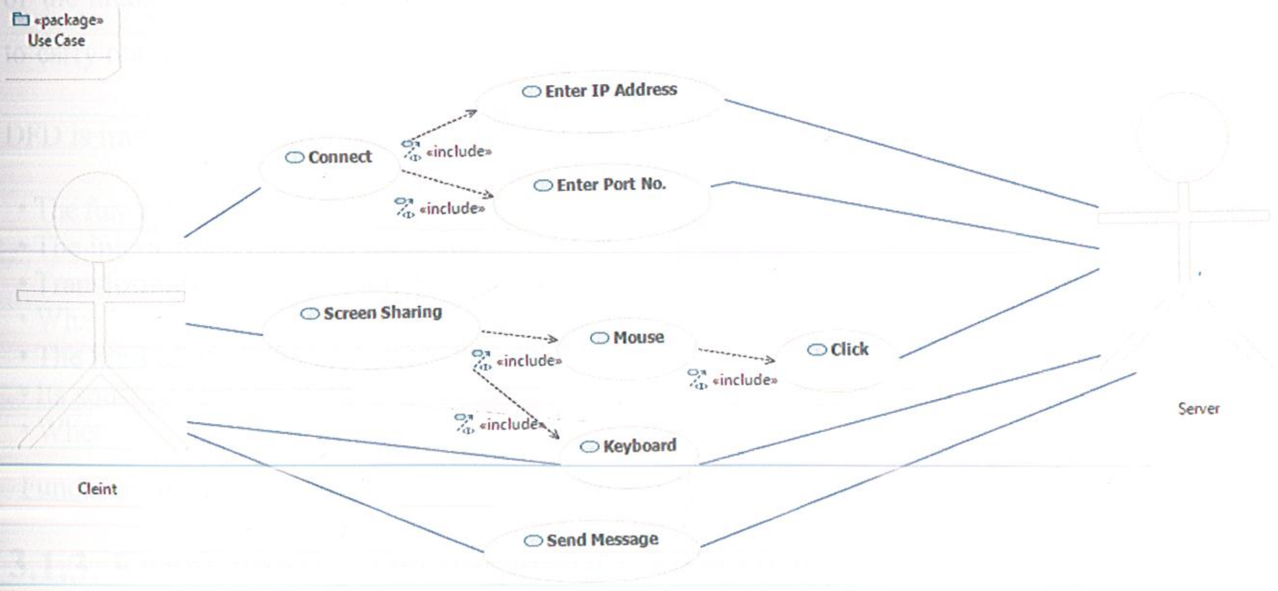


Fig 11: Use Case Diagram

3.1.2. Data Flow Diagram

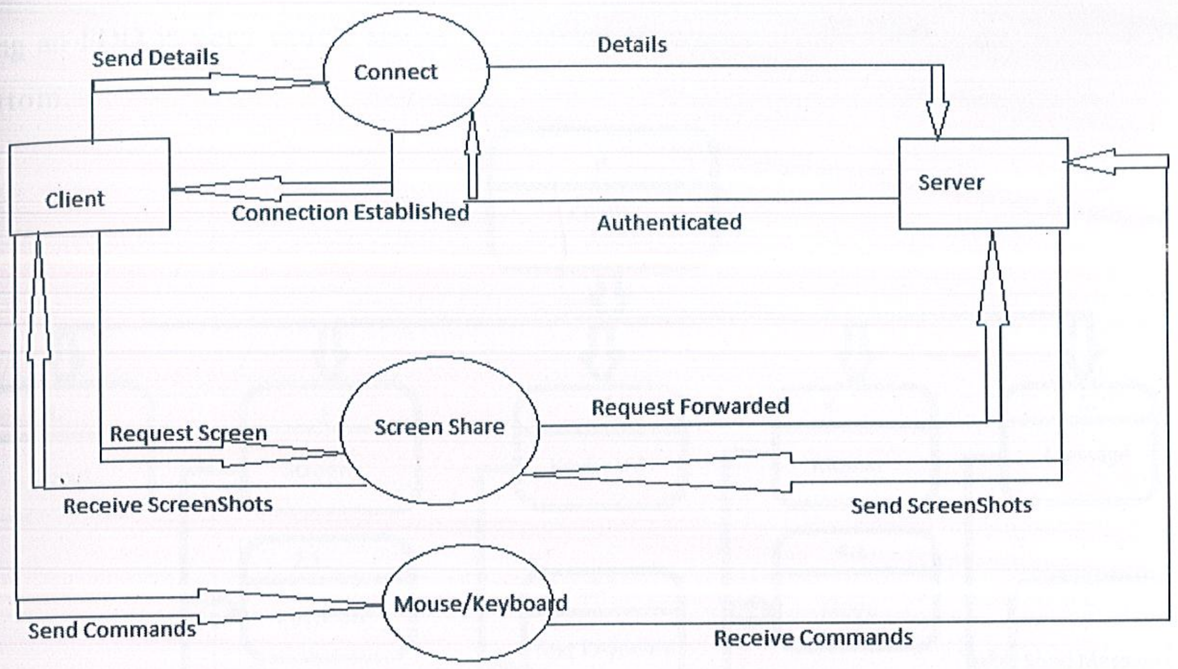


Fig 12: Data Flow Diagram

Data flow diagrams provide a distinct and not vague representation of any business function that exists. It involves an overall picture of the business and is followed by keenly analysing each of the areas of interest that are functional. It easily exploits a method called top-down expansion to carry out its analysis as it desires.

DFD is important to know to get the knowledge of :

- The functions that the system perform
- The interactions between these functions
- Transformations carried out by the system
- What inputs are transformed into what outputs?
- The kind of work the system is expected to do
- Its source of information
- Where does it deliver its results

Functions in the system are *Connect, Screen Share, Mouse /Keyboard.*

3.1.3. Functionality Decomposition Diagram

Functionality Decomposition Diagram presents a top-down representation of a function or even a process.
These are also known as the structure charts.

System analysts use FDD's for the purpose of modelling business functions and depict their organization into lower-level processes. Those processes convert themselves into program modules during the phase of application development.

Creating an FDD is very much similar to drawing an organizational chart – go from to top to bottom.

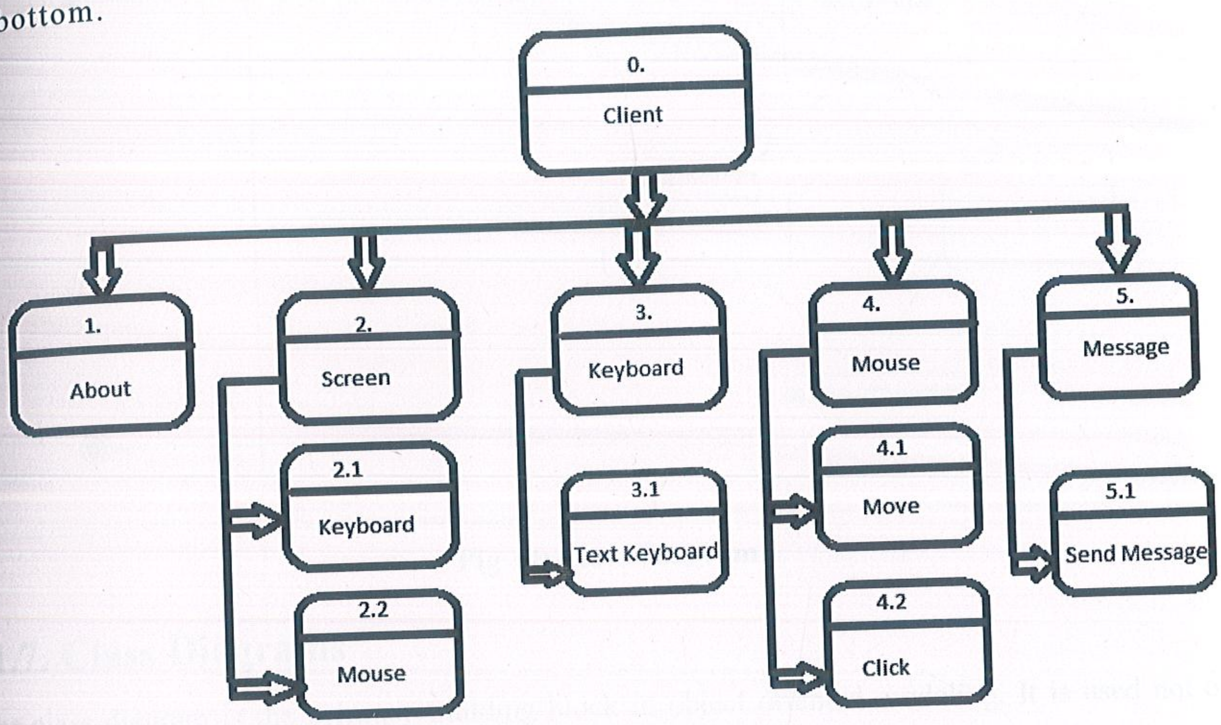


Fig 13:Functionality Decomposition Diagram

3.1.6. State Diagram

A diagram that is used in the field of computer science and related fields to describe the behavior of systems is called a state diagram. In this type of diagram the system so described is comprises of a finite number of states; sometimes, this is indeed the case, while at other times this is an abstraction. Many forms of state diagrams exist, that differ slightly from each other and have different semantics.

