CONTROL APPLICATION FOR CLIENT/SERVER BASED: INTERACTIVE PUZZLE GAME

by,

EVELYN BT. FRANCIS

WEK 000430

WXES3182

Under the supervision of
Mr. Noorzaily Mohamed Noor
and
Moderatored by,
Dr. Rosli Salleh

Faculty of Computer Science & Information Technology
University of Malaya

Acknowledgements

First of all, I would like to thank to my supervisor, Mr. Noorzaily Mohamed Noor for supervising me during the development of this project. Without his helping, guidance and thought, this project would not be complete. I also want to express my deepest gratitude to Dr. Rosli Salleh for sparing his time as a moderator of this project.

I also want to give my appreciation to all my friends, lecturers for their supports, ideas, and thoughts. Last but not least, I want to thank my beloved family for giving supports and prayers that keep me strong in time of trouble.

I thank GOD for all.

Abstract

Interactive Puzzle Game (IPG)

IPG is a web-enabled application developed through client/server based architecture. Client/server architecture can be define as a design approach that divides the functional processing of an application and distributes it across two or more processing platforms. IPG is design in fulfillment of the client/server environment, which provides the infrastructure of such as middleware, networks, operating systems, hardware that supports the current distributed applications. IPG gives the interactive utility that enables concurrent users in the networked computing. This application supports the gaming features such as winning, loss, and draw, create a new game and many else. IPG also acts and tries to get a feel for the real life gaming capability into the virtual concept. The methodology to implement this application is using the V-Model. The software and programming languages that use to develop this application are varies. Java and Visual Basic are the most essential tools for network and the application itself. This project will try to create a new web-enabled client/server application that ensures additional gaming skill.

Table of contents

List of tables

List of figures

Chapter 1: Introduction

1.0 An Overview: Traditional Puzzle Game

Traditionally, puzzle game was being played minimum number of one person in a room, hall and even everywhere. Then, the number of player can be add as many as they can to play simultaneously and get the result. The main goal of playing this game is only one which is to solve the puzzle and create a full result than needed. The material that was used in the real - world are many and from different types. The demand of applications that related to online and networked game become many and its growth are getting excellent by the use of modem technologies.

The chapter will explain the purpose on this project, the reason of its development, objectives, scopes and many more on this section.

1.1 Problem Definition

As we see, the increases of computer program applications around the world are getting bigger and wider. There are many programming and graphic application nowadays since they are getting popular and the modem technologies are getting improve by time. Most of today's popular desktop database development products having been around for several years in one form or another. In fact, most were around long before the Internet was as popular and as well adopted as it is now. While many are extremely admirable in their range of features, they all have had to be adapted to work with Internet or Intranet style applications, atomic transactions, and the feature rich capabilities of the modem browser.

But, nowadays the application of virtual reality is not the new thing for programmers and many software developers. Concerning the issue related to the subject matter, this project is proposed as to tackle the problem.

Let us consider for two-users want to play game in the different places. First user wants to play the game simultaneously and all the opponent movement available to be seen on the screen. This is also goes the same with user two. The problem with this matter is how is the two players (which is considered as two mouse pointer) can be seen on the same screen on the web but in the different places in the networked computing environment. The first matter is the *concurrent issues* that can be undertake as same as the programming solutions nowadays.

The concern is not only just concurrent but something more. If the two-users play simultaneously, there is problem, example when they want to choose, pick and drag on the same cubic. What will come about in the real life when the thing occurs? They will bump into. Therefore, the problem is the *reality concept,* which this project will try to deal with.

The other one is the *network issues.* This application gives the ability to control and monitor the game between two - users. So that, there will be someone that in charge for this. The administrator has that power over the database server and the application server. So, the programming will be more complicated and difficult.

Web - enabled application is technically client/server application running inside a browser. In order to run a Web-enabled application, user must install additional software on client machine. Normally, the Web-enabled application forces the browser the download some sort of a CAB file or ActiveX.dll. This causes maintenance headache for supporting the application. Eventually, user must deal with dll version conflict, dll registration, configuration, installation, uninstallation, upgrading, etc. Communication between the client and the database typically uses programming protocol such CORBA, DCOM, and sockets. Client machines usually must meet a certain speed/memory requirement to run the application. That translates to time and money. In general, deployment of this type of application is not an easy task when involve with security and other network components. It involves compromising the firewall integrity to allow the requested traffic.

1.2 Project Scope

Identifying the scope of application will help determine if application needs to be integrated with any current systems and problems that need to be solved. If this is the case, the important thing is to identify what data needs to be exchanged between any current system and the application that will develop.

From the issues and problems that arise, for this project, few scopes have been identified and separated to further guide to overcome and construct a more flexible and interactive application.

1.2.1 Synchronization

- This is how the application can be manage or the entire components to build the system can be put together. This also collaborate the concurrent issue.

1.2.2 Communication

- The main part of the network is the communication issues. This is how the network can be solved between the administrator (server) and the users (clients).

1.2.3 Distance (latency)

- The problem comes from the data latency, which is related to the network and the web. These also range on the application server and the request on data from the database application.

1.2.4 Reliability (network congestion and failure)

- Handling reliability .reliability problem are congestion related packet loss and sequence loss, and network failure.

1.3 Project Objective / Purpose

The goal of this thesis is, as the name implies, to create a flexible and interactive puzzle game for concurrent user in the networked distributed computing using the client/server architecture approach. As it is hard to envision something solely on this description a more detailed discussion of the subject matter follows.

The objectives developments of JPG are:

- IPG is develop to fulfill the gaming features that enables distributed computing and application to minimum two users in the distributed computing.
- IPG gives the reliable communication skills and abilities of client/server architecture using the networking environment.
- 1PG provides the ability to give the easy and user-friendly game as same as the real world game.
- IPG is also implementing to offer a multi-function administrator (server) that have the control and monitoring application to the clients.

1.4 Project Challenges

1.4.1 Client-server issues

Besides taking into account all the aforementioned issues when developing a client-server application, there are also some specific designs issues that need to be consider.

Data distribution is necessary in order to obtain certain performance levels. Data distribution does have its own problems such as ensuring consistent data throughout the network, making data distribution transparent, dealing with failures and deciding who should own the data, but these are minimal when compared to the benefits.

Although it may be hard to believe, the main problems associated with client/server architecture tend to be politically rather than technically based, because the CS architecture represents a paradigm shift in relation to the processing of information. Therefore, it is vital that three political problems be addressed before client/server technology is introduced:

- Organization
- Resources
- Functionality

1.4.1.1 Organization

Highly compartmentalized proprietary hardware and software which was previously stored in a central repository known as the corporate data centre, will now be replaced with multivendor hardware and software that may be poorly integrated. This can lead to disputes regarding the control of, and responsibility for, various components of a client/server architecture.

1.4.1.2 Resources

The allocation of human resources plays a significant role in determining whether a client/server application project succeeds or fails. It may seem logical to appoint the old communication / network specialist the role of installing and supporting the new environment, however these specialists will now be confronted with hardware and software that is simplistic, relatively unstable and lack vendor support. This can lead to a situation in which the specialist uncomfortable, or in which the specialist's skills cannot be fairly assessed.

1.4.1.3 Functionality

The introduction of client/server technology forces the data centre to move into new functional areas within the organization. Therefore, it is important to identify exactly what those functions will be especially if it is use in the business environment. The fact that the mainframe environment didn't support the integration of mainstream mission-critical business applications with desktop based administrative, management and planning tools forced end-user departments to take over the desktop and often establish their own LAN in order

to use these functions. Client/server technology is however capable of doing this, but in order to provide the necessary functions the end user departments must have some control over the client/server implementation.

1.4.2 Other Challenges

The construction of an IPG application produces many challenges related to distributed application:

1.4.2.1 Heterogeneity

They must be constructed from a variety of different networks, operating systems, computer hardware and programming languages. The internet communication protocols mask the difference in networks, and middleware can deal with the other differences.

1.4.2.2 Openness

IPG should be extensible - the first step is to publish the interfaces of the components, but the integration of components written by different programmers is challenge.

1.4.2.3 Security

Encryption can be used to provide adequate protection of shared resources and keep sensitive information secret when is transmitted in messages over a network. Denial of service attacks are still a problem.

1.4.2.4 Scalability

A distributed system is scalable if the cost of adding a user is a constant amount in terms of the resources that must be added. The algorithms used to access shared data should avoid performance bottlenecks and data should be structured hierarchically to get the best access times. Frequently accessed data can be replicated.

1.4.2.5 Failure handling

Any process, computer or network may fail independently of the others. Therefore, each component needs to be aware of the possible ways in which the components it depends on may fail and be designed to deal with each of those failures appropriately.

1.4.2.6 Concurrency

The presence of multiple users in distributed system is a source of concurrent requests to its resources. Each resource must be designed to be safe in a concurrent environment.

1.4.2.7 Transparency

The aim is to make certain aspects of distribution invisible to the application programmer so that they need only be concerned with the design of their particular application. For example, they need not be concerned with its location or the details of how its operations are accessed by other components, or whether it will be replicated or migrated. Even failures of networks and processes can be

presented to application programmers in the form of exceptions – but they must be handled.

1.5 Project Motivation

1.5.1 Benefits to Developers

The benefits of client/server design to end-users have been trumpeted for years. Nevertheless, the benefits are not lopsided in end-users favor. There are benefits to developers as well.

1.5.1.1 Reliability

Because server routines are separated (physically) from application code (on the clients), there is a reduced chance of one corrupting the other. This is not the case with static or runtime libraries. Libraries eventually share the same logical address space with the application, risking the traumas of wayward pointer manipulation. Servers, however, are insulated-running as their own process in their own address space and, quite frequently, on their own machine. In the case of machine failure, redundant servers elsewhere on the network (along with the necessary error-recovery code in the RPC library) can continue processing client requests, all transparent to end-users.

In many software or application projects, bugs are found at a rate proportional to their use. The more a function is used the more likely each of its logic branches will be taken. Additionally, the longer a process runs the more likely memory

leaks and other wear-items are likely to be discovered. Server processes epitomize both scenarios: they are frequently used and run for a long time. Both of these features commonly yield more bugs in the development and testing phase than typical, library-only systems-saving developers grief and agony by finding bugs early before users do.

1.5.1.2 Security

RPCs provide as much protection to servers as they do clients. More so, in fact, if access security is implemented. Logon security is an important firewall to database, transaction, and mail servers.

1.5.1.3 Mobility / Porting

Instead of porting the entire system, we can port just the client software or the server software. To get into new markets or new platforms the whole application does not have to move, just the part of developer want to move. This is where the concept of software plumbing originates.

1.5.1.4 Easier Software Distribution

If wants to distributes both client and server software to networked environments, imagine the ease of upgrading users (customers) systems by simply loading the server onto a single machine. This is not unlike the benefits of using runtime libraries; except runtimes must be loaded onto each machine where software that uses them will run.

1.5.1.5 Easier to Debug

Servers are incredibly easy to debug. This is because they are independent from applications, they can be started under a debugger (or attached to by a debugger) without changing the client's environment.

Because of the way servers are traditionally designed, a breakpoint can be set at the entry to the server to catch all requests from all clients. This is also a great place to put logging routines.

1.5.1.6 Easier to Support

Because the server is separated from applications, support personnel do not have to be as concerned about the applications. They can focus on the server and the diagnostics available and ignore the application (until the process of elimination proves the problem is not the server's).