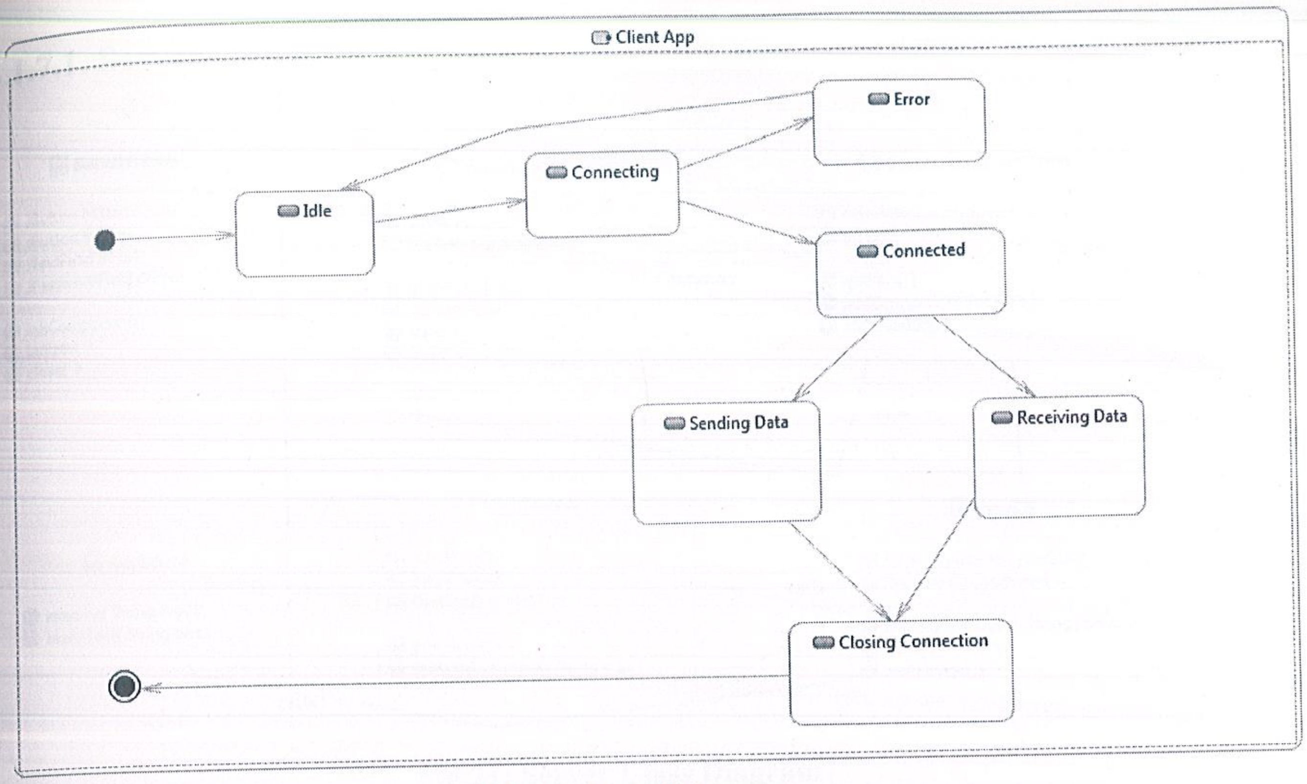


Fig 14: State Diagram

3.1.7. Class Diagrams

The class diagram is the primary building block in object oriented modeling. It is used not only for the building a general conceptual model of the application, but also for the detailed modeling which involves translation of the models into somewhat of a programming code. The classes that are present in a class diagram represent both the main objects and interactions that exist in the application and the objects that have to be programmed. The class diagram represents these classes in three parts:

A class with three sections.

- The name of the class is held in the upper part
- The attributes of the class is stored in the middle part
- The`bottom part is responsible for providing the methods a particular class can take

In the system design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modeling, the classes of the conceptual design are often split into a number of subclasses.

3.1.7.1. Server

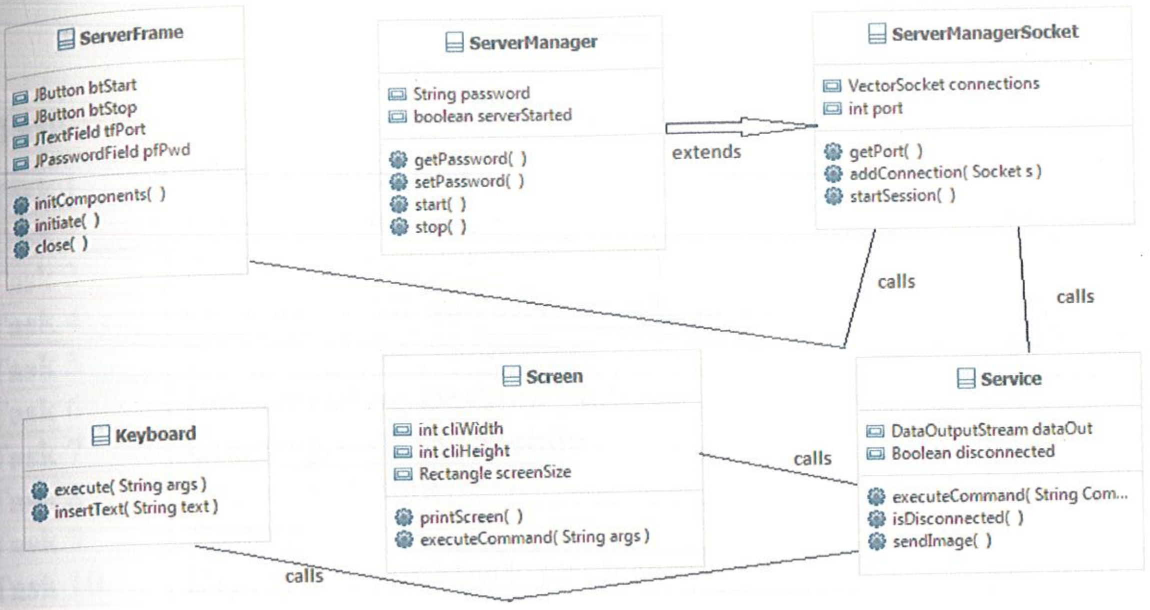


Fig 21: Server Class Diagram

3.1.7.2. Client

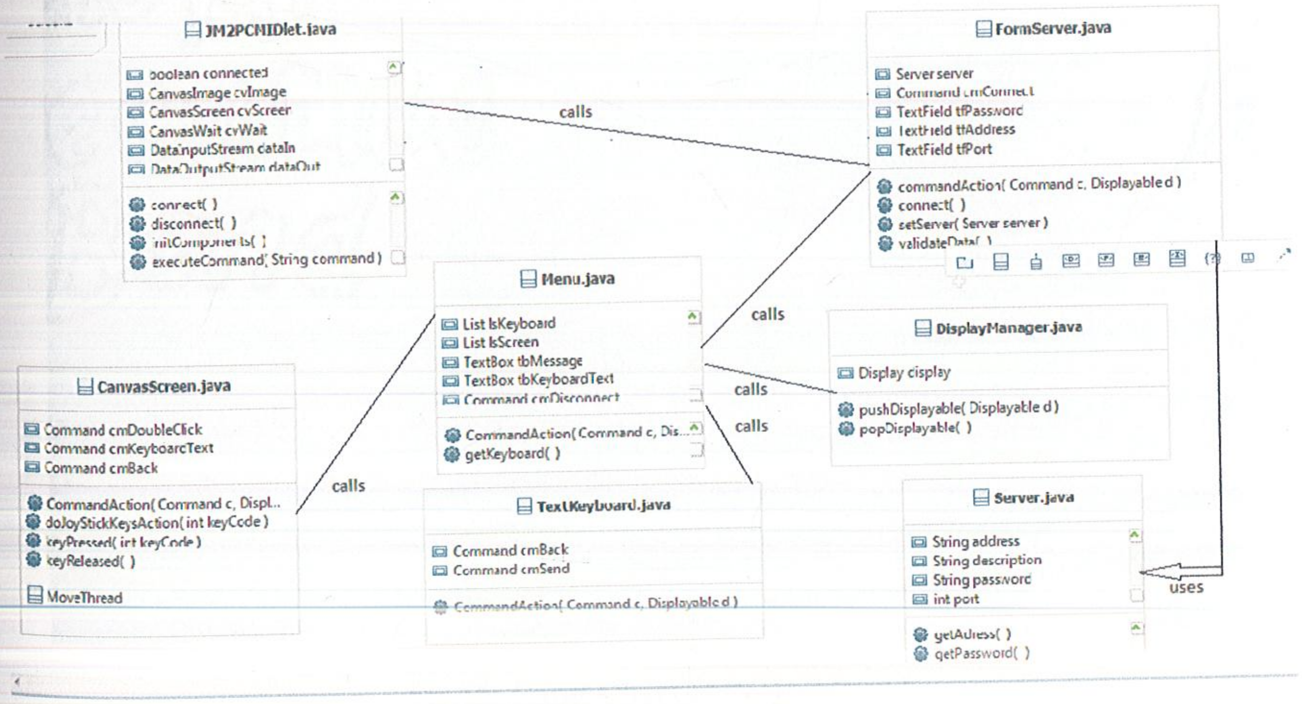


Fig 22: Client Class Diagram

3.2. Plan

Table 1

Task No.	Description	Target Date
Task 1	Topic Finalization	27th July 2011
Task 2	Literature Survey	17th August 2011
Task 3	Review of Design	30th August 2011
Task 4	Learning RMI and Networking in Java	20th September 2011
Task 5	Designing and Analyzing	10th October 2011
Task 6	Implementing Desktop Sharing	20th November 2011
Task 7	Creating GUI for Mobile	15th February 2012
Task 8	Implementation	25th March 2012
Task 9	Testing	5th April 2012
Task 10	Deployment in J2ME Environment	1st May 2012
Task 11	Project Documentation	18th May 2012

3.3. Gantt Chart

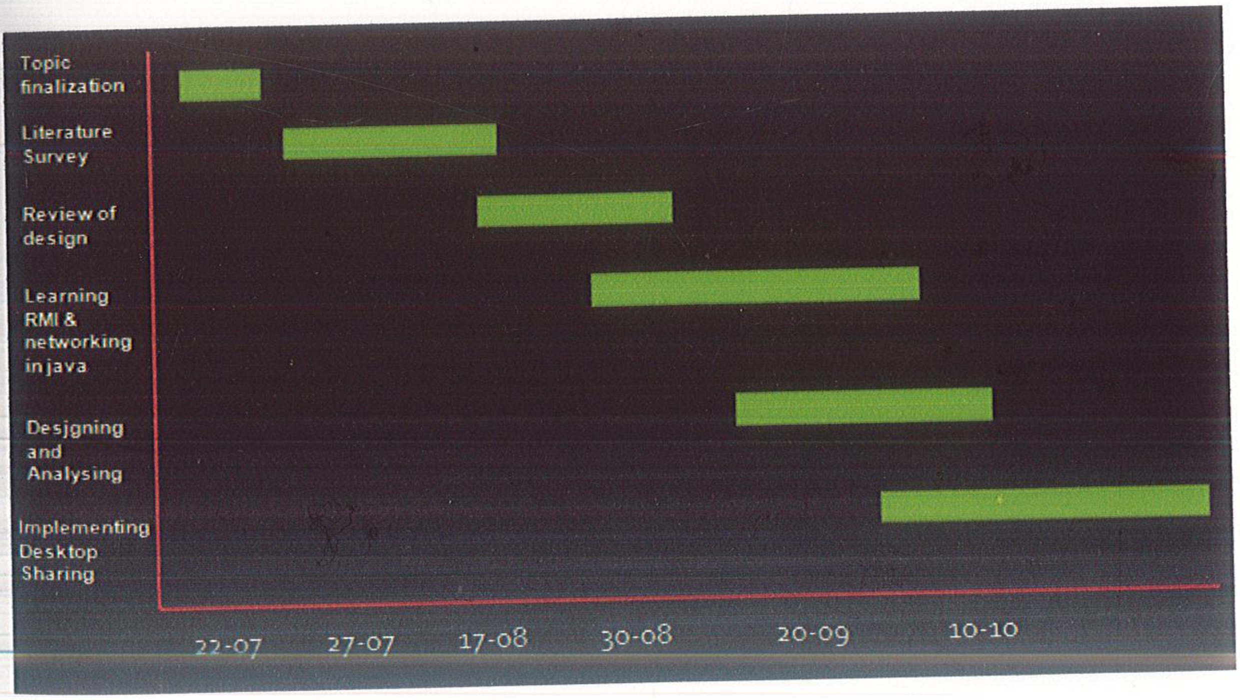


Fig 15: Gantt Chart of Project Part 1

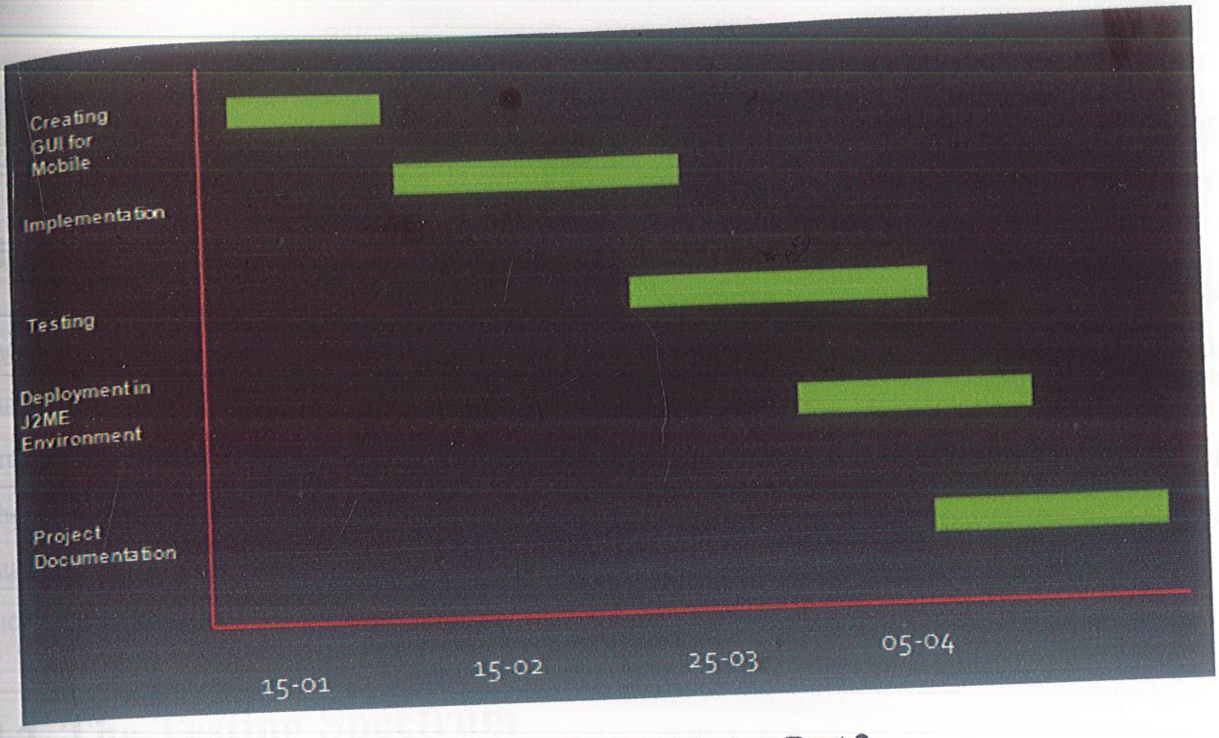


Fig 16: Gantt Chart of Project Part 2

Chapter 4
Testing

4.1. The Goal of Testing

The description of testing as given by Miller : The goal of testing is to test the software systems quality by systematically exercising and examining the software in carefully controlled circumstances.

The most important aspect of testing is "Performance Testing". The reason for this is when multiple users simultaneously use the application accessing the database at the same time at such frequent high rates it is crucial that performance should not deteriorate.

4.2. The Testing Spectrum

Testing is an integral part in every possible stage of the software, however the testing involved at each level of software development is different in nature and objective from the testing involved at another layer.

4.3. Unit Testing

Unit testing is done at the lowest possible level. What id does is, it tests the basic unit of software. The basic unit of a software is the smallest testable piece of software. It is usually called unit, module, or component interchangeably.

4.4. Integration Testing

Integration Testing is performed when two or more thn two tested units are brought together into a larger structure. This testing is performed on both the interfaces between the components and the larger structure being constructed.

4.5. System Testing

System testing assures the end-to-end quality of the entire system. It is performed keepin in mind the functional/requirement specification of the system. It also involves checking of the Non-functional quality attributes, such as reliability, security, and maintainability.

4.6. Acceptance Testing

Acceptance testing is done when the entire system after its development phase is handed over from the developers to the customers or users alike. Rather than finding errors the basic aim of acceptance testing is to give confidence that the system is working.

All the aforementioned techniques will be implemented by us.

4.7. Performance Testing

For the purpose of performance tesing we use special tools that are available which enable us to performance test any application with n number of users.

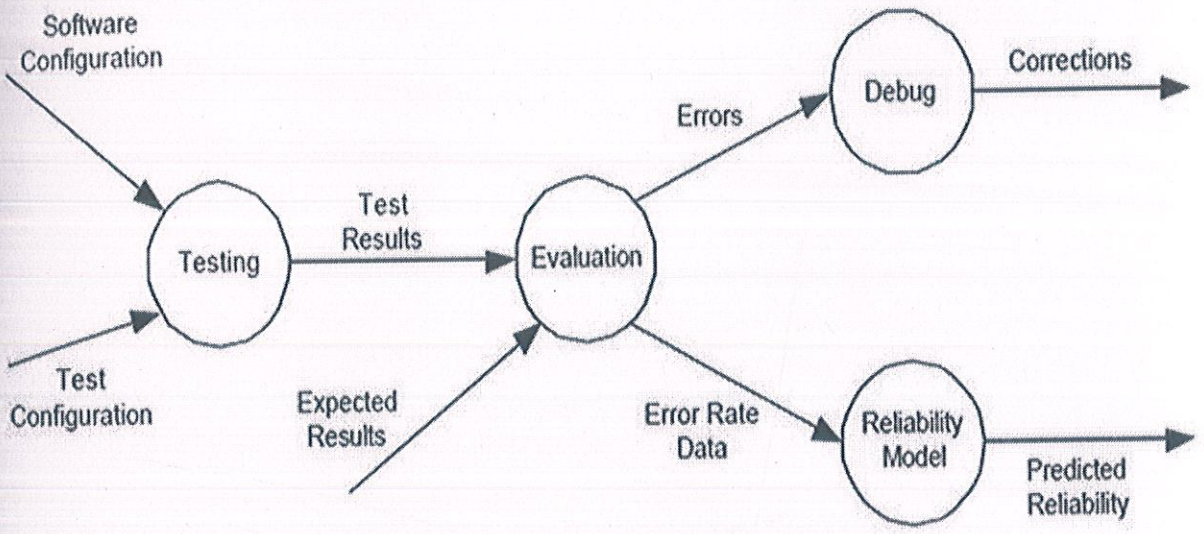


Figure 21: Testing Information Flow

4.8. Test Cases

4.8.1. Unit Test Cases

Table 2

S No.	Test Case Name	Test Case Description	Expected output	Inference
1.	Make Connection	Enter all the details on the connect page and press connect button.	When all fields are entered and connect button is pressed Menu Page should appear.	Passed
2.	Screen Share	Click the screen option on the menu page.	Desktop screen should appear on mobile.	Passed
3.	Send Message	Click the send option on menu and type text and press send button.	Message should be sent to the server.	Passed

4.8.2. Acceptance Testing

We can fully access the static computer from anywhere using an internet connection and fully control it and hence we have accomplished all the aforementioned requirements.

4.8.3. Performance Testing

There are few performance barriers in our project on which we are still working .

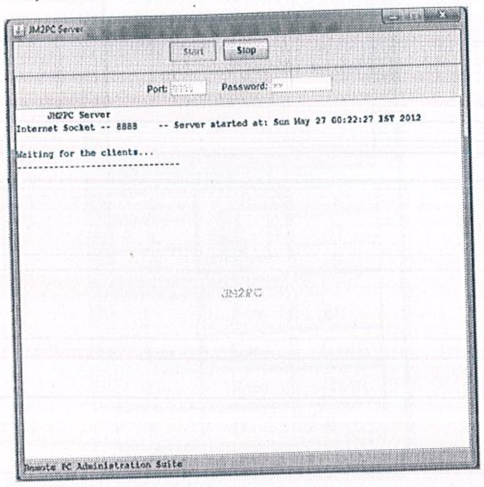
- Delay in Screen Transfering
- Less accuracy in position of the mouse

Chapter 5

Results and Conclusion

5.1 Results

The Server runs on the Static IP PC, and waits for the clients.

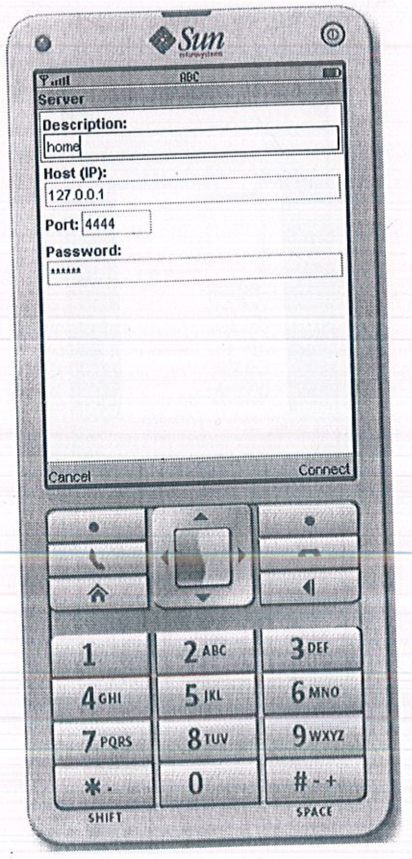


```
    JM2PC Server
Internet Socket -- 8888     -- Server started at: Sun May 27 00:22:27 IST 2012

Waiting for the clients...
------------------------------
```

Fig 23: Server Waiting for Clients

The Client runs on the mobile device, it requests the user to enter IP address of the server, the port number on which the server is running and other details.