1.6 Project Schedule

1.6.1 Time Schedule

As the task is really quite enormous and many aspects of the final output can only be estimate at this point in time, I will present a pretty rough time schedule:

1.6.1.1 Seeking for information

During this stage, the actual planning of the project will be done. This not only includes identifying the necessary components, but also trying to assess how different external packages (libraries for physics, scripting languages, etc.) can

be used in the context of this project. It also includes doing quick technology prototypes to test how a certain piece of technology works out in code. This seems like valuable extra step to take before trying to integrate all components in a single software system. Estimated time: 3 weeks.

1.6.1.2 Reviewing and comparison of tools

As everything has to work within the client/server architecture, the model for this component should be developed first. This includes the abstraction of the actual communication (local vs. remote) as the implementation of object distribution features. Estimated time: 2 weeks.

1.6.1.3 Project time – table

Below is the time - table of the task and the estimated time to accomplish the project.

Activities / Task	March	April	May	Jun	July	Aug	Sept
Introduction							
Literature Review							
Analysis							
Design							
Coding							
Testing							
Documentation							

Table 1.0 Project Time - Table

Chapter 2: Literature Review

2.0 Introduction

2.0.1 Purpose

The purpose of literature review is finding any related information about the system that plan to develop, to discuss regarding the terminology and indication of same areas covered in the project. After gather information, the information is analysed and the existing system is evaluate so that a better product can be develop.

The research of this project has been done on the resources below:

- Internet
- Books
- Documentation Room
- Journals
- Interviews
- Questionnaires

This chapter will explain the terms and terminologies that being developed that related to this project. This chapter also will review over the existing system and analysing its strength and weaknesses.

2.1 Terms and Terminologies

This project highlighted the terms of Control Application for Client/Server Based: Interactive Puzzle Game. The research of the terms and terminologies itself help the developer gain information about the project that going to be develop.

The most well known terminologies and terms in the networking area are as below:

- Networked environment application
- Client/Server Architecture

2.1.1 Networked environment application

What is a network?

A network is a set of devices (often referred to as nodes) connected by media links. A node can be a computer, printer, or any other device capable of sending and/or receiving data generated by other nodes on the network. The links connecting the devices are often called communication channels.

So, what is a networking environment?

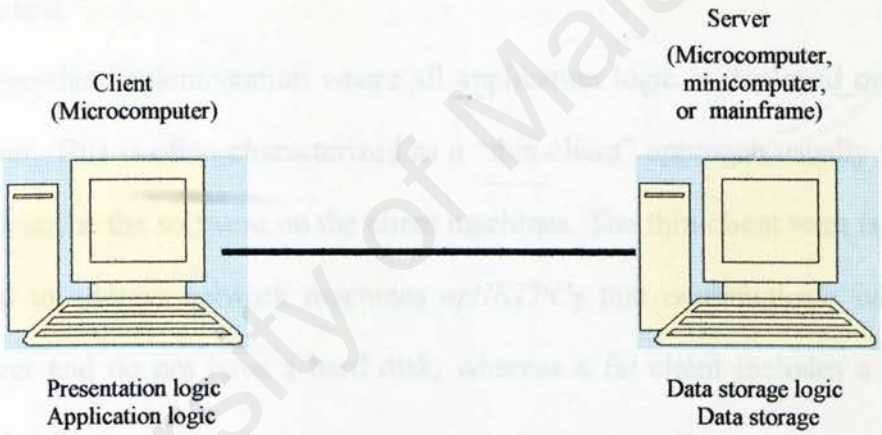
This means in the case of the project, it will be develop by network programming and implement in the environment of set of devices that connected and communicate each other using specified protocols and protected by standards.

2.1.2 Client/Server architecture

The term client-server architecture is a general description of a networked system where a client program initiates contact with a separate server program (possibly on a different machine) for a specific function or purpose. The client exists in the position of the requester for the service, provided by the server.

There are many choices of client / server based architecture. Following are the list of the client/server architecture. The pros and cons of each will the in the next chapter.

Figure 1.0 Client-Server based Architecture (two-tiered)



Two-tier approach on client-server was developed in the early 80's to overcome the disadvantages of file sharing and to address the trend towards graphical GUIs. One-step was to replace the file server with a database server. With this approach the client sends a query to the server, which processes it, and returns the exact information wanted instead of an entire file (which was the case with file sharing). This reduces network traffic and increases update rates, reflecting a true multi-user environment. There is a number of two-tier implementations.

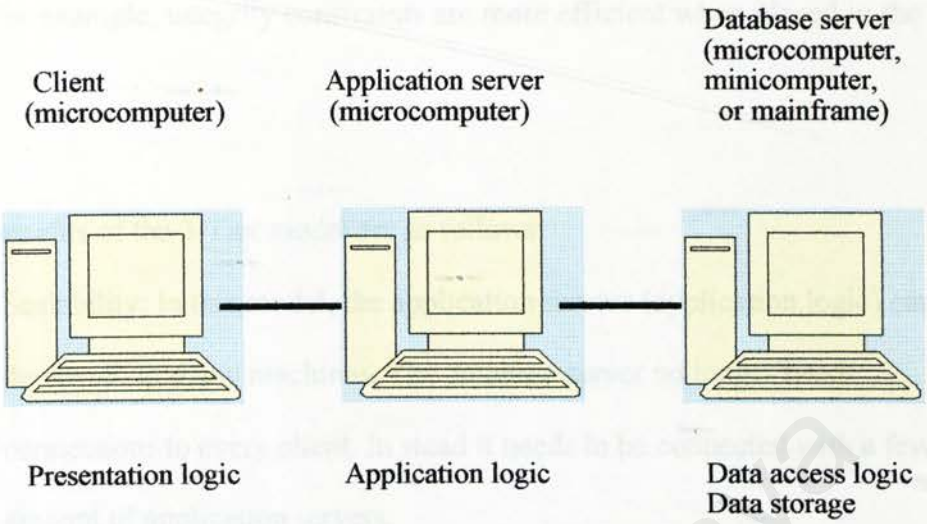
- A two-tier implementation where the bulk of data processing operations is performed in the client side. The database server acts only as a repository of data imposing only simple constraints to the data (like foreign key constraints, not null constraints etc.), ensuring data integrity. This approach is often characterized as "fat client" approach.

  Although "fat client" is a term that refers to software, it can also be used for a network computer with relatively strong processing abilities.
- A two-tier implementation where application logic can be implemented in both the server and the user-end depending on where it is judged to be more efficient.
- A two-tier implementation where all application logic is deployed on the server. This is often characterized as a "thin-client" approach usually used to describe the software on the client machines. The thin-client term is also used to address network machines *orNETPCs* that communicate with a server and do not have a hard disk, whereas a fat client includes a hard disk.

Applications with the following attributes are well suited to 2-tier architecture:

- The application is expected to support a limited number of users (e.g. no more than a few hundred)
- The application is networked and databases are "local" (i.e. not over WAN or Internet)
- A normal level of security is required (data is not overly sensitive)
- Data access from outside applications is minimal

Figure 2.0 Three-Tiered Client-Server Architecture



To overcome the limitations of two-tier architecture, a middle logical tier (also called application server), was added between the input / output device (presentation tier) and the server (data tier). This is where the application / business logic of the system now resides performing a number of different functions like queuing, application execution, database staging and so forth.

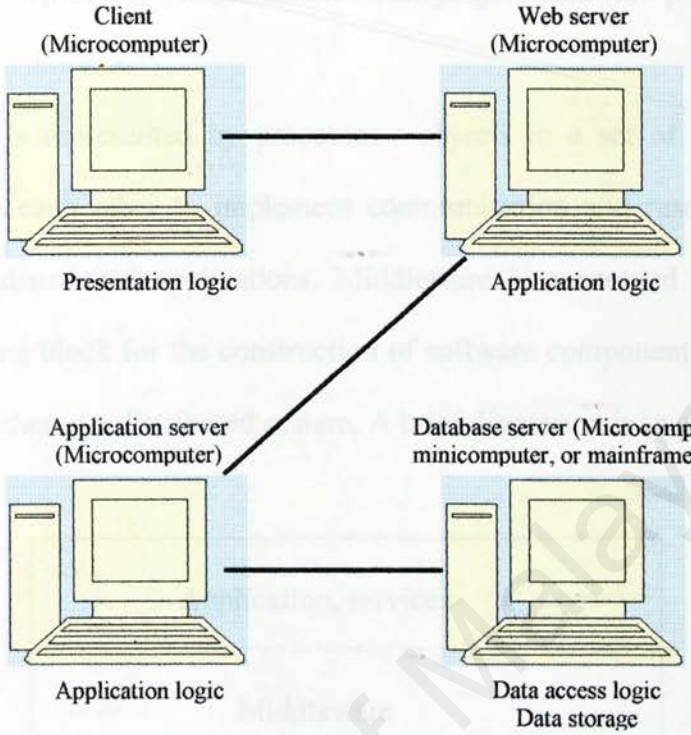
The middle tier can be physically located on a separate machine but this is not always necessary. In an application where the amount of users is limited, the application server can be deployed on the client(s) machines or on the database server, reducing the number of interface connections required making the system faster. As the number of users grows, it can be moved on its separate machine(s), addressing the desired scalability benefits. As the number of users or processing needs grow, the number of machines, that application logic resides, and the processing power in these machines can grow as needed.

It is not always rationale to place application logic in its entirety in the middle tier. For example, integrity constraints are more efficient when placed in the data tier.

The benefits of the 3-Tier model are as follows:

- Scalability: In this model, the application servers (application logic) can be deployed on many machines. The database server no longer needs connections to every client. In stead it needs to be connected with a fewer amount of application servers.
- Data Integrity: Because of the fact that all database updates pass through the middle tier, the middle tier can ensure that only valid data is allowed to be updated in the database thus removing the risk of data corruption from fraud client applications.
- Security: Security is implemented in multiple levels thus making more difficult for a client to access unauthorized data, than it would be if security were placed only on the database. Business logic is implemented on a more secure central server, than if it was distributed across the network.
- Reduced distribution: Potential changes in the business logic can be centralized into one place.
- Hidden Database structure: The structure of the database is hidden from the caller, so a potential enhancement of the database application (due to a new app. Release) will be transparent from him.

Figure 3 0 Four-tiered Client-Server Architecture



This is just almost the same with three-tiered architecture.

This project is using the two-tiered client/server architecture. The pros and cons of it will be discussed later. This explains on how the design of architecture will affect the overall performance. The briefly discussion on this matter will be find in next topic.

2.1.3 Middleware

What is Middleware?

Middleware is a logical software layer placed between software applications and the various system components (operating systems, protocols) that are distributed over a network. It simplifies the development of a distributed computer system

by eliminating the confusion caused from heterogeneous operating systems, communication protocols, implementation languages, hardware platforms.

Middleware is represented by processes / objects in a set of computers that interact with each other to implement communication and resource - sharing support for distributed applications. Middleware is concerned with providing useful building block for the construction of software components that can work with one another in a distributed system. A brief discussion is in chapter 4.

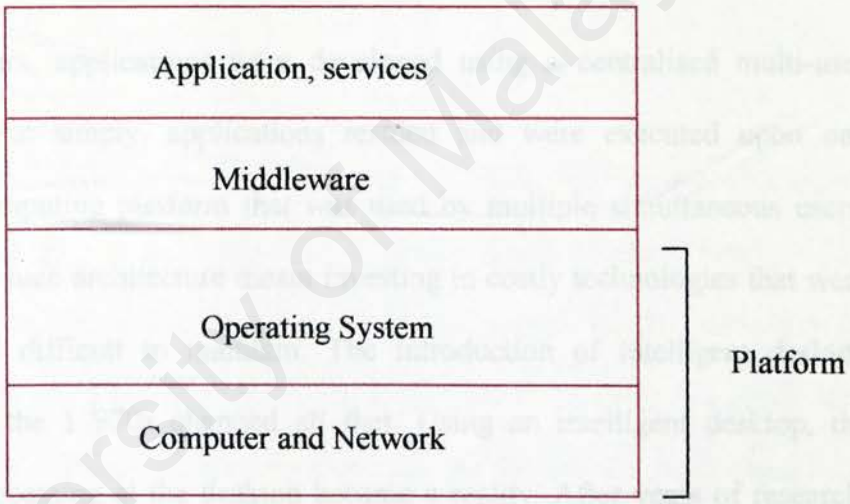


Figure 4.0 The location of Middleware in the layer of distributed systems

Middleware can be implemented using various communication models. International Systems Group Inc. classifies middleware into five distinct categories:

- Remote Procedure Calls (RPC) based Middleware

- Message Oriented Middleware (MOM). MOM products can be categorized as message passing, message queuing, and publish & subscribe products
- Portable Transaction Processing (TP) Monitors
- Object Request Brokers (ORBs) including OLE/COM/DCOM
- Database Middleware

2.2 Hardware and Software

2.2.1 Hardware details

For many years, applications were developed using a centralised multi-user architecture. Put simply, applications resided and were executed upon one centralised computing platform that was used by multiple simultaneous users. Implementing such architecture meant investing in costly technologies that were inflexible and difficult to maintain. The introduction of intelligent desktop computers in the 1 970s changed all that. Using an intelligent desktop, the concept of processing at the desktop became a reality. After years of research, the early 1 980s saw the introduction of client-server architecture.

Client/server architecture can be defined as a design approach that divides the functional processing of an application and distributes it across two or more processing platforms. The division of functional processing is largely based upon which functions provide commonly used services and which perform specific tasks.

The success of client/server architecture is a direct result of its associated benefits such as improved flexibility, the potential for technological advancements and lower cost. These benefits can only be achieved if client/server applications are designed properly.

Client/server processing is seen as the way of the 90s. Its growth in popularity has been mainly due to the fact that it is a technology that eliminates application backlogs, reduces software maintenance costs, increases application portability, improves system and network performance and even eliminates the need for minicomputers and mainframes.

The client/server model is an extension of the shared device model. The shared device model gives personal computers (PCs) the ability to share resources like files and printers. These shared devices are called servers in LAN terminology, (a file server for a file and a printer server for a printer). "Server" is a good word for these shared devices as they receive requests for service from other PCs. However, this technology has a major problem; all processing is done on each PC, meaning great delays especially when concurrent access to a file is required.

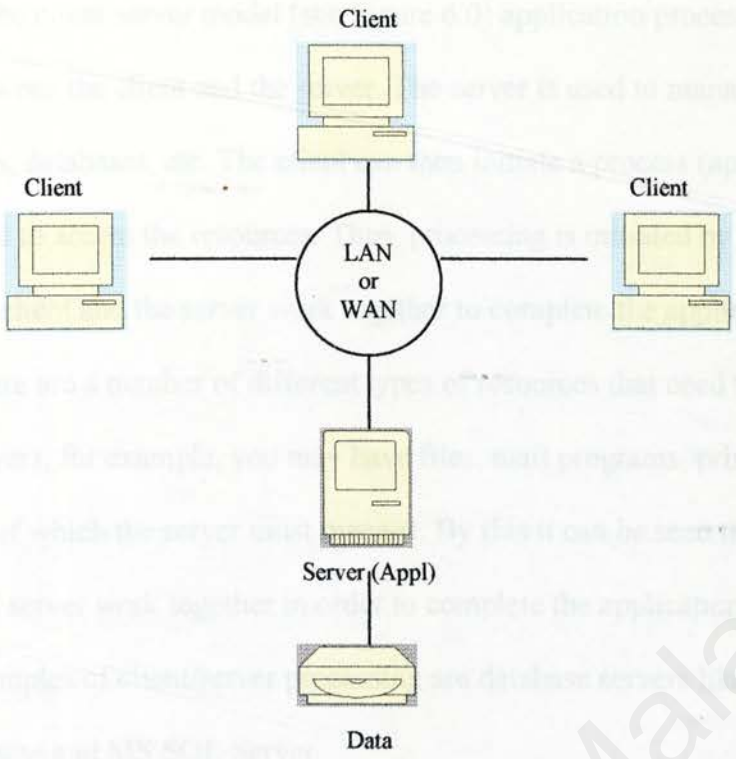


Figure 5.0 is an example of shared device-processing environment

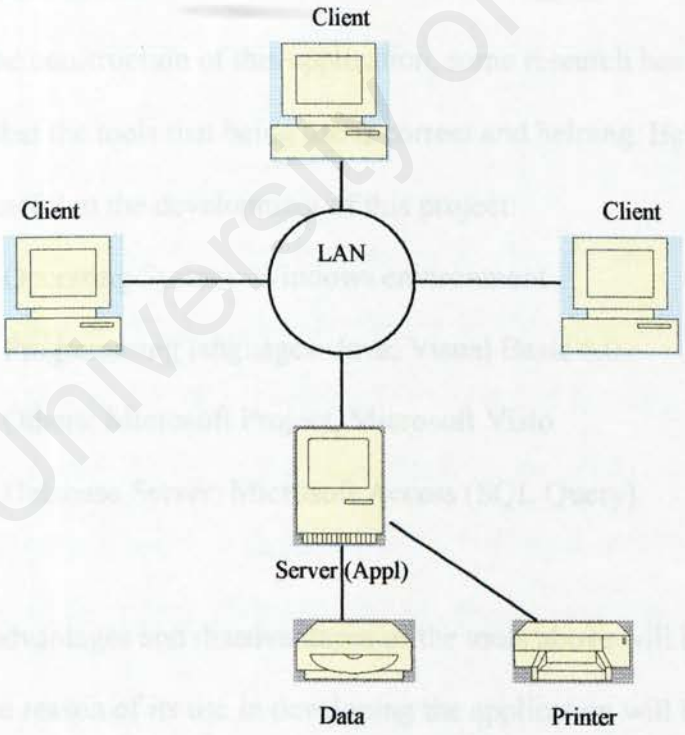


Figure 6.0 Application divided to the clients only

In the client server model (see Figure 6.0) application processing can be divided between the client and the server. The server is used to manage resources, such as files, databases, etc. The client can then initiate a process (application) which is used to access the resources. Thus, processing is initiated by the client but both the client and the server work together to complete the application successfully. There are a number of different types of resources that need to be managed by servers, for example, you may have files, mail programs, printer resources, etc. all of which the server must manage. By this it can be seen that both the client and server work together in order to complete the application together. Excellent examples of client/server processing are database servers like Centura SQLBase, Sybase and MS SQL Server.

2.2.2 Software details

For the construction of this application, some research has been done to make sure that the tools that being use is correct and helping. Below are the software that useful in the development of this project:

- Operating System: Windows environment
- Programming languages: Java, Visual Basic 6.0
- Others: Microsoft Project, Microsoft Visio
- Database Server: Microsoft Access (SQL Query)

The advantages and disadvantages of the tools above will be discussed in chapter 4. The reason of its use in developing the application will be discuss briefly in the next chapter.

2.3 Research on the existing System

2.3.1 BINGO

Traditional BINGO is played in person in a large hail. Players meet at the hall, pay a fee to get in, then the games begins. A night of BINGO consists of many BINGO games played continuously, one after another.

A single BINGO game proceeds like this: Each player has a number of BINGO cards (players can usually play any number of cards). Each BINGO card has *5* rows and *5* columns thus providing 25 spaces.

The columns are labeled from left to right with the letters: 'B', 'I', 'N' 'G', '0'. With one exception (the centre space is "free") the spaces in the card are assigned values as follows:

Each space in the 'B' column contains a number from 1 - 15.

Each space in the 'I' column contains a number from 16 - 30.

Each space in the 'N' column contains a number from 31 - 45.

Each space in the 'G' column contains a number from 46 - 60.

Each space in the '0' column contains a number from 61 - 75.

Furthermore, a number can appear only once on a single card.

Here's a sample BINGO card:

B	I	N	G	O
10	17	39	49	64
12	21	36	55	62
14	25	Free Space	52	70
7	19	32	565	68
5	24	34	54	71

Figure 7.0 Bingo Box

The number of unique BINGO cards is very large and can be calculated with this equation:

// the B, I, G, and 0 columns * the N column

(15 * 14 * 13 * 12 * 11) ^ 4 * (15 * 14 * 13 * 12)

While perhaps interesting to a statistician, the number of possible BINGO cards has nothing to do with player's chances of winning.

You will note that there are 75 possible BINGO numbers:

B1, B2, B3,... B15, I16, 117, 118,...130, N31, N32,...074, 075.

Each of these numbers is represented by a ball in a large rotating bin. Each ball is painted with its unique BINGO number. An announcer spins the bin, reaches in a

selects a ball, and announces it to the room. The players check all of their cards to see if that number appears on their card. If it is, they mark it.

When a player has a BINGO (5 in a row, column, or diagonal), he or she calls out BINGO. The game pauses while the card is verified. If indeed a winner, the game stops and a new game begins. If the card wasn't a winner, the game proceeds where it left off. Each BINGO game proceeds until someone wins (there's always a winner).

What are the chances of winning?

Every BINGO game has a winning card, so a player's chances of winning depend on the number of cards in the game and how many cards s/he is playing. For example, if a player has 12 cards in a game with 1200 cards, the chances of winning for that player is 1 in 100.

2.3.1.1 A Brief Description of the BINGO Programs

The implementation of the BINGO game in Java is a client/server application, and as such, is comprised of two Java programs that run in separate Java Virtual Machines. The Game application is the server, and the Player application is the client.

There are three main applications:

- The Game
- The Player
- The Shared

2.3.1.4 Strength and Weakness

This application uses the client/server architecture fully and was developed fully by Java. This application, I would say that no way of chance that the player can play cheat and it is fully controlled by the server. This game also can be played many clients and the weakness is the game is played by turn. So, no one can make a move if not after the opponent.

2.3.2 TIC - TAC - TOE

This is a classic game of tic-tac-toe. Once two players have connected to the server, the game begins. The first person to down load this applet gets to go first. Once a game ends, a new one may be started by reloading the applet again. This is the example on how the client applets will run.

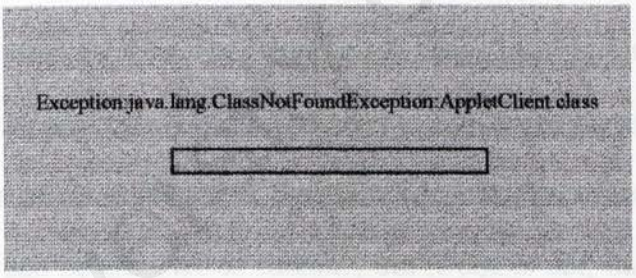


Figure 8.0 The applet tries to upload

If the applet appears like above, the server may not be running currently. If the applet above fails, then you will need to activate the server and reload the applet. The notification window appears if the server is started. Then, there is an icon which asks the user to click to run the client.