Description: home
Host (IP): 127.0.0.1
Port: 4444
Password: ******

Fig 24: Client requesting for details

The client now connects to the server and waits for the user to perform any action for it to execute.

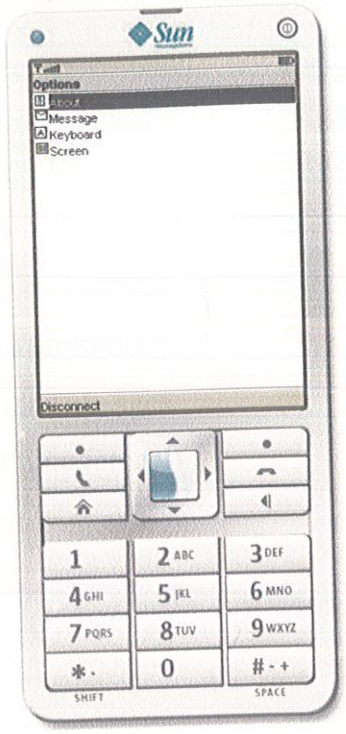


Fig 25: Client connected and waiting for commands

When the Client chooses to access the screen of the server, the client requests the server to send the screen and displays it to the user.

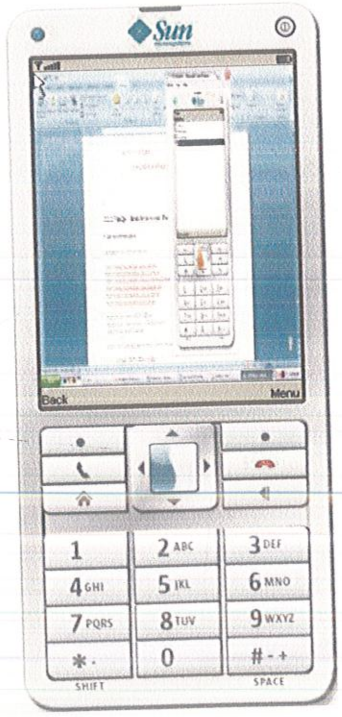


Fig 26: Server Screen shared

5.2. Conclusion

We present a Remote PC Administration Suite for Mobile Devices that can serve as a basic for accessing any Remote PC using Internet Connection. A 2-tier architecture is used comprising of the interface, application server and the client. We have developed an application which has following advantage –

- Faster, decentralized decision making.
- Increased responsiveness to customers.
- Increased sensitivity to market changes.
- Lowered commuting costs/time for staff.
- Increased productivity.

References:

Books:

- Kathy Sierra & Bert Bates, *Head First Java,* O'Reilly, 2005
- William Grosso, *Java RMI,* O'Reilly Media, 2001
- Dr. Yu Feng, *Wireless Java Programming with J2ME*, Sam's Publishers, 2001
- Herbert Schidt, *JAVA 2-The Compete Reference*, McGraw Hill, 5th ed, 2002
- James Keogh, *J2ME-The Complete Reference*, McGraw Hill,2003
- Elliotte Rusty Harold, *Java Network Programming*, O' Reilly Media, 3rd Edition, 2004
- Kim Topley, J2ME in a Nutshell, O'Reilly, 2002

World Wide Web:

- http://www.tutorialspoint.com/java/java_networking.htm
- http://www.roseindia.net/j2me/
- http://developers.sun.com/mobility/midp/articles/wtoolkit/
- http://docs.oracle.com/javase/6/docs/api/
- http://docs.oracle.com/javame/

Journal Article:

- Mr. S. Kulkarni , Miss S. Diwan Prof. N.K. Bansode, "Device Independent Mobile Application Controller For Remote Administration Of A Server Over A GPRS Link Using a J2ME Cellular Phone**"**, *Department of Computer Engineering, Army Institute of Technology, Dighi Hills, Pune*, pp 6.
- Phillip Pressley,"Integrating new PC Remote Access, Remote Desktop Software and Remote Access Software technologies into business"
- "J2ME Building Blocks for mobile Devices" White Paper by Sun Microsystems java.sun.com/products/cldc/wp/KVMwp.pdf
- Andre N. Klingsheim, Vebjorn Moen and Kjell J. Hole, "Challenges in securing networked J2ME applications", *University of Bergen*, pp. 30.

Appendix A

Server

jm2pc.server.gui Package

ServerFrame.java

```
package jm2pc.server.gui;

import java.awt.*;
import java.awt.event.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Properties;

import javax.comm.*;
import javax.swing.*;
import javax.swing.border.*;

import jm2pc.server.connection.ServerManager;
import jm2pc.server.connection.internet.ServerManagerSocket;
import jm2pc.server.i18n.Messages;
import jm2pc.server.log.Log;

public class ServerFrame extends JFrame {

        public static final long serialVersionUID = 1l;
        public static final String PROPS_KEY_AUTHENTICATE = "authenticate";
        public static final String PROPS_KEY_AUTHORIZE = "authorize";
        public static final String PROPS_KEY_ENCRYPT = "encrypt";
        public static final String PROPS_KEY_BAUD = "baud_rate";
        public static final String PROPS_KEY_BTYPE = "bluetooth_type";
        public static final String PROPS_KEY_COMM = "serial_port";
        public static final String PROPS_KEY_PORT = "port";
        public static final String PROPS_KEY_PASSWORD = "password";
        public static final String PROPS_KEY_LANGUAGE = "language";
        public static final String PROPERTIES_FILE = "jm2pc_server.properties";

        public static final String COMM_PROPERTIES_FILE =
"lib/comm/javax.comm.properties";

        private HashMap<String, CommPortIdentifier> mapComm;

        private JButton btStart;

        private JButton btStop;
        private JLabel lbPort;
private JLabel lbPassword;
private JPanel panelControl;
private JPanel panelButtons;
private JPanel panelOptions;
private JPanel panelOptionsType;
private JPasswordField pfPassword;
private JScrollPane scLog;
private LogArea taLog;
private JTextField tfPort;
private JComboBox cbCommPort;
private JComboBox cbBaudrate;
private JLabel lbBps;
private JCheckBox chkAuthenticate;
private JCheckBox chkAuthorize;
private JCheckBox chkEncrypt;
private LogoWindow logoWindow;
private JTextArea taCredits;
private ServerManager gerServer;
private Log log;
private Properties props;
private String PortCommDefault;

public ServerFrame() {
        super("JM2PC Server");
        props = new Properties();
        PortCommDefault = null;

        initComponents();
        pack();
}
private void initComponents() {
        panelControl = new JPanel(new GridLayout(2, 1));
        panelOptions = new JPanel();
        panelButtons = new JPanel();
        panelOptionsType = new JPanel();

        btStart = new JButton(Messages.getMessage("start"));
        btStart.setToolTipText(Messages.getMessage("helpStart"));
        btStart.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent event) {
                        initiate();
                }
        });

        btStop = new JButton(Messages.getMessage("stop"));
        btStop.setToolTipText(Messages.getMessage("helpStop"));
        btStop.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent event) {
                        stop();
                }
        });

        lbPort = new JLabel();
        tfPort = new JTextField();
        lbPassword = new JLabel();
        pfPassword = new JPasswordField();
        lbBps = new JLabel("Bps:");
        cbCommPort = new JComboBox();
        cbCommPort.setToolTipText(Messages.getMessage("helpCommPort"));
        cbBaudrate = new JComboBox();
        populateBoundRate();
        cbBaudrate.setToolTipText(Messages.getMessage("helpBps"));
        chkAuthenticate = new JCheckBox("Authenticate");
        chkAuthenticate

.setToolTipText(Messages.getMessage("helpAuthenticate"));
        chkAuthorize = new JCheckBox("Authorize");
        chkAuthorize.setToolTipText(Messages.getMessage("helpAuthorize"));

        chkEncrypt = new JCheckBox("Encrypt");
        chkEncrypt.setToolTipText(Messages.getMessage("helpEncrypt"));
        logoWindow = new LogoWindow(this);
        Font font = new Font("Monospaced", Font.BOLD, 12);
        taLog = new LogArea(font, logoWindow);
        scLog = new JScrollPane(taLog);

        taCredits = new JTextArea(1, LogArea.COLS);
        taCredits.setBackground(Color.LIGHT_GRAY);
        taCredits.setForeground(Color.BLACK);
        taCredits.setFont(font);
        taCredits
                .setText("Remote PC Administration Suite");
        taCredits.setEditable(false);

        btStop.setEnabled(false);

        panelButtons.add(btStart);
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelButtons.add(btStop);
        lbPort.setText(Messages.getMessage("port") + ":");
        panelOptionsType.add(lbPort);
        tfPort.setColumns(4);
        tfPort.setText("8888");
        tfPort.setToolTipText(Messages.getMessage("helpPort"));
        panelOptionsType.add(tfPort);
        panelOptionsType.add(new JSeparator());
        panelOptions.add(panelOptionsType);
        panelOptions.add(new JSeparator());
        lbPassword.setText(Messages.getMessage("password") + ":");
        panelOptions.add(lbPassword);
        pfPassword.setColumns(7);
        pfPassword.setToolTipText(Messages.getMessage("helpPassword"));
        panelOptions.add(pfPassword);
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));


        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelButtons.add(new JSeparator(JSeparator.HORIZONTAL));
        panelOptions.setBorder(new BevelBorder(BevelBorder.LOWERED));
        panelControl.add(panelButtons);
        panelControl.add(panelOptions);
        getContentPane().add(panelControl, BorderLayout.NORTH);
        getContentPane().add(scLog, BorderLayout.CENTER);
        getContentPane().add(taCredits, BorderLayout.SOUTH);
        pack();
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        WindowListener windowListener = new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                close();
            }

            public void windowIconified(WindowEvent arg0) {
                hides();
            }

        };

        addWindowListener(windowListener);
    }

    protected void setPassword(String Password) {
        pfPassword.setText(Password);
    }


    public void initiate() {

        String Password = new String(pfPassword.getPassword());
        if (Password.length() == 0) {
            JOptionPane.showMessageDialog(this, Messages
                    .getMessage("errorNoPassword"), Messages
                    .getMessage("error"), JOptionPane.ERROR_MESSAGE);
            pfPassword.requestFocus();
            return;
        }

        try {
            gerServer = initiateSocket();
            gerServer.setPassword(Password);
            gerServer.start();
            btStart.setEnabled(false);
            btStop.setEnabled(true);
            setEditFields(false);
            logoWindow.setServerStart(true);
        } catch (Exception e) {
            StringBuffer sbMsg = new StringBuffer(Messages
                    .getMessage("errorStartServer")
                    + ": \n\t");
            String msg = e.getMessage();
            System.out.println(e.toString());
            if (msg != null)
                sbMsg.append(msg);
            taLog.logError(sbMsg.toString());

        }


}


private ServerManager initiateSocket() {
        int Port;
        try {
            Port = getPort();
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, Messages
                    .getMessage("errorPortNumber"), Messages
                    .getMessage("error"), JOptionPane.ERROR_MESSAGE);

            tfPort.requestFocus();
            return null;
        }

        ServerManagerSocket gerServerSocket;

            gerServerSocket = new ServerManagerSocket(taLog);


        gerServerSocket.setPort(Port);

        return gerServerSocket;

    }

    public int getPort() throws NumberFormatException {

        int Port = Integer.parseInt(tfPort.getText());
        if (Port < 0 || Port > 65535)
            throw new NumberFormatException();

        return Port;

    }

    private void setEditFields(boolean enabled) {
        tfPort.setEnabled(enabled);

        pfPassword.setEnabled(enabled);
        cbBaudrate.setEnabled(enabled);
        cbCommPort.setEnabled(enabled);

    }

    public void stop() {
        String[] Options = { Messages.getMessage("yes"),
                        Messages.getMessage("no") };
        if (JOptionPane.showOptionDialog(this, Messages
                        .getMessage("stopServerConfirm"), Messages
                        .getMessage("confirm"), JOptionPane.YES_NO_OPTION,
                        JOptionPane.QUESTION_MESSAGE, null, Options, Options[1])
== JOptionPane.YES_OPTION) {

                try {
                        gerServer.stop();
                        btStop.setEnabled(false);
                        btStart.setEnabled(true);
                        setEditFields(true);
                        logoWindow.setServerStart(false);
                        if (log != null) {
                                log.close();
                                taLog.setLogFile(null);
                        }
                } catch (Exception e) {
                        taLog.logError(Messages.getMessage("errorStopServer")
                                        + ": \n\t" + e.getMessage());

                }

        }

    }

    protected void hides() {

            logoWindow.setVisible(true);
            this.setVisible(true);

    }

    public void close() {

            String[] Options = { Messages.getMessage("yes"),
                            Messages.getMessage("no") };
            if (JOptionPane.showOptionDialog(this, Messages
                            .getMessage("exitConfirm"),
Messages.getMessage("confirm"),
                            JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE,
null,
                            Options, Options[1]) == JOptionPane.YES_OPTION) {
                    if (log != null) {
                            log.logSucess(new java.util.Date() + ": "
                                            + Messages.getMessage("exitLog"));
                            log.close();
                    }
                    try {
                            //saveProperties();
                    } catch (Exception e) {
                            e.printStackTrace();
                    } finally {
                            System.exit(0);
                    }
            }
    }

    protected LogoWindow getLogoWindow() {
          return logoWindow;
    }

    public static void main(String args[]) {
          ServerFrame frame = new ServerFrame();
                frame.setPreferredSize(new Dimension(600,600));
             frame.pack();
          frame.setVisible(true);
    }
}
```

jm2pc.server.connection.internet Package

ServerManagerSocket.java

```
package jm2pc.server.connection.internet;

import java.io.*;
import java.net.*;
import java.util.Date;

import jm2pc.server.connection.ClientInfo;
import jm2pc.server.connection.SessionManager;
import jm2pc.server.i18n.Messages;
import jm2pc.server.log.Loggable;
import jm2pc.server.service.Service;
import jm2pc.server.utils.DateFormat;

public class SessionManagerSocket extends SessionManager {

   private Loggable log;
   private Socket socket;
   private BufferedReader inReader;
   private InputStream in;
   private OutputStream out;
   private ClientInfo clientInfo;
   private int clientNumber;
   private Service service;

   public SessionManagerSocket(Socket socket, Loggable log, int clientNumber,
String password){
        super(password);
          this.socket = socket;
        this.log = log;
        this.clientNumber = clientNumber;

   }

   public void run() {

        String dscClient = " Socket Client ";
```

```
    try {
        in = socket.getInputStream();
        inReader = new BufferedReader(new InputStreamReader(in));
        out = socket.getOutputStream();

        setDataIn(new DataInputStream(in));
        setDataOut(new DataOutputStream(out));
      StringBuffer sb = new StringBuffer((Messages.getMessage("client") + " " +
clientNumber + " = " + socket.getInetAddress().getHostAddress()));
        sb.append('\n');
        sb.append(DateFormat.format(new Date()) + " -> " +
Messages.getMessage("client") + " -- " + clientNumber + " -- " +
Messages.getMessage("connected"));

        log.logSucess(sb.toString());
        if(login()) {
                log.logSucess(DateFormat.format(new Date()) + " -> " +
Messages.getMessage("client") + " -- " + clientNumber + " -- " +
Messages.getMessage("authenticationOK"));

                clientInfo = receiveClientInfo();
                sendServerInfo();
                sb = new StringBuffer(Messages.getMessage("client") + " -- ");
                sb.append(clientNumber);
                sb.append(" [");
                sb.append(clientInfo.getPlataform());
                sb.append("] -- ");

                dscClient = sb.toString();

                service = new Service(log, dscClient, clientInfo, out);

                log.logSucess(DateFormat.format(new Date()) + " -> " + dscClient
+ " " + Messages.getMessage("ready"));
                String command;
                while(!(service.isDisconnected()) && (command =
inReader.readLine()) != null) {
                        service.executeCommand(command);
                }
        } else {
                log.logError(DateFormat.format(new Date()) + " -> " +
Messages.getMessage("client") + " -- " + clientNumber + " -- " +
Messages.getMessage("accessDenied"));
        }
    } catch(SocketTimeoutException timeoutException) {
        log.logError(DateFormat.format(new Date()) + " -> " + dscClient + " " +
Messages.getMessage("closeConnection") + " -- " +
Messages.getMessage("timeoutMessage"));
    } catch (SocketException socketException){
        log.logError(clientNumber + " - " +
Messages.getMessage("socketDisconnect"));
    } catch(IOException e) {
        log.logError(clientNumber + " - " + Messages.getMessage("error") + ": "
+ e.toString());
    } finally {
        log.logSucess(DateFormat.format(new Date()) + " -> " + dscClient + " "
+ Messages.getMessage("closeConnection"));
        try {
            socket.close();
        } catch(IOException ioe) { }
    }

    log.logError(clientNumber + " " + Messages.getMessage("leaving") +
"...");
  }}
```

jm2pc.server.service Package

Service.java

```
    import java.awt.Image;
    import java.io.*;
    import java.util.ArrayList;
    import java.util.Date;
    import java.util.Hashtable;

    import com.jm2pc.Command;

    import jm2pc.server.connection.ClientInfo;
    import jm2pc.server.i18n.Messages;
    import jm2pc.server.log.Loggable;
    import jm2pc.server.utils.DateFormat;
    import jm2pc.utils.Constants;

    public class Service {

        private Loggable log;

        private boolean disconnected;
        private String dscClient;
        private Hashtable<String,Command> commands;
        private Hashtable<String, Command> plugins;

        private DataOutputStream dataOut;
        private ClientInfo clienteInfo;

        private DataFormat dataFormat;

        public Service(Loggable log, String dscClient, ClientInfo clientInfo,
OutputStream out){

            this.log = log;
            this.dscClient = dscClient;

            disconnected = false;

            dataOut = new DataOutputStream(out);
            this.clienteInfo = clientInfo;

            commands = new Hashtable<String, Command>();
        plugins = new Hashtable<String, Command>();

        dataFormat = new DataFormat(clientInfo, out);

        Control control = Control.getInstance();

        addCommand(new Mouse(log, dscClient, control));
        addCommand(new Message(log, dscClient));
        addCommand(new Keyboard(log, dscClient, control));
        addCommand(new Screen(log, dscClient, control, dataFormat, out));
    }

    public void addCommand(Command cmd) {
            commands.put(cmd.getName(), cmd);
    }

    public boolean isDisconnected() {
            return disconnected;
    }

    public void sendString(String str) throws IOException {
        dataFormat.formatAndSend(str);
    }

    public void sendImage(Image im) {

        Image scaledImg = im.getScaledInstance(clienteInfo.getWidth(),
clienteInfo.getHeight(), Image.SCALE_AREA_AVERAGING);
        int tam = dataFormat.formatAndSend(scaledImg);
        log.logSucess(DateFormat.format(new Date()) + " -> " + dscClient + "
[Plug-In] " + Messages.getMessage("capturingScreen") + "... " + tam + " bytes");
    }

    public void executeCommand(String command) {

        if(command == null || command.equals(Constants.CMD_SAIR)) {
            disconnected = true;
        }
        else {
          try {

            int i = command.indexOf(" ");
                String commandName = command.substring(0, i);
                String args = command.substring(i + 1, command.length());

                Command cmd;
                cmd = commands.get(commandName);
                if(cmd != null) {
                  cmd.execute(args);
                } else {
                  cmd = plugins.get(commandName);
                  Object returnType = cmd.execute(args);
                  if(returnType == null) {
                        dataOut.writeInt(Constants.Plugin_TP_VOID);
                        }
                        else {
                              if(returnType instanceof Image) {

                                    dataOut.writeInt(Constants.Plugin_TP_IMAGE);
                                    Image im = (Image) returnType;
                                    sendImage(im);
                              } else if(returnType instanceof String) {

                                    dataOut.writeInt(Constants.Plugin_TP_TEXTO);
                                    String str = (String) returnType;

                                    sendString(str);
                              } else {
                                    dataOut.writeInt(Constants.Plugin_TP_VOID);
                                                            }

                        }
                        dataOut.flush();

                        log.logSucess(DateFormat.format(new Date()) + " -> " +
dscClient + " -- Plug-In -- " + command);
                  }

            } catch(Exception e) {
                  log.logError(DateFormat.format(new Date()) + " -> SERVER :
" + dscClient + " " + Messages.getMessage("error") + ": " + e.toString() + "\t"
+ e.getMessage());
            }
      }
   }
}
```