Source Code:

Server.java

Connection.java

AnpletClient.java

StreamListener.java

This game consists of two separate programs, the Server and the Applet Client. The Server can theoretically support an infinite number of connections, which comes out to half infinity games, because there are two players per game. Of course being a computer, infinity in this case is defined as an int, or 32 bits, which are 4294967294 simultaneous connections. Anyways, the server spawns a thread for each game which~ in order to keep the players honest, keeps track of which player has selected which square. The server is made of two classes, which are also threads. The Server class keeps looking for available connections while the Connections class is spawned to handle the players of the game.

When the server is started, it creates a ServerSocket to listen on a network port. The port can be assigned at runtime, but if one is not, the DEFAULT_PORT is used. Once the socket has been established, the server enters its run loop. The run loop waits for two clients to connect to the server and then spawns a Connection thread, which handles the actual game. The server is then free to accept new clients.

The Connection thread establishes DataInputStreams and DataOutputStreams for both clients and then enters its run loop, which handles the actual game. The

game engine is pretty simple, the current player is determined based on how many turns have passed (the first client to connect gets to go first). When a turn begins, the DataInputStream is flushed so that any button clicks by the user during the other player's turn are ignored (This is where SkipByteO would have been used if it worked). When the player selects a square, the server determines if it is a valid move and ignores if it is not. If it was a valid move, the server sends each client a message indicating the location of the move, the first client is X and the second is 0. Finally, in a rather ugly if statement with embedded logic, the server checks to see if there was a victory by checking every possible combination of squares for three in a row. Because this is tic-tac-toe, this simple method can be used, but a more complicated check would be necessary for a game like chess. Perhaps in a more complicated game an algorithmic solution would be most efficient. If there was a victory, the winning client is sent a victory code, and the losing client is sent a loser code. The Client is an applet, and really doesn't have much in the way of smarts. The client is simple because it really cannot be trusted. "But it's only a game!" You say. Yes, it is only a game, but that is no excuse to be needlessly sloppy. The client is made of two classes, which are also threads. The AppletClient class deals with user input while the StreamListener class listens to what the server sends the client.

The AppletClient starts as soon as it is down loaded and attempts to contact the server it was down loaded from on the DEFAULT_PORT. Due to security limitations in Java, the client is not capable of changing its port dynamically, so port changes require a recompile. Once a Socket is created, the client creates and

a DataInputStream and a DataOutputStreams to communicate with the server with. Once a connection is established, the program draws the tic-tac-toe board and spawns a StreamListener thread. The AppletClient class has an event driven user interface that sends any button presses to the server for validation. The StreaniListener thread listens to the server a handles all user feed back. The StreamListener's run ioop spins forever processing information from the server. There are a finite number of codes that the server can send to the client. When a code is received, it is tested to find out what the action should be. Moves are computed based on which button was pressed.

2.3.2.1 Strength and Weakness

In order to listen and connect to other machines, the server is a java application not an applet, but it uses CGI technology.

The server is free to accept the new client so it is open game. However, the weakness of this game is that once there has been a victory, the game ends, and the player must reload the applet to play again. The others weaknesses are:

- **Blocking reads:** Java only supplies blocking reads in its stream class. A blocking read is a read in which the program halts execution while it waits for input data to appear. A non-blocking read would simply see that there was no data and continue on. This annoys me because the only way around this is to create a thread that will try to read and thus block while the main thread continues. It seems like a waste to me, but perhaps it is what the programmers at Sun had envisioned.

- **InputStreams and skipping bytes:** Occasionally there comes a time when a programmer would like to skip a few bytes in a stream. The creators of the java.io.DataInputStream class saw this need and created a skipBytes() method. This was wise, but it would help if it didn't always throw 10 exceptions to the extent that it never works properly. At least I could get around this manually with a for loop and readByte~.
- **The final label:** This has got to be the lamest thing in Java. Final keeps a thus labeled method from being inherited, or extended, one of the fundamental powers of Object Oriented Programming. If it were not for this stupid label, I could have made a better stream class based on java.io.DataInputStream that had improvements that would have allowed the class to actually function reasonably.

2.3.3 SLIDERS

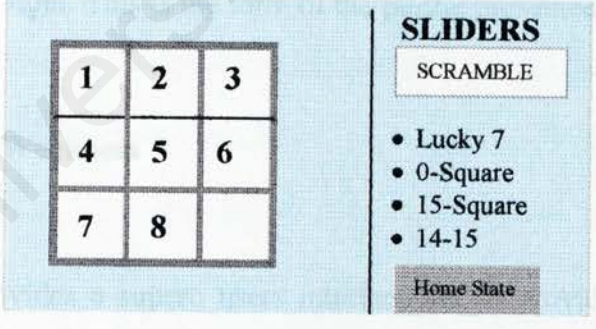


Figure 10 Client Interface for Slider

The goal is to first scramble the board, then move the blocks to recreate the *Home State* of the puzzle. Once the puzzle is scrambled, click on a block with the mouse to move it into the vacant square.

How to play on the 14 -15 Puzzle?

Created in the 1870s by the American puzzlist Sam Loyd, the 14-15 puzzle begins with 15 blocks in regular order, except that blocks 14 and 15 are reversed. The goal of the original puzzle was to move the blocks into correct numerical order, with the vacant square in the lower-right corner (the Home State of the 15 - Square puzzle).

What is interesting?

Loyd, this game developer offered $1,000 to the first person to find the correct solution. The 14 -15 puzzles, also known as the *Boss* puzzle, became quite famous internationally, but no one ever received the prize. The puzzle is in fact unsolvable.

But it is possible to solve the 14 -15 puzzle if the Home State is modified slightly, by placing the vacant square in the upper-left corner of the board rather than the lower right. This is the form of the puzzle presented here.

2.3.3.1 Strength and Weakness

This game provides a superb users interface and it provides function that can manually initiates the server first when it fails to upload. However, this will takes time.

Chapter 3: Methodology

3.0 Introduction

After analysing the information chapter, this chapter will compose the justifications for the methodology to execute the project. This also includes the project planning and all outlined procedures that should be covered in to understand the project requirement better.

What is a Methodology?

These new realities force us to come face to face with the "M" word - methodology. A methodology is simply a clear declaration of "how we do things around here." A methodology can be defined as a collection of procedures, techniques, tools and documentation aids. These collections of procedures, techniques, tools and documentation aids help the software developer to speed up and simplify the software development process. A methodology consist phases as a guide for developer in the choice of techniques that are appropriate. A methodology also helps the system developer to plan, manage, control and evaluate information system project. (Pfleeger, 2001)

Methodologies depend on the type of architectures being used For example, methodologies for developing client/server systems differ from the conventional mainframe development methodologies. In particular, project-planning steps are dependent on the underlying application architecture being deployed. Is ***it***

possible to use expert systems in developing project plans based on a few questions? Source:

An Expert System for Generating Methodologies

Hunter, R., and Conway, B., "C/S 10,000: Methodology Gain without Pain?" Gartner Group Research Note, August 29, 1996.

A methodology may urge one or more techniques for carrying out specific activities. Examples of techniques are root definition, entity-relations (E-R) modeling, normalization, decision table, data flow diagram (DFD) and object oriented.

3.1 Project Development Life Cycle

It is necessary to have a good method of design process before begin any software development project. A systematic approach to the analysis and design of information system will be the main cause of successful software development project. So, this chapter covers the stages of feasibility study, system analysis and database design, coding and testing, implementation and maintaining. Much of this is embodied in System Development Life Cycle (SDLC).

A Process is referred to a series of steps involving activities, constraints and resources that help us produce our intended output. Thus, it is usually involves a set of tools and techniques. When the process involves the building of a certain kind of output it is referred as a life cycle. Therefore, the development process of this system is defined as the Project Development Life Cycle.

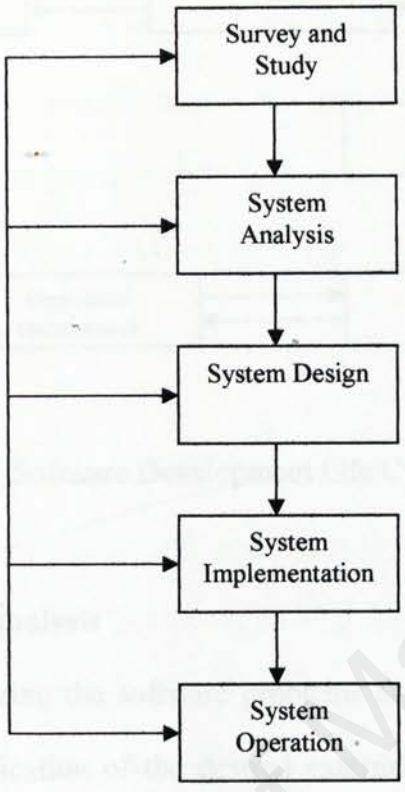


Figure 11 System Development Life Cycle

3.2 The Software Engineering

Software engineering is the application of scientific principles to the orderly transformation of problem into a working software solution and the subsequent maintenance of that software until the end of its useful life.

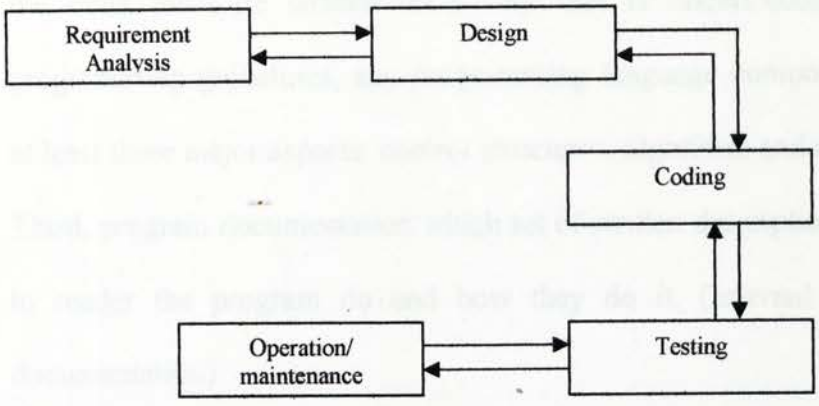


Figure 12 Software Development Life Cycle

- **Requirements analysis**

  Including analyzing the software problems at hand and concludes with a complete specification of the desired external behaviour of the software system to build; also called functional description, function requirements and specification by others.

- **Design Phase**

  Decomposes the software system into actual constituent (architectural) components and then interactively decomposes those components into smaller and smaller sub-components. Define and documents algorithms for each module in the design tree that will be realized a code; also called program design by others.

- **Coding Phase**

  Write the programs that implement the design. First, know the organization's standards and procedures before begin to write code. This standards and procedures can help to organize and avoid mistakes. Second,

the code must be written in a way that is understandable base on programming guidelines, any programming language component involves at least three major aspects: control structures, algorithm and data structure. Third, program documentation which set of written description that explain to reader the program do and how they do it. (Internal and external documentation)

- **Testing Stage**

  Once the program components are coded, we have to test them to ensure the software we produce works properly every time it is run. Test types of faults and failures and how to classify them. The process find faults in components by examine the code, by reading through it, spot algorithm, and syntax faults. Next, compile the code and eliminate remaining syntax faults. Finally, develop test cases to show the input is properly converted to desired output. These also called unit testing. Once the software working correctly and meet the objectives, we combine them into working system. This called integration testing. Test planning after integration testing help us to design and organize the tests and finally we should know when to stop testing by measure the software quality or if it achieve our objectives.