Screen.java

```
package jm2pc.server.service;

import java.awt.Image;
import java.awt.image.BufferedImage;
import java.awt.Toolkit;
import java.awt.Rectangle;
import java.io.OutputStream;
import java.util.Date;

import com.jm2pc.Command;
import jm2pc.server.i18n.Messages;
import jm2pc.server.log.Loggable;
import jm2pc.server.utils.DateFormat;
import jm2pc.utils.Constants;

public class Screen implements Command {

      private Control control;
      private Rectangle screenSize;
```

```
    private int cliWidth;
    private int cliHeight;

    private Loggable log;
    private String dscClient;

    private DataFormat dataFormat;

    public Screen(Loggable log, String dscClient, Control control, DataFormat
dataFormat, OutputStream out) {
            this.log = log;
            this.dscClient = dscClient;
            this.control = control;
            this.dataFormat = dataFormat;

            screenSize = new
Rectangle(Toolkit.getDefaultToolkit().getScreenSize());

            cliWidth = dataFormat.getClienteWidth();
            cliHeight = dataFormat.getClienteHeight();

    }

    public Object execute(String args) throws Exception {
            String[] param = args.split(" ");
            String type = param[0];

            int tam = 0;

            if(type.equals(Constants.CMD_TELA_TP_INT)) {
                    tam = printScreen();
            } else if(type.equals(Constants.CMD_TELA_TP_XY)) {
                    int x = Integer.parseInt(param[1]);
                    int y = Integer.parseInt(param[2]);
                    int zoom = Integer.parseInt(param[3]);

                    tam = printScreen(x, y, zoom);
            }

            log.logSucess(DateFormat.format(new Date()) + " -> " + dscClient + "
" + Messages.getMessage("capturingScreen") + "... " + tam + " bytes");
            return null;
    }

    public String getName() {
            return Constants.CMD_TELA;
    }

    public String getVersion() {
            return "OEM";
    }

    public int printScreen() {

        Image img = control.createScreenCapture(screenSize);
        Image scaledImg = img.getScaledInstance(cliWidth, cliHeight,
Image.SCALE_AREA_AVERAGING);

        return dataFormat.formatAndSend(scaledImg);
    }

    public int printScreen(int x, int y, int zoom) {

        Image img = control.createScreenCapture(new Rectangle(x, y,
cliWidth*zoom, cliHeight*zoom));
        Image scaledImg = img.getScaledInstance(cliWidth, cliHeight,
BufferedImage.SCALE_AREA_AVERAGING);

        return dataFormat.formatAndSend(scaledImg);

    }
}
```

Appendix B

Client

jm2pc.client Package

JM2PCMIDlet.java

```
package jm2pc.client;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;

import jm2pc.client.config.Config;
import jm2pc.client.config.ConfigRepository;
import jm2pc.client.devices.out.CanvasScreen;
import jm2pc.client.devices.out.ImageCapture;
import jm2pc.client.devices.out.Zoom;
import jm2pc.client.i18n.MyResourceBundle;
import jm2pc.client.servers.ListServers;
import jm2pc.client.servers.Server;
import jm2pc.client.utils.CanvasImage;
import jm2pc.client.utils.CanvasAbout;
import jm2pc.client.utils.CanvasWait;
import jm2pc.client.utils.DisplayManager;
import jm2pc.utils.Constants;

public class JM2PCMIDlet extends MIDlet {

        private Display display;
        private DisplayManager displayManager;

        private CanvasWait cvWait;
        private CanvasScreen cvScreen;

        private List lsServers;
        private Menu lsCommands;

        private CanvasImage cvImage;

        private Zoom zoomImg;

        private int serverScreenWidth;
        private int serverScreenHeight;

        private StreamConnection connection;
        private InputStream in;
private OutputStream out;
private DataInputStream dataIn;
private DataOutputStream dataOut;

private ImageCapture imCapture;
private int screenX;
private int screenY;

private long totalBytes;

private boolean cancelled;
private boolean alertOk;
private boolean connected;

private String lastCommand;

public MyResourceBundle messages;

public JM2PCMIDlet() {

        messages = new MyResourceBundle();

        messages.loadMessages();

        cancelled = false;
        alertOk = false;

        zoomImg = new Zoom();

        display = Display.getDisplay(this);

        initComponents();

        totalBytes = 0;

        connected = false;

}
public void initComponents() {

        cvWait = new CanvasWait(this);

        lsServers = new ListServers(this);
        lsCommands = new Menu(this);

        imCapture = new ImageCapture(this);

        cvScreen = new CanvasScreen(this);

        cvImage = new CanvasImage(this);

        displayManager = new DisplayManager(display, lsServers);

}
public synchronized void connect(final Server server) {
        display.setCurrent(cvWait);

        cancelled = false;

        class ConnectThread extends Thread {
```

```
public void run() {
        try {

                String protocol = "socket";
                if (server.getType() == Server.TYPE_BTSPP) {
                        protocol = "btspp";
                } else if (server.getType() == Server.TYPE_SSL)
                        protocol = "ssl";

                StringBuffer sb = new StringBuffer(protocol);
                sb.append("://");
                sb.append(server.getAddress());
                sb.append(":");
                sb.append(server.getPort());

                connected = false;

                connection = (StreamConnection) Connector.open(sb
                                .toString(), Connector.READ_WRITE,
true);

                if (cancelled) {
                        closeConnection();
                        cancelled = false;
                        return;
                }

                in = connection.openInputStream();
                out = connection.openOutputStream();

                dataIn = new DataInputStream(in);
                dataOut = new DataOutputStream(out);

                if (cancelled) {
                        closeConnection();
                        cancelled = false;
                        return;
                }

                dataOut.writeUTF(server.getPassword());
                dataOut.flush();
                boolean authenticated = dataIn.readBoolean();
                if (!authenticated) {
                        throw new Exception(messages
                                        .getMessage("invalidPassword"));

                }

                ConfigRepository repCfg = new ConfigRepository();
                Config cfg;

                if (repCfg != null)
                        cfg = repCfg.load();
                else
                        cfg = new Config();

                alertOk = cfg.isAlertOk();

                String platform = System
                                .getProperty("microedition.platform");

                if (platform == null || platform.length() == 0)
                        platform = "J2ME-JM2PC";
```

```
                    if (cancelled) {
                            closeConnection();
                            cancelled = false;
                            return;
                    }

                    dataOut.writeInt(cvScreen.getWidth());
                    dataOut.writeInt(cvScreen.getHeight());
                    dataOut.writeUTF(platform);
                    dataOut.writeBoolean(cfg.isJpegImages());
                    dataOut.writeBoolean(cfg.isReceivePlugins());
                    dataOut.writeInt(cfg.getMaxBytesDownload());
                    dataOut.writeInt(cfg.getBitsColor());
                    dataOut.flush();


                    serverScreenWidth = dataIn.readInt();
                    serverScreenHeight = dataIn.readInt();

                    totalBytes += 50;

                    if (cancelled) {
                            closeConnection();
                            cancelled = false;
                            return;
                    }

                    connected = true;

                    display.setCurrent(lsCommands);

                } catch (Exception e) {
                    e.printStackTrace();
                    if (!cancelled)
                    {
                            StringBuffer sb = new StringBuffer(messages
                                            .getMessage("errorConnect"));
                            sb.append(' ');
                            sb.append(e.getMessage());
                            showAlert(sb.toString(), AlertType.ERROR,
true,
                                            lsServers);
                    } else {
                            AlertType.ERROR.playSound(display);
                    }
                    closeConnection();
                } finally {
                    try {
                            System.gc();
                            Thread.sleep(200);
                    } catch (InterruptedException e) {

                    }
                }
            }

        }

        Thread t = new ConnectThread();
        t.setPriority(Thread.MAX_PRIORITY);
        t.start();
    }
    public void startApp() {
            Displayable nextDisplayable;
            try {
                    if (connected) {
                            nextDisplayable = lsCommands;
                            display.setCurrent(new CanvasAbout(this,
nextDisplayable));
                    } else {
                            nextDisplayable = lsServers;
                            display.setCurrent(new CanvasAbout(this,
nextDisplayable));
                    }
                    Thread.sleep(2100);
                    display.setCurrent(nextDisplayable);

            } catch (Exception e) {
                    e.printStackTrace();
            }
    }

    public void pauseApp() {

    }

    public void destroyApp(boolean unconditional) {
            if (!unconditional) {
                    disconnect();
            }

            notifyDestroyed();
    }

    public void sendData(byte[] data) throws IOException {

            for (int i = 0; i < data.length; i++)
                    out.write(data[i]);

            out.flush();

            totalBytes += data.length;
    }

    public void disconnect() {

            connected = false;
            displayManager.home();

            try {
                    System.gc();
                    Thread.sleep(200);
            } catch (InterruptedException e) {

            }

            class DisconnectRun implements Runnable {
                    public void run() {
                            try {
                                    sendData("SAIR\n".getBytes());
                            } catch (IOException ioe) {

                            } finally {
                                    closeConnection();
                    }
                }
            }

            Thread t = new Thread(new DisconnectRun());
            t.start();
        }

        private boolean hasResponse(String command) {

            int j = command.indexOf(" ");
            String c;
            if (j == -1)
                c = command;
            else
                c = command.substring(0, j);

            if (c.equals(Constants.CMD_TELA) )
                return true;
            return false;
        }

        public void executeCommand(final String command) {

            lastCommand = command;

            final boolean hasResponse = hasResponse(command);

            class RequestRun implements Runnable {
                public void run() {
                    try {
                        sendData(command.getBytes());
                        if (alertOk) {
                            if (display.getCurrent() instanceof Canvas)
                                AlertType.INFO.playSound(display);
                            else {

    showAlert(messages.getMessage("successCmdMessage"),
                                                    AlertType.INFO, false,

null);
                            }
                        }
                    } catch (IOException e) {
                        StringBuffer sb = new StringBuffer(messages
                                .getMessage("error"));
                        sb.append(": [");
                        sb.append(e.getMessage());
                        sb.append("] ");
                        sb.append(messages.getMessage("connectAgain"));
                        showAlert(sb.toString(), AlertType.ERROR, true,

lsCommands);
                    }
                }
            }

            class ResponseRun implements Runnable {

                public void run() {

                    int i = command.indexOf(" ");
                    String type = command.substring(i + 1, command.length()

- 1);
                    i = type.indexOf(" ");
                if (i != -1)
                    type = type.substring(0, i);

                if (type.equals(Constants.CMD_TELA_TP_INT)) {
                    imCapture.createImageCapture(0, true);
                }
            }
        }

        Thread reqThread = new Thread(new RequestRun());
        reqThread.setPriority(Thread.NORM_PRIORITY);
        reqThread.start();

        if (hasResponse) {

            Thread resThread = new Thread(new ResponseRun());
            resThread.setPriority(Thread.NORM_PRIORITY);
            resThread.start();
        }

    }

    public void sendCommand(final String command, final String param) {
        StringBuffer sb = new StringBuffer(command);
        sb.append(' ');
        sb.append(param);
        sb.append('\n');
        executeCommand(sb.toString());

    }

    public DisplayManager getDisplayManager() {
        return displayManager;
    }

    public Zoom getImageZoom() {
        return zoomImg;
    }

    public void setImageZoom(int zoom) {
        zoomImg.setZoomImg(zoom);
    }

    public void setScreenXY(int x, int y) {

        screenX = x;
        screenY = y;

        int width = cvScreen.getWidth();
        int height = cvScreen.getHeight();
        int zoom = zoomImg.getZoomImg();
        int coverageX = x + (width * zoom);
        int coverageY = y + (height * zoom);

        if (x < 0)
            screenX = 0;
        else if (coverageX > serverScreenWidth) {
            screenX = serverScreenWidth - (width * zoom);
        }

        if (y < 0)
            screenY = 0;
        else if (coverageY > serverScreenHeight) {
            screenY = serverScreenHeight - (height * zoom);
        }

}

public int getScreenX() {
        return screenX;
}

public int getScreenY() {
        return screenY;
}

public void showScreen(Image im, int type) {
        if (im != null) {
                cvScreen.setImage(im);
                cvScreen.setType(type);

                display.setCurrent(cvScreen);
        } else {
                showAlert(messages.getMessage("errorDownload"), null, true,
                                lsCommands);
        }
}

public void showImage(Image im) {
        if (im != null) {
                cvImage.updateImage(im);
                displayManager.pushDisplayable(cvImage);
        } else {
                showAlert(messages.getMessage("errorDownload"), null, true,
                                lsCommands);
        }
}

public void showAlert(String msg, AlertType type, boolean modal,
                Displayable displayable) {
        displayManager.showAlert(msg, type, modal, displayable);
}

public int getServerScreenHeight() {
        return serverScreenHeight;
}

public int getServerScreenWidth() {
        return serverScreenWidth;
}

public void closeConnection() {

        cancelled = true;
        connected = false;

        try {
                if (in != null)
                        in.close();
                if (out != null)
                        out.close();
        } catch (IOException e) {

        } finally {
                try {
                        if (connection != null)
```

```
                    connection.close();
            } catch (IOException e) {
            }
        }
    }
    public CanvasWait getCanvasWait() {
        return cvWait;
    }

    public InputStream getIn() {
        return in;
    }

    public OutputStream getOut() {
        return out;
    }

    public Displayable getMenu() {
        return lsCommands;
    }

    public String getLastCommand() {
        return lastCommand;
    }

}
```