- **Operation / maintenance**

  Maintenance focuses on four major aspects of system evolution simultaneously

  1. Maintaining control over the system's day-to-day functions
  2. Maintaining control over system modifications
  3. Perfecting existing acceptable functions

4. Preventing system performance from degrading to unacceptable levels.

3.3 The V-Model

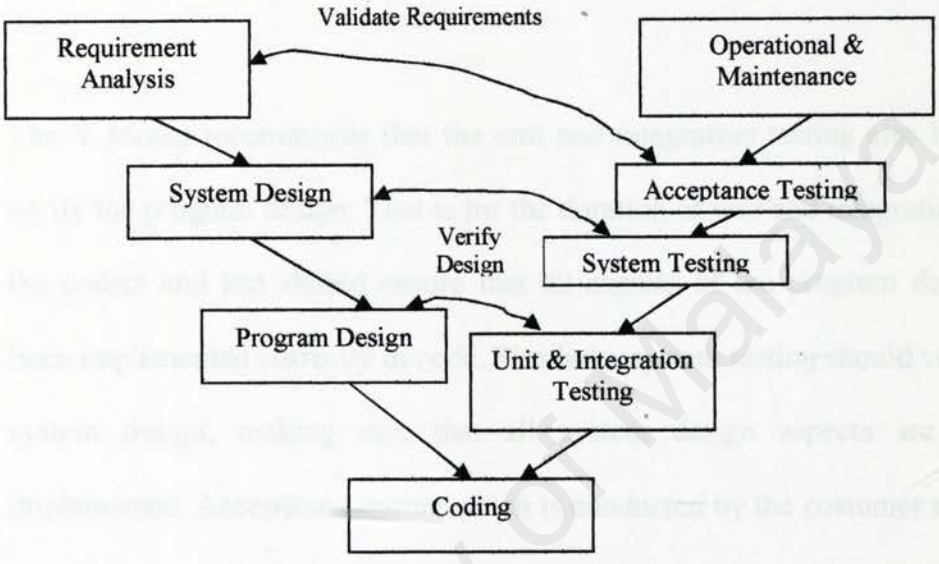


Figure 13 The V – Model

The V Model is a variation of the waterfall model that demonstrates how the testing activities are related to analysis and design (German Ministry of Defence 1992). As shown in the figure 6.0, coding forms the point of V, with analysis and design on the left, testing and maintenance on the right. Unit and integration testing addresses the correctness of programs.

The V Model recommends that the unit and integration testing also be used to verify the program design. That is for the duration of unit and integration testing, the coders and test should ensure that all aspects of the program design have been implemented correctly in code. Similarly, system testing should validate the system design, making sure that all system design aspects are correctly implemented. Acceptance testing which is conducted by the costumer rather than the developer validates the requirements by associating a testing step with each element of the specification; this type of testing checks to see that all requirements have been fully implemented before the system is accepted and paid for.

Why the V Model not other models?

The model's linkage of the left side with the right side of the V implies that if problems are found during verification and validation, then the left side of the V can be re-executed to fix and improve the requirements, design, and code before the testing steps on the right side are re-enacted. In other words, the V Model makes explicit some of the iteration and the rework that are hidden in the waterfall depiction.

Whereas the focus of the waterfall is often documents and artifacts, the focus of V Model is activity and correctness.

Chapter 4: System Analysis

4.0 Introduction

System analysis is a systematic approach to find out the pros and cons of the hardware or software that will be use in the development process of this project. Some analysis has been done to gather and interpret facts and also help diagnosing problems. This will help developer to improve the system.

Following are some of the goals:

- To review the problem faced by the user and find the solutions.
- To study how the new system will be use over the user.
- To acquire knowledge on how this system will be develop with the new emerging technology.
- To identify the major modules to be included in the system
- To identify what are the modules that are feasible to develop and the knowledge and tolls need to have in order to develop them.

4.1 Client/ server Architecture

There are certain requirements, which need to be met in order for the client-server model to run successfully. These are:

- The client and server co-operate in their interactions, which the client must initiate

- The server settles conflicting requests
- A reliable communication link between all clients and servers
- The server controls what services or data can be requested.

Before we can discuss the development of client/server applications, we must identify the foundations of the environment. Client/server relies on several different technologies and it is important to have an understanding of each of them as they can have a dramatic affect on an application. The client/server environment essentially is made up of three dominant technologies:

- Client technologies
- Server technologies
- Network technologies

4.1.1 Client technologies

Client technologies are concerned with the functions that run on the client's workstation. These functions make up a considerable portion of a client/server application and this is why (along with the fact that the network and server only exist to serve the client) many people consider the client the most important of the three technologies.

Developers must be aware of the existing software on client workstations so that they can relieve the server of unnecessary functions using communication mechanisms such as dynamic data exchange (DDE), object linking and embedding (OLE) and remote procedure calls (RPC).

4.1.2 Server technologies

The server acts as the shared resource component of the client/server environment, and so it must be able to service multiple requests simultaneously. The technology used for the server platform can range from the equivalent of a client up to a large mainframe.

Because the server is servicing so many clients, if the server fails, all the clients relying on the services provided by that server fail as well. That is why the criteria for selecting a server are stability, performance and flexibility. Another consideration when selecting and configuring server technology is the functions, which the server is to provide.

4.1.3 Network technologies

In order to implement client/server architecture, the client and server must be able to communicate. Local area networks are commonly used for establishing communication between clients and servers. A local area network can be defined as a combination of software and hardware used to transfer data between stations over a limited geographical distance.

Like client and server technologies, the choice of local area networks should be based on the requirements of your client/server applications in terms of the number of clients and servers, what data is to be transferred between client and server, how much data is being transferred, the required response time and of course the project's budget.

Because of the application is develop in the two-tiered client/server application, below are the main advantages of the 2-tier model:

- Productive: many advanced tools have special optimizations that mean less effort is required when working within the two-tier model (e.g. Visual Basic, PowerBuilder, and Delphi)
- Better Re-use: Where application logic is placed solely on the server, it can be initiated from many client applications and tools.

The main disadvantages of the 2-tier model are:

- Inability to partition application logic
- Lack of robust security
- Lack of scalability

The table below show of the reason of developing client/server architecture in the previous chapter.

BENEFIT	LIMITATION
Scalable	Complexity
Works with multiple vendors/products through middleware	New programming languages and techniques(add stress to personnel)
Improves modularity of web-based systems	More complex to update
No central point of failure	

Table 2.0 Client/server Attributes

Table below is the reason of why this project develops in using the two-tiered architecture model (refer to the previous chapter).

BENEFITS	LIMITATIONS
Separates processing for better balance load on difference server	Greater load on network
More scalable	More difficult to program and test

Table 3.0 N-Tiered versus Two-Tiered Client/Server Architecture

4.2 Why Java not like others such as C++ and C?

Java (with a capital J) is a high-level, third generation programming language, like C, FORTRAN, Smalltalk, Perl, and many others. You can use Java to write computer applications that crunch numbers, process words, play games, store data or do any of the thousands of other things computer software can do.

Compared to other programming languages, Java is most similar to C. However although Java shares much of C's syntax, it is not C. Knowing how to program in C or, better yet, C++, (best of all: Objective C) will certainly help you to learn Java more quickly; but you don't need to know C to learn Java. Unlike C++, Java is not a superset of C. A Java compiler will not compile C code, and C programs need to be changed substantially before they can become Java programs.

What is most special about Java in relation to other programming languages is that it lets you write special programs called applets that can be downloaded from the Internet and run safely within a web browser. Traditional computer programs have far too much access to the system to be downloaded and executed willy-nilly. Although it generally trust the maintainers of various FTP archives and bulletin boards to do basic virus checking and not to post destructive software, a lot still slips through the cracks. Even more software that is dangerous would be promulgated if any web page you visited could run programs on your system. There is no way of checking these programs for bugs or for out-and-out malicious behavior before downloading and running them.

Java solves this problem by severely restricting what an applet can do. A Java applet cannot write to your hard disk without your permission. It cannot write to arbitrary addresses in memory and thereby introduce a virus into your computer. It cannot crash your system.

There's another problem with distributing executable programs from web pages. Computer programs are very closely tied to the specific hardware and operating system they run on. A Windows program will not run on a computer that only runs DOS. A Mac application can't run on a UNIX workstation. VMS code cannot be executed on an IBM mainframe, and so on. Therefore, major commercial applications like Microsoft Word or Netscape Navigator have to be written almost independently for all the different platforms they run on. Netscape

Navigator is one of the most cross-platform of major applications, and it still only runs on a minority of platforms.

Java solves the problem of platform-independence by using *byte code.* The Java compiler does not produce native executable code for a particular machine like a C compiler would. Instead, it produces a special format called Java byte code. Java byte code looks like this

CA FE BA BE 00 03 00 2D 00 3E 08 00 3B 08 00 01 08 00 20 08

This looks a lot like machine language, but unlike machine language Java byte code is exactly the same on every platform. This byte code fragment means the same thing on a Solaris workstation as it does on a Macintosh PowerBook. Java programs that have been compiled into byte code still need an interpreter to execute them on any given platform. The interpreter reads the byte code and translates it into the native language of the host machine on the fly. The most common such interpreter is Sun's program Java (with a little j). Since the byte code is completely platform independent, only the interpreter and a few native libraries need to be ported to get Java to run on a new computer or operating system. The rest of the runtime environment including the compiler and most of the class libraries are written in Java, and are therefore platform independent themselves.

Here are just a few things that others can do on a web page with Java that could not do before:

- Play a sound whenever a user visits a pane
- Play music in the background while the user reads a page
- Use vector graphics instead of bitmaps and GIFs
- Run animation sequences in real-time
- Create forms that verify the user's input
- Create real-time multiplayer interactive games

Java changes the Web from a static, fixed medium to a real-time interactive, dynamic, expandable multimedia environment.

4.3.1 Java Makes Web Pages Dynamic

Until Java, web pages were static. They had text and they had pictures but they had very little else, and they did not change much. For the most part text and pictures just sat there. If you wanted to see something new, you clicked on a link and loaded an entirely new page. Then that page sat there, doing nothing. Java makes web pages dynamic. By using Java, you can make a page change while the user watches it. You can animate pictures. You can change the text that s displayed without reloading the entire page.

4.3.2 Java Adds New Content Types to the Web.

Before Java, users were limited to seeing the kinds of content their web browsers supported; and web developers were limited to the most basic content supported

by their readers' web browsers, generally HTML 2.0 and GIF's. Java lets developers expand the range of content types indefinitely.

For instance, HotJava was the first web browser to include inline sound in a web page. Inline means that the sound plays inside the browser automatically. The reader doesn't need to launch a separate helper application to view it. The sound can play when the reader first visits a page, it can play when the reader clicks a button on a page, or it can play continuously in the background.

However, Java is more than just a web browser with special features. Although HotJava was the first browser to include inline sound and animation, other browsers have long since added this feature.

What makes Java special is that it doesn't just allow you to add new types of content to pages like Netscape and Internet Explorer do. Rather it lets you add both the content and the code necessary to display that content. You no longer need to wait for a browser to be released that supports your preferred image format or special game protocol. With Java, you send browsers both the content and the program necessary to view this content at the same time!

Think about what this means for a minute. Previously you had to wait for all the companies that make the web browsers your readers use to update their browsers before you could use a new content type. Then you had to hope that all your

readers actually did update their browsers. Java compatibility is a feature that any browser can implement and, by so doing, implement every feature!

For instance, suppose you want to use EPS files on your Web site. Previously you had to wait until at least one web browser implemented EPS support. Now you don't wait. Instead, you can write your own code to view EPS files and send it to any client that requests your page at the same time they request the EPS file. On the other hand, suppose you want people to be able to search your electronic card catalog. However, the card catalog database exists on a mainframe system that doesn't speak HTTP. Before Java, you could hope that some browser implemented your proprietary card catalog protocol; or you could try to program some intermediate CGJ on a UNIX box that can speak HTTP and talk to the card catalog, not an easy task. With Java when a client wants to talk to your card catalog, you send them the code they need to do so. You do not have to try to force everything through an httpd server on port 80.

4.3.3 Java Lets Users Interact With a Web Page.

Before Java, people browsed the Web. They moved from site to site passively reading the text and viewing the pictures there, but they rarely interacted with the page or changed it in any meaningful way. Occasionally someone might fill out a form which would be submitted back to the server, but even that was slow and limited to typing text and making menu choices. This was hot stuff in 1975, but is not so exciting in an era where users are accustomed to the interactivity of Quake.

After Java, users can use the keyboard for more than typing text and the mouse for more than choosing from menus. Instead of just reading a page and perhaps filling out a form, users can now play games, calculate spreadsheets, draw pictures, and in general do anything they might do within a window displayed by a traditional computer program. Most importantly, users get immediate feedback. When you press enter in a spreadsheet cell, you don't want to wait for the entire spreadsheet to be sent back to the server and then the entire revised spreadsheet to be sent back to you. You want the update to happen instantaneously. With Java, the calculations are performed on the client system and the results updated locally rather than remotely as would have to be done using a CGI program.

4.3.4 Why Java's a Better Programming Language?

If that were all Java was, it would still be more interesting than a <marquee> or <frame> tag in some new browser beta, but there's a lot more. Java isn't just for web sites. Java is a programming language that can do almost anything a traditional programming language like FORTRAN, Basic or C++ can do. However, Java has learned from the mistakes of its predecessors. It is considerably easier to program and to learn than those languages without giving up any of their power.

The Java language shares many superficial similarities with C, C++, and Objective C. For instance, loops have identical syntax in all four languages,

However, Java is not based on any of these languages, nor have efforts been made to make it compatible with them.

Java is sometimes referred to as C++++--. James Gosling invented Java because C++ proved inadequate for certain tasks. Since Java's designers were not burdened with compatibility with existing languages, they were able to learn from the experience and mistakes of previous object-oriented languages. They added a few things C++ doesn't have like garbage collection and multithreading (the ++) and they threw away C++ features that had proven to be better in theory than in practice like multiple inheritance and operator overloading (the --). A few advanced features like closures and parameterized types that the Java team liked were nonetheless left out of the language due to time constraints. There is still argument over whether the right choices were made. Parameterized types (templates to C++ programmers) may be added in a later revision of Java. Java has learned a lot from previous languages.

4.3.4.1 Java is Simple

Java was designed to make it much easier to write bug free code. According to Sun's Bill Joy, shipping C code has, on average, one bug per *55* lines of code. The most important part of helping programmers write bug-free code is keeping the language simple.

Java has the bare bones functionality needed to implement its rich feature set. It does not add lots of syntactic sugar or unnecessary features. The language

specification for Java is only about eighty pages long compared to a couple of hundred pages for C and even more for C++. Despite its simplicity, Java has considerably more functionality than C.

Because Java is simple, it is easy to read and write. Obfuscated Java isn't nearly as common as obfuscated C. There are not a lot of special cases or tricks that will confuse beginners.

About half of the bugs in C and C++ programs are related to memory allocation and deallocation. Therefore, the second important addition Java makes to provide bug-free code is automatic memory allocation and deallocation. The C library memory allocation functions malloc() and free() are gone as are C++'s destructors.

Java is an excellent teaching language, and an excellent choice with which to learn programming. The language is small so it's easy to become fluent in it. The language is interpreted so the compile-link-mn cycle is much shorter. (In fact, the link phase is eliminated entirely.) The runtime environment provides automatic memory allocation and garbage collection so there is less for the programmer to think about. Java is object-oriented (unlike Basic) so the beginning programmer doesn't have to unlearn bad programming habits when moving into real world projects. Finally, it's very difficult (if not quite impossible to write a Java program that will crash your system, something that you can't say about any other language.

4.3.4.2 Java is Object-Oriented

Object oriented programming was the catch phrase of computer programming in the 1990's. Although object oriented uroarammin2 has been around in one form or another since the Sirnula language was invented in the 1960's, it really took hold in modern GUI environments like Windows, Motif and the Mac. In object-oriented programs, data is represented by objects. Objects have two sections, fields (instance variables) and methods. Fields tell you what an object is. Methods tell you what an object does. These fields and methods are closely tied to the object's real world characteristics and behavior. When a program runs messages are passed back and forth between objects. When an object receives a message, it responds accordingly as defined by its methods.

Object oriented programming is alleged to have a number of advantages including:

- Simpler, easier to read programs
- More efficient reuse of code
- Faster time to market
- More robust, error-free code

In practice, object-oriented programs have been just as slow, expensive and buggy as traditional non-object-oriented programs. In large part, this is because the most popular object-oriented language is C++. C++ is a complex, difficult language that shares all the obfuscation of C while sharing none of C's

efficiencies. It is possible in practice to write clean, easy-to-read Java code. In C++, this is almost unheard of outside of programming textbooks.

4.3.4.3 Java is Platform Independent

Java was designed to not only be cross-platform in source form like C, but also in compiled binary form. Since this is frankly impossible across processor architectures, Java is compiled to an intermediate form called *byte-code.* A Java program never really executes natively on the host machine. Rather a special native program called the Java interpreter reads the byte code and executes the corresponding native machine instructions. Thus to port Java programs to a new platform, all you need to do is run it with an interpreter written for the new platform. You don't even need to recompile. Even the compiler is written in Java. The byte codes are precisely defined, and remain the same on all platforms.

The second important part of Java's cross-platform savvy is the elimination of undefined and architecture dependent constructs. Integers are always four bytes long, and floating point variables follow the IEEE *754* standard for computer arithmetic exactly. You don't have to worry that the meaning of an integer is going to change if you move from a Pentium to a PowerPC. In Java, everything is guaranteed.

However, the virtual machine itself and some parts of the class library must be written in native code. These are not always as easy or as quick to port as pure

Java programs. This is why for example, there's not yet a version of Java 1.2 for the Mac.

4.3.4.4 Java is Safe

Java was designed from the ground up to allow for secure execution of code across a network, even when the source of that code was untrusted and possibly malicious.

This required the elimination of many features of C and C±+. Most notably there are no pointers in Java. Java programs cannot access arbitrary addresses in memory. All memory access is handled behind the scenes by the (presumably) trusted runtime environment. Furthermore, Java has strong typing. Variables must be declared, and variables do not change types when you are not looking. Casts are strictly limited to casts between types that make sense. Thus, you can cast an int to a long or a byte to a short but not a long to a Boolean or an int to a String.

Java implements a robust exception handling mechanism to deal with both expected and unexpected errors. The worst that a Java program can do to a host system is bringing down the runtime environment. It cannot bring down the entire system.

Most importantly, Java applets can be executed in an environment that prohibits them from introducing viruses, deleting or modifying files, or otherwise destroying data and crashing the host computer. A Java enabled web browser

checks the byte codes of an applet to verify that it doesn't do anything nasty before it will run the applet.

However, the biggest security problem is not hackers. It's not viruses. It's not Visual Basic worms transmitted by Outlook Express. It's not even insiders erasing their hard drives and quitting your company to go to work for your competitors. No, the biggest security issue in computing today is bugs. Regular, ordinary, non-malicious, unintended bugs are responsible for more data loss and lost productivity than all other factors combined. Java, by making it easier to write bug-free code, substantially improves the security of all kinds of programs.

4.3.4.5 Java is High Performance

Java byte codes can be compiled on the fly to code that rivals C++ in speed using a 'just-in-time compiler." Several companies are also working on native-machine-architecture compilers for Java. These will produce executable code that does not require a separate interpreter, and that is indistinguishable in speed from C+±. While you will never get that last ounce of speed out of a Java program that you might be able to wring from C or FORTRAN, the results will be suitable for all but the most demanding applications.

As of May 1999, the fastest VM, IBM's Java 1.1 VM for Windows, is very close to C++ on CPU-intensive operations that do not involve a lot of disk I/O or GUI work; C++ is itself only a few percent slower than C or FORTRAN on CPU intensive operations.

It is certainly possible to write large programs in Java. The HotJava web browser, the JBuilder integrated development environment and the *javac* compiler are large programs that are written entirely in Java.

3.2.4.6 Java is Multi-Threaded

Java is inherently multi-threaded. A single Java program can have many different processes executing independently and continuously. Three Java applets on the same page can run simultaneously with each getting equal time from the CPU with very little extra effort on the part of the programmer. This makes Java incredibly responsive to user input. It also helps to contribute to Java's robustness and provides a mechanism whereby the Java environment can ensure that a malicious applet doesn't steal all of the host's CPU cycles.

Unfortunately multithreading is so tightly integrated with Java, that it makes Java rather difficult to port to architectures like Windows 3.1 or the PowerMac that do not natively support preemptive multi-threading.

There is another cost associated with multi-threading. Multi-threading is to Java what pointer arithmetic is to C; that is, a source of devilishly hard to find bugs. Nonetheless, in simple programs it is possible to leave multi-threading alone and normally be OK.

4.3.4.7 Java is Dynamically Linked

Java does not have an explicit link phase. Java source code is divided into *.java* files, roughly one per each class in your program. The compiler compiles these into *.class* files containing byte code. Each *.java* file generally produces exactly one *.class* file.

The compiler searches the current directory and a few other well-specified places to find other classes explicitly referenced by name in each source code file. If the file you're compiling depends on other, non-compiled files, then the compiler will try to find them and compile them as well. The Java compiler is quite smart, and can handle circular dependencies as well as methods that are used before they are declared. It also can determine whether a source code file has changed since the last time, it was compiled.

More importantly, classes that were unknown to a program when it was compiled can still be loaded into it at runtime. For example, a web browser can load applets of differing classes that it has never seen before without recompilation. Furthermore, Java *.class* files tend to be quite small, a few kilobytes at most. It is not necessary to link in large runtime libraries to produce an executable. Instead, the necessary classes are loaded from the user's local system.

4.3.4.8 Java is Garbage Collected

You do not need to explicitly allocate or deallocate memory in Java. Memory is allocated, as needed, both on the stack and on the heap, and reclaimed by the

garbage collector when it is no longer needed. There are no malloc(), free(), or destructor methods. There are constructors and these do allocate memory on the heap, but this is transparent to the programmer.

Most Java virtual machines use an inefficient, mark and sweep garbage collector. Some more recent virtual machines have improved matters quite a bit by using generational garbage collection.