Menu.java

```
package jm2pc.client;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.TextBox;

import jm2pc.client.devices.in.TextKeyboard;
import jm2pc.client.utils.CanvasAbout;
import jm2pc.client.utils.Images;
import jm2pc.utils.Constants;

public class Menu extends List implements CommandListener {

        private JM2PCMIDlet midlet;

        private TextBox tbMessage;
        private TextBox tbkeyboardText;

        private List lsKeyboard;
        private List lsScreen;

        private Command cmSend;
        private Command cmBack;
        private Command cmDisconnect;

        public static final int ABOUT = 0;

        public static final int MSG = 1;
```

```
    public static final int KEY = 2;
    public static final int SCREEN = 3;

    public Menu(JM2PCMIDlet midlet) {
            super(midlet.messages.getMessage("menu"), List.IMPLICIT);
            this.midlet = midlet;

            append(midlet.messages.getMessage("about"), Image.createImage(Images
                            .createIconSobre()));
            append(midlet.messages.getMessage("message"),
Image.createImage(Images
                            .createIconMsg()));
            append(midlet.messages.getMessage("keyboard"), Image
                            .createImage(Images.createIconTeclado()));
            append(midlet.messages.getMessage("screen"),
Image.createImage(Images
                            .createIconTela()));
            cmDisconnect = new Command(midlet.messages.getMessage("disconnect"),
                            Command.EXIT, 1);
            addCommand(cmDisconnect);
            setCommandListener(this);

            cmBack = new Command(midlet.messages.getMessage("back"),
Command.BACK,
                            1);
            cmSend = new Command(midlet.messages.getMessage("send"), Command.OK,
1);

            tbMessage = new TextBox(midlet.messages.getMessage("message"), null,
                            800, 0);
            tbkeyboardText= new TextKeyboard(midlet);

            tbMessage.addCommand(cmBack);
            tbMessage.addCommand(cmSend);
            tbMessage.setCommandListener(this);

    }

    public void commandAction(Command c, Displayable d) {
            if (c == List.SELECT_COMMAND) {
                    switch (getSelectedIndex()) {
                    case ABOUT:
                            midlet.getDisplayManager().pushDisplayable(
                                            new CanvasAbout(midlet, this));

                            break;
                    case MSG:
                            midlet.getDisplayManager().pushDisplayable(tbMessage);
                            break;
                    case KEY:

                    midlet.getDisplayManager().pushDisplayable(tbkeyboardText);
                            break;

                    case SCREEN:
                            midlet.sendCommand(Constants.CMD_TELA,
Constants.CMD_TELA_TP_INT);
                            break;
                    }
            } else if (c == cmDisconnect) {
                    midlet.disconnect();
            } else if (c == cmBack) {
                    midlet.getDisplayManager().popDisplayable();
            } else if (c == cmSend) {
                if (d == tbMessage) {
                        midlet.sendCommand("MSG",
tbMessage.getString().replace('\n',
                                        '|'));
                }
        }
    }

    public Displayable getTeclado() {
        return lsKeyboard;
    }

}
```

4.2.2.2 jm2pc.client.utils Package

DisplayManager.java

```
package jm2pc.client.utils;

import java.util.Stack;
import javax.microedition.lcdui.*;

public class DisplayManager extends Stack
{

    private Display display;
    private Displayable displayableMain;

    private static final int ALERT_DEFAULT_TIME = 2000;

    public DisplayManager(Display display, Displayable displayableMain)
    {
        this.display = display;
        this.displayableMain = displayableMain;
    }
    public void pushDisplayable(Displayable d, boolean forcarEmpilhamento)
    {
      if(forcarEmpilhamento)
      {
                push(display.getCurrent());
      }
        display.setCurrent(d);
    }

    public void pushDisplayable(Displayable d)
    {

      if(!(display.getCurrent() instanceof Canvas))
      {
            push(display.getCurrent());
      }

        display.setCurrent(d);
    }

    public void home()
    {
        while(elementCount > 1)
            pop();
        display.setCurrent(displayableMain);
    }

    public void popDisplayable()
    {
        if(isEmpty())
            display.setCurrent(displayableMain);
        else
        {
            display.setCurrent((Displayable) pop());
        }
    }

    public void showAlert(String msg, AlertType type, boolean modal, Displayable 
displayable)
    {

        Alert alStatus = new Alert("Status", msg, null, type);

        if(modal)
            alStatus.setTimeout(Alert.FOREVER);
        else
        {
            alStatus.setTimeout(ALERT_DEFAULT_TIME);
        }
        if(displayable == null)
            display.setCurrent(alStatus);
        else
        {
            display.setCurrent(alStatus, displayable);
        }
    }

}
```

4.2.2.3 jm2pc.client.servers Package

FormServer.java

```
package jm2pc.client.servers;

import javax.microedition.lcdui.*;

import jm2pc.client.*;

public class FormServer extends Form implements CommandListener {

        private JM2PCMIDlet midlet;

        private ListServers lsServers;
        private Server server;

        private TextField tfDescription;
        private TextField tfAddress;
        private TextField tfPort;
        private TextField tfPassword;

        private Command cmConnect;
        private Command cmCancel;
```

```
public FormServer(JM2PCMIDlet midlet, ListServers lsServers,
            Server server) {
        super(midlet.messages.getMessage("server"));

        this.midlet = midlet;
        this.lsServers = lsServers;

        tfDescription = new TextField(
                    midlet.messages.getMessage("description"), null, 80,
                    TextField.ANY);
        tfAddress = new TextField(midlet.messages.getMessage("host"), null,
80,
                    TextField.ANY);
        tfPort = new TextField(midlet.messages.getMessage("port"), null, 5,
                    TextField.NUMERIC);
        tfPassword = new TextField(midlet.messages.getMessage("password"),
                    null, 30, TextField.ANY | TextField.PASSWORD);

        setServer(server);

        this.append(tfDescription);
        this.append(tfAddress);
        this.append(tfPort);
        this.append(tfPassword);

        cmConnect = new Command(midlet.messages.getMessage("connect"),
                    Command.OK, 1);
        cmCancel = new Command(midlet.messages.getMessage("cancel"),
                    Command.CANCEL, 3);

        this.addCommand(cmConnect);
        this.addCommand(cmCancel);

        this.setCommandListener(this);
    }

    public void setServer(Server server) {

        if (server != null) {
            this.server = server;

            tfDescription.setString(server.getDescription());
            tfAddress.setString(server.getAddress());
            tfPort.setString(String.valueOf(server.getPort()));
            tfPassword.setString(server.getPassword());

            switch (server.getType()) {
            case Server.TYPE_BTSPP:

    tfAddress.setLabel(midlet.messages.getMessage("btAddress"));
                tfPort.setLabel(midlet.messages.getMessage("channel"));
                break;
            default:
                tfAddress.setLabel(midlet.messages.getMessage("host"));
                tfPort.setLabel(midlet.messages.getMessage("port"));
            }
        } else {
            this.server = new Server();
            clear();
        }
    }
    public void clear() {
        tfDescription.setString("");
        tfAddress.setString("");
        tfPort.setString("");
        tfPassword.setString("");
    }

    private boolean validateData() {
        String description = tfDescription.getString();
        String address = tfAddress.getString();
        String port = tfPort.getString();
        String password = tfPassword.getString();

        if (address.trim().length() == 0) {
            midlet.showAlert(midlet.messages.getMessage("noHost"),
                        AlertType.ERROR, true, this);
            return false;
        }
        if (port.trim().length() == 0) {
            midlet.showAlert(midlet.messages.getMessage("noPort"),
                        AlertType.ERROR, true, this);
            return false;
        }
        if (password.trim().length() == 0) {
            midlet.showAlert(midlet.messages.getMessage("noPassword"),
                        AlertType.ERROR, true, this);
            return false;
        }
        if (description.trim().length() == 0) {
            description = midlet.messages.getMessage("unknown");
        }

        server.setDescription(description);
        server.setAddress(address);
        server.setPort(Integer.parseInt(port));
        server.setPassword(password);

        return true;
    }


    private void connect() {
        if (validateData()) {
            try {
                midlet.connect(server);
            } catch (Exception e) {

            }
        }
    }

    public void commandAction(Command c, Displayable d) {
        if (c == cmConnect) {
            connect();
        } else if (c == cmCancel) {
            midlet.getDisplayManager().popDisplayable();
        }
    }

}
```