To sum up, Java is a safe, robust, garbage-collected, object-oriented, high-performance, multi-threaded, interpreted, architecture-neutral, cross-platform, buzzword-compliant programming language.

4.4 Why ActiveX?

ActiveX provides the bond that ties jointly a broad assortment of technology building blocks to enable these "active" Web sites.

The primary benefits for ActiveX are:

- Active Web Content with Impact that will attract and retain users.
- Open, Cross-Platform Support on Macintosh, Windows and UNIX operating systems.
- Familiar Tools from a wide assortment of tools and programming language vendors, including Visual Basic, Visual C++, Borland Delphi, Borland

C++, Java, and Java-enabled tools. Developers can use what they know and be productive immediately.

- Existing Inventory of ActiveX controls available today for immediate use by Web producers.
- Industry Standards, with built-in support for key industry and de-facto marketplace standards, including HTML, TCP/IP, Java, COM, and others.

To understand more about ActiveX, we should know what are its elements as ActiveX includes both client and server technologies.

- ActiveX Controls are the interactive objects in a Web page that provide interactive and user-controllable functions and hence enliven the experience of a Web site.
- ActiveX Documents enable users to view non-HTML documents, such as Microsoft Excel or Word files, through a Web browser.
- Active Scripting controls the integrated behaviours of several ActiveX controls and/or Java Applets from the browser or server.
- Java TM Virtual Machine is the code that enables any ActiveX-supported browser such as Internet Explorer 3.0 to run Java applets and to integrate Java applets with ActiveX controls.
- ActiveX Server Framework provides a number of Web server-based functions such as security, database access, and others.

4.5 Why Microsoft Access (using SQL) server?

4.5.1 The Power of SQL

Structured Query Language: The power behind today's data intensive websites. SQL is used to access data in relational database management systems, such as Oracle, Sybase, Informix, Microsoft SQL Server, Access, and others, by allowing users to describe the data the user wishes to see. SQL also allows users to define the data in a database, and manipulate that data. You have the power to create your survey questionnaire to collect the data you wish to manipulate that data, and to analyze the responses. All performed with SQL.

The best part of our service is that you do not have to an expert at SQL or even know any of the commands. The Application software developed handles all of that for you.

4.6 System Requirements

There are three main requirements to fulfill when designing this application. Following are the requirements:

- Operational requirement
- Performance requirement
- Security requirement

4.6.1 Operational requirement

REQUIREMENT	DEFINITION	EXAMPLE
Technical Environment	Special hardware, software, and network requirements imposed by business requirements	Always-on network connection permitting real-time database u dates
System Integration	The extent to which the system will operate with other systems	The system will read and write to the main inventory database
Portability	The extent to which the system will need to operate in other environments	The system may need to operate with handheld devices
Maintainability	Expected business changes to which the system should be able to adapt	The system must accommodate new manufacturing plants

Table 4.0 Operational Requirement

4.6.2 Performance requirement

REQUIREMENT	DEFINITION	EXAMPLE
Speed	Time within which the system must perform its function	Network transaction response time <= 7 seconds
Capacity	Total and peak number of users and the volume of data expected	Maximum of 100-200 simultaneously users at peak times
Availability and Reliability	Extent to which the system will be available to users and the permissible failure rate due to errors	99% uptime performance

Table 5.0 Performance requirement

4.6.3 Security requirement

REQUIREMENT	DEFINITION	EXAMPLE
Access Control	Limitation on who can access what data	Any changes can be made only by System Administrator
Encryption and Authentication	Defines what data will be encrypted where and whether authentication will be needed for user access	Data will be encrypted from the server computer to the client to provide secure ordering
Virus Control	Controls to limit viruses	All uploaded files will be checked for viruses before being saved in the system

Table 6.0 Security requirement

Chapter 5: System Design Analysis

5.0 Introduction

System design consists of all things that you need to do when designing a system. This is the task where need to give priority detailed specifications of the system that is going to be develop. The design process put together all knowledge obtained from the analysis process. This chapter focused both on the logical and physical or technical aspects of the system, which are based on the data, processes, and interface components.

The system is carried out two levels, logical design and physical design. At the logical level, physical or implementation details are ignored. The focus at this level is on the processes, not on the physical aspects of the system. The physical design follows the logical design. At this level, the hardware, physical file and storage media, as well as the language used need to be specified.

5.1 System Module

When designing this application, I have identified number of module that need to be consider and separate as to make this phase easier and helpful to manage. Generally, module is a component of the system that provides services to other components; however, it does not mean that they are separate system.

Followings are the module to implement:

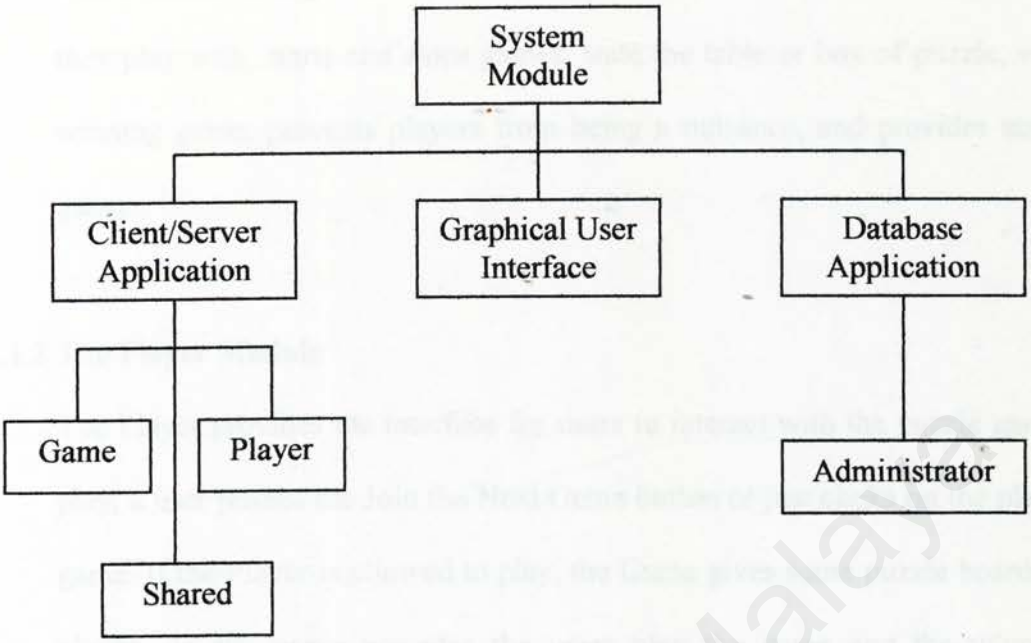


Figure 14 The Design Module

5.1.1 The Client/Server Application Module

This module covered the discussion of the design and development of client/server applications from various solutions and study of problem, system planning, analysis, development, and implementation. This is where to design event- driven applications utilizing applications management tools within the database. This is where the discussion on object- oriented programming capabilities and benefits.

This module separate into three sub-modules:

- The Game Module
- The Player Module
- The Shared Module

5.2.1.1 The Game Module

This module manages the games. It registers players and creates the puzzle that they play with, starts and stops games, state the table or box of puzzle, verifies winning game, prevents players from being a nuisance, and provides status of game.

5.2.1.2 The Player Module

The Player provides the interface for users to interact with the puzzle game. To play, a user pushes the Join the Next Game button or just clicks on the play new game. If the Player is allowed to play, the Game gives some puzzle board to the player. As the game precedes the users play the game and the winner are announced.

5.2.1.3 The Shared Module

This module provides all the shared features that involve in the actions. This module will have the coding that manages the shared application.

5.1.2 The Graphical User Interface (GUI) Module

In this module, it will combines all the fundamentals of GUI design by using driven programming concepts and techniques to create several small graphical user interfaces for the Windows environment. Visual Basic and Java will be used to design, layout and implement screen controls, menus and graphical objects. Programming techniques that being used such as logic flow and input validation, as well as general GUI guidelines.

5.1.3 The Database Application Module

In this module, students will continue the creation of applications using SQL tools. Programming techniques such as logic flow, input validation, data management, object linking and embedding (OLE), on-line help, and graphical integration will be working in this module. All the information and data of the game will be working in this module.

5.1.3.1 The Administrator Module

This is where all the control and monitoring tools will be doing. Therefore, all the additional features that need to accomplish in this application such as the game, client and database will be implementing in this module. The administrator has all the responsibilities and authority of the server and database.