Server.java

```
package jm2pc.client.servers;

public class Server {

        public static final int TYPE_SOCKET = 0;
        public static final int TYPE_SSL = 1;
        public static final int TYPE_BTSPP = 2;

        private int id;

        private String description;
        private String address;
        private int port;
        private String password;
        private int type;

        public Server() {
                id = 0;
                type = TYPE_SOCKET;
                port = 8888;
        }

        public int getId() {
                return id;
        }

        public void setId(int id) {
                this.id = id;
        }

        public int getType() {
                return type;
        }

        public void setType(int type) {
                this.type = type;
        }

        public String getDescription() {
                return description;
        }

        public void setDescription(String description) {
                this.description = description;
        }

        public String getAddress() {
                return address;
        }

        public void setAddress(String address) {
                this.address = address;
        }

        public int getPort() {
                return port;
        }

        public void setPort(int port) {
                this.port = port;
        }
```

```
	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

}
```

jm2pc.client.devices.in Package

TextKeyboard.java

```
package jm2pc.client.devices.in;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;

import jm2pc.client.JM2PCMIDlet;
import jm2pc.utils.Constants;



public class TextKeyboard extends TextBox implements CommandListener
{

	private JM2PCMIDlet midlet;

	private Command cmEnviar;
	//private Command cmAtalho;
	private Command cmVoltar;

	public TextKeyboard(JM2PCMIDlet midlet)
	{
		super(midlet.messages.getMessage("keyText"), "", 100,
TextField.ANY);

		this.midlet = midlet;

		cmEnviar = new Command(midlet.messages.getMessage("send"),
Command.OK, 1);
		cmVoltar = new Command(midlet.messages.getMessage("back"),
Command.BACK, 2);

		addCommand(cmEnviar);
		addCommand(cmVoltar);
		setCommandListener(this);
	}

	public void commandAction(Command c, Displayable d)
	{
		if(c == cmEnviar)
		{

			String txt = getString();
```

```
            txt = txt.replace('\n', '-');
            StringBuffer sbParam = new StringBuffer();
            sbParam.append(Constants.CMD_TECLADO_TP_TEXTO);
            sbParam.append(' ');
            sbParam.append(txt);
            midlet.sendCommand(Constants.CMD_TECLADO, sbParam.toString());
        }
        else if(c == cmVoltar)
        {
            midlet.getDisplayManager().popDisplayable();
        }

    }

}
```

jm2pc.client.devices.out Package

CanvasScreen.java

```
package jm2pc.client.devices.out;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;

import jm2pc.client.JM2PCMIDlet;
import jm2pc.client.devices.in.TextKeyboard;
import jm2pc.utils.Constants;

public class CanvasScreen extends Canvas implements CommandListener {

        private JM2PCMIDlet midlet;

        private Zoom zoom;

        private Command cmBack;
        private Command cmDoubleClick;
        private Command cmKeyboardText;

        private Image image;

        private int width;
        private int height;

        private int cursorX;
        private int cursorY;
        private int type;

        public boolean pressing;

        public CanvasScreen(JM2PCMIDlet midlet) {

                this.midlet = midlet;

                zoom = midlet.getImageZoom();

                width = getWidth();
                height = getHeight();
```

```
            image = null;
            cursorX = 0;
            cursorY = 0;

            pressing = false;

            cmBack = new Command(midlet.messages.getMessage("back"),
Command.BACK,
                        1);
            cmDoubleClick = new Command("Click",
                        Command.SCREEN, 2);
            cmKeyboardText = new Command(midlet.messages.getMessage("keyText"),
                        Command.SCREEN, 3);

            addCommand(cmBack);
            addCommand(cmDoubleClick);
            addCommand(cmKeyboardText);

            setCommandListener(this);
      }

      class MoveThread extends Thread {

            private int x;
            private int y;

            public MoveThread(int x, int y) {
                  this.x = x;
                  this.y = y;

                  setPriority(Thread.NORM_PRIORITY);
                  start();
            }

            public void run() {
                  while (pressing) {
                        cursorMove(x, y);
                        repaint();
                        try {
                              Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                  }
            }

            public void cursorMove(int x, int y) {
                  cursorX += x;
                  cursorY += y;

                  if (cursorX < 0)
                        cursorX = 0;
                  else if (cursorX > width - 1)
                        cursorX = width - 1;

                  if (cursorY < 0)
                        cursorY = 0;
                  else if (cursorY > height - 1)
                        cursorY = height - 1;
            }

      }
```

```
    public void setType(int tipo) {
          this.type = tipo;
    }

    public void setImage(Image im) {
          image = im;
    }

    public void getServerImage(int x, int y) {

          midlet.setScreenXY(x, y);

          StringBuffer sb = new StringBuffer();
          sb.append(Constants.CMD_TELA);
          sb.append(' ');
          sb.append(Constants.CMD_TELA_TP_XY);
          sb.append(' ');
          sb.append(midlet.getScreenX());
          sb.append(' ');
          sb.append(midlet.getScreenY());
          sb.append(' ');
          sb.append(zoom.getZoomImg());
          sb.append('\n');

          midlet.executeCommand(sb.toString());
    }

    protected void paint(Graphics g) {
          g.setColor(0, 0, 0);
          g.fillRect(0, 0, width, height);

          if (image != null)
                g.drawImage(image, width / 2, height / 2, Graphics.VCENTER
                            | Graphics.HCENTER);
                g.setColor(0, 0, 0);

                g.drawLine(cursorX, cursorY, cursorX, cursorY + 14);

                g.drawLine(cursorX, cursorY, cursorX + 10, cursorY + 10);

                g.drawLine(cursorX, cursorY + 14, cursorX + 10, cursorY + 10);

                g.drawLine(cursorX + 4, cursorY + 12, cursorX + 9, cursorY +
22);
                g.drawLine(cursorX + 6, cursorY + 12, cursorX + 10, cursorY +
21);

                g.setColor(255, 255, 255);

                g.drawLine(cursorX + 5, cursorY + 12, cursorX + 9, cursorY +
21);

                g.drawLine(cursorX + 1, cursorY + 2, cursorX + 1, cursorY +
13);
                g.drawLine(cursorX + 2, cursorY + 3, cursorX + 2, cursorY +
12);
                g.drawLine(cursorX + 3, cursorY + 4, cursorX + 3, cursorY +
12);
                g.drawLine(cursorX + 4, cursorY + 5, cursorX + 4, cursorY +
12);
                g.drawLine(cursorX + 5, cursorY + 6, cursorX + 5, cursorY +
11);
```

```
                g.drawLine(cursorX + 6, cursorY + 7, cursorX + 6, cursorY +
11);
                g.drawLine(cursorX + 7, cursorY + 8, cursorX + 7, cursorY +
10);
                g.drawLine(cursorX + 8, cursorY + 9, cursorX + 8, cursorY +
10);

        }

        public void commandAction(Command c, Displayable d) {

                if (c == cmBack) {
                        midlet.getDisplayManager().popDisplayable();
                }else if (c == cmDoubleClick) {

                        int cursorRealX = midlet.getScreenX() + cursorX *
zoom.getZoomImg();
                        int cursorRealY = midlet.getScreenY() + cursorY *
zoom.getZoomImg();

                        StringBuffer sbCommand = new StringBuffer();
                        sbCommand.append(Constants.CMD_MOUSE);
                        sbCommand.append(' ');
                        sbCommand.append(Constants.CMD_MOUSE_TP_DOUBLE_CLICK);
                        sbCommand.append(' ');
                        sbCommand.append(cursorRealX);
                        sbCommand.append(' ');
                        sbCommand.append(cursorRealY);
                        sbCommand.append(' ');
                        sbCommand.append(0);
                        sbCommand.append('\n');

                        midlet.executeCommand(sbCommand.toString());
                }else if (c == cmKeyboardText) {
                        midlet.getDisplayManager().pushDisplayable(
                                        new TextKeyboard(midlet), true);

                }
        }

        public int getCursorX() {
                return cursorX;
        }

        public int getCursorY() {
                return cursorY;
        }

        protected void keyPressed(int keyCode) {

                pressing = true;

                int control=0;

                switch (keyCode) {
                case KEY_NUM1: {
                                new MoveThread(-3, -3);
                        break;
                }
                case KEY_NUM2: {
                                new MoveThread(0, -3);
                        break;
                }
                case KEY_NUM3: {
```

```
                new MoveThread(3, -3);
            break;
        }
        case KEY_NUM4: {
                new MoveThread(-3, 0);
            break;
        }

        case KEY_NUM5: {
            StringBuffer sbCommand = new StringBuffer();
            sbCommand.append(Constants.CMD_TELA);
            sbCommand.append(' ');
            sbCommand.append(Constants.CMD_TELA_TP_INT);
            sbCommand.append('\n');
            midlet.executeCommand(sbCommand.toString());
            break;
        }
        case KEY_NUM6: {
                new MoveThread(3, 0);
            break;
        }
        case KEY_NUM7: {
                new MoveThread(-3, 3);
            break;
        }
        case KEY_NUM8: {
                new MoveThread(0, 3);
            break;
        }
        case KEY_NUM9: {
                new MoveThread(3, 3);
            break;
        }
        case KEY_STAR: // '*'
        {
            break;
        }
        case KEY_POUND: // '#'
        {
            break;
        }
        case KEY_NUM0: {

                int cursorRealX = 0;
                int cursorRealY = 0;
                StringBuffer sbComando = new StringBuffer();
                sbComando.append(Constants.CMD_MOUSE);
                sbComando.append(' ');
                sbComando.append(Constants.CMD_MOUSE_TP_MOVE);
                sbComando.append(' ');
                sbComando.append(cursorRealX);
                sbComando.append(' ');
                sbComando.append(cursorRealY);
                sbComando.append('\n');

                midlet.executeCommand(sbComando.toString());
            break;
        }
        default:
            doJoystickKeysAction(keyCode);
        }
}
```

```
        protected void doJoystickKeysAction(int keyCode) {

                pressing = true;

                int control=0;

                switch (getGameAction(keyCode)) {
                case UP: {
                                new MoveThread(0, -3);
                        break;
                }
                case DOWN: {
                                new MoveThread(0, 3);
                        break;
                }
                case LEFT: {
                                new MoveThread(-3, 0);
                        break;
                }
                case RIGHT: {
                                new MoveThread(3, 0);
                        break;
                }
                case FIRE: {
                        break;
                }
                }
        }
        protected void keyReleased(int keyCode) {
                pressing = false;
        }
}
```