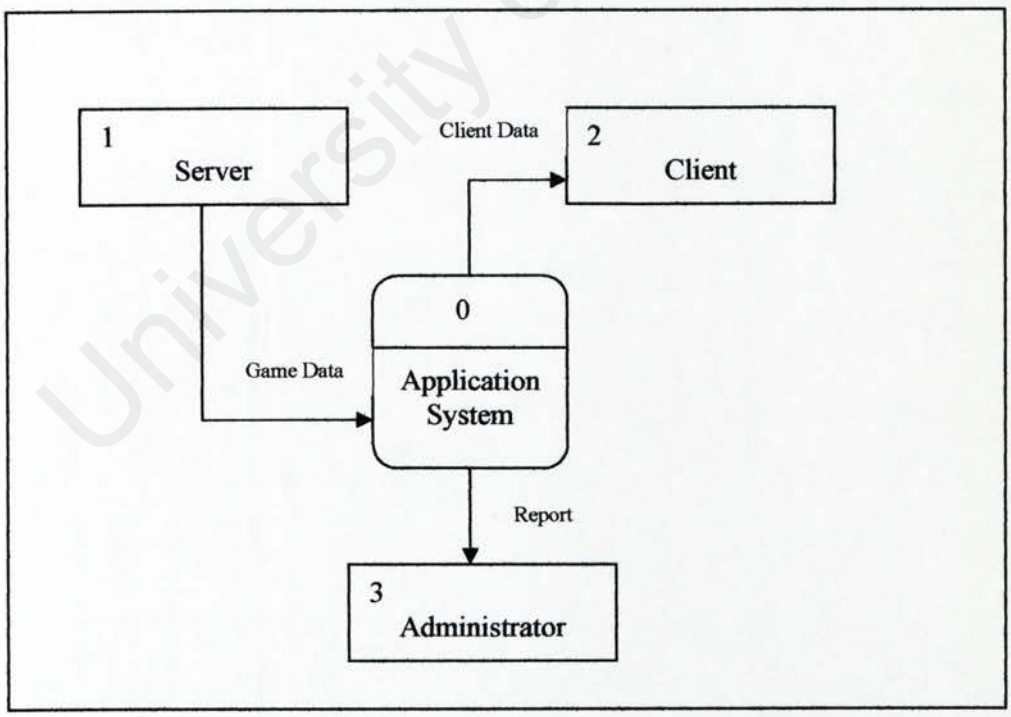


Figure 15 Context Diagram of System

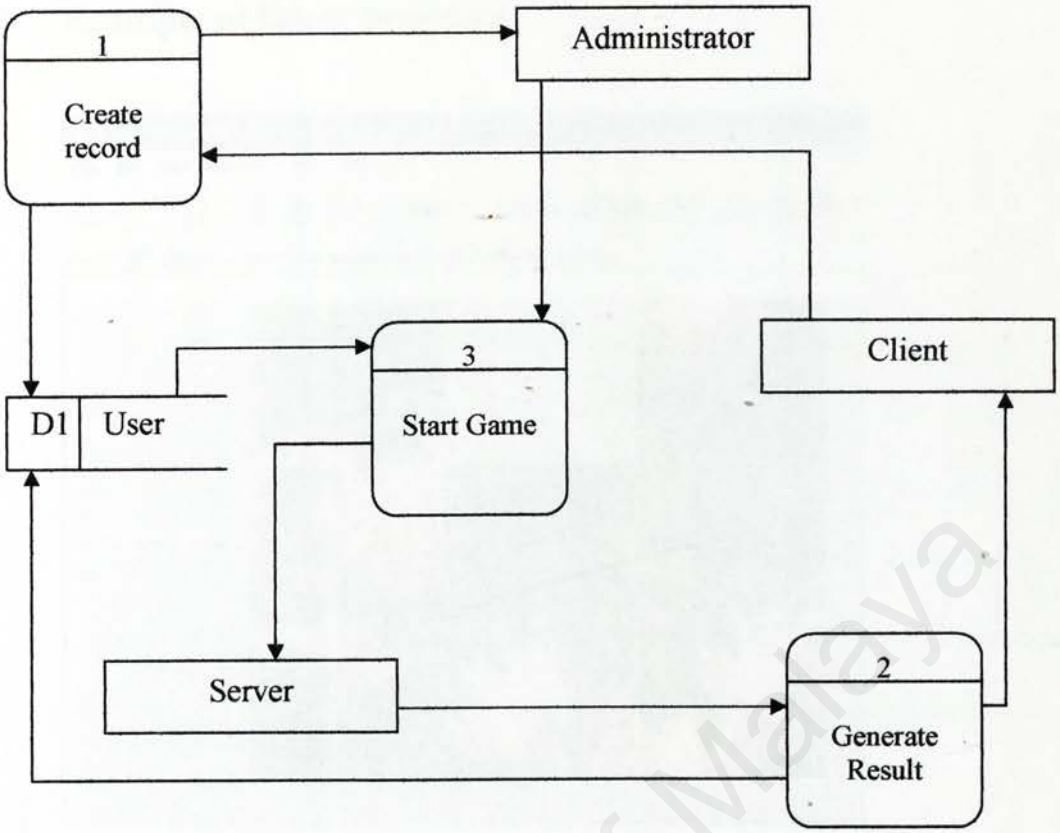


Figure 16 Data Flow Diagram (DFD)

5.2 Example of Client Interface

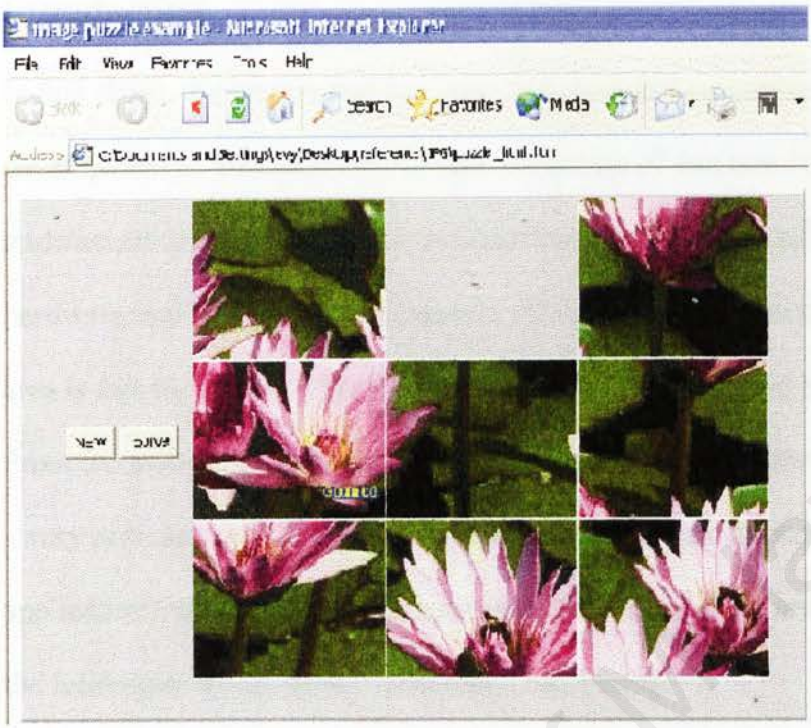


Figure 17 Example Interface

Chapter 6: System Development/Implementation

6.0 Introduction

In this phase, the system that has been designed will be implemented. The development of software tools to take advantage of fast new hardware traditionally lags of a cluster of homogeneous workstations all running behind hardware production and development. The major process in this development area is that the system must fulfill the requirements that have been written down. From the analysis and research, the hardcore of the algorithms and modules offer a very profound attention on how to employ this design to perform the desired application that go after the requirements. But firstly, there are few overviews on the techniques to implement distributed computing.

6.1 System Development Strategies

In this phase, there will be a few strategies on how is this application going to be develop. This is important part of development process, a simple mistake can ruin the whole plan.

6.1.0 Distributed Concurrent/ Parallel Processing

6.1.0.1 CORBA, RMI and Sockets

CORBA allows applications implemented in differing languages the ability to communicate while RMI is a Java feature. Sockets provide a lower level of communication where the programmer is responsible for establishing the method of communication. This is the tools that provide a convenient tool for utilizing the distributed systems.

6.2 The Changes

There are many problems occur during the beginning of the development of the system. So, few changes have been done to make sure that it can be develop and success.

6.2.0 Design Module Changes

The design module that has been changed is the application module and the database module. The hierarchy is shown below:

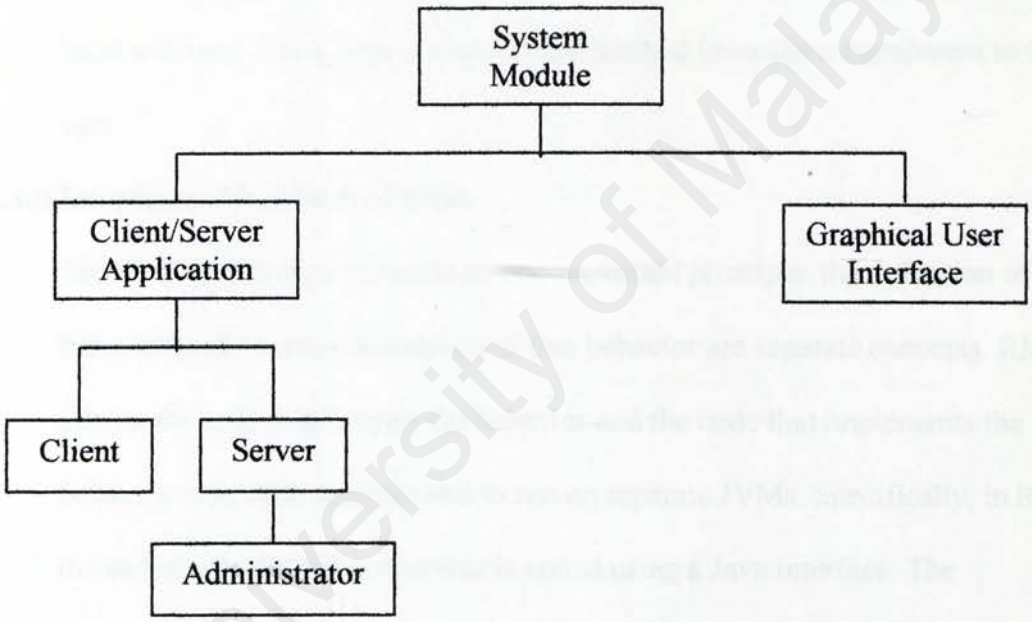


Figure 18 Design Module

6.2.1 Algorithm Changes

At the early stage of development process, the RMI was chosen.

Why RMI (Java Method Invocation)?

Unlike sockets, Java RMI is a higher form of communication where the messaging is handled by the environment rather than explicitly by the programmer. It is the programmer's responsibility to provide the appropriate environment for the remote methods to be invoked. Once the environment has been established, the messaging is handled as a simple method invocation, hence the name. Java remote method invocation establishes interobject communication. It is fundamental to the Java model. If the particular method happens to be on a remote machine, Java provides the capability to make the remote method invocation appear to the programmer to be the same as if the method is on the local machine. Thus, Java makes remote method invocation transparent to the user.

6.2.1.0 Interfaces: The Heart of RMI

The RMI architecture is based on one important principle: the definition of behavior and the implementation of that behavior are separate concepts. RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVMs. Specifically, in RMI, the definition of a remote service is coded using a Java interface. The implementation of the remote service is coded in a class. Therefore, the key to understanding RMI is to remember that *interfaces define behavior* and *classes define implementation.* While the following diagram illustrates this separation,

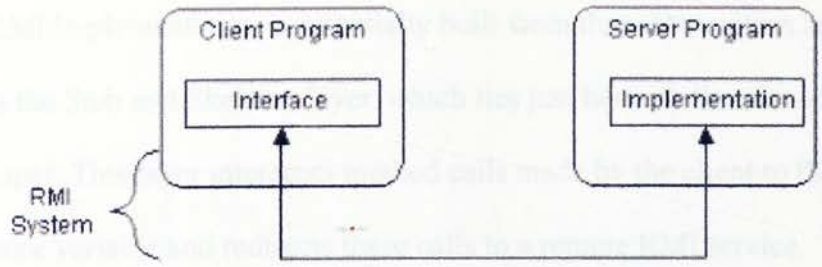
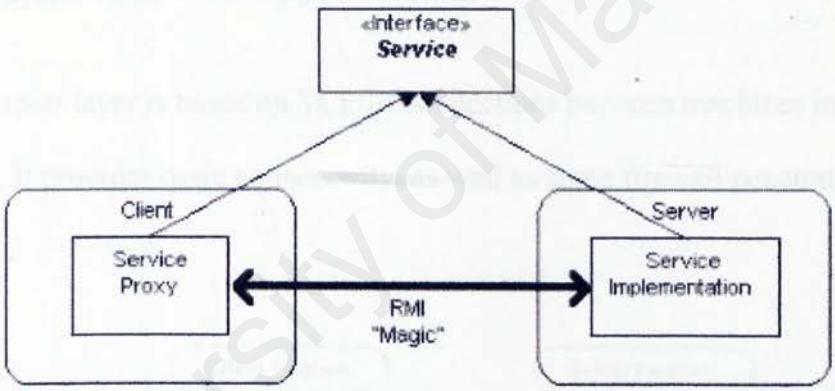


Figure 19 RMI Interfaces

Java interface does not contain executable code. RMI supports two classes that implement the same interface. The first class is the implementation of the behavior, and it runs on the server. The second class acts as a proxy for the remote service and it runs on the client. This is shown in the following Figure 20,



A client program makes method calls on the proxy object, RMI sends the request to the remote JVM, and forwards it to the implementation. Any return values provided by the implementation are sent back to the proxy and then to the client's program.

6.2.1.1 RMI Architecture Layers

The RMI implementation is essentially built from three abstraction layers. The first is the Stub and Skeleton layer, which lies just beneath the view of the developer. This layer intercepts method calls made by the client to the interface reference variable and redirects these calls to a remote RMI service.

The next layer is the Remote Reference Layer. This layer understands how to interpret and manage references made from clients to the remote service objects. In JDK 1.1, this layer connects clients to remote service objects that are running and exported on a server. The connection is a one-to-one (unicast) link. In the Java 2 SDK, this layer was enhanced to support the activation of dormant remote service objects via *Remote Object Activation.*

The transport layer is based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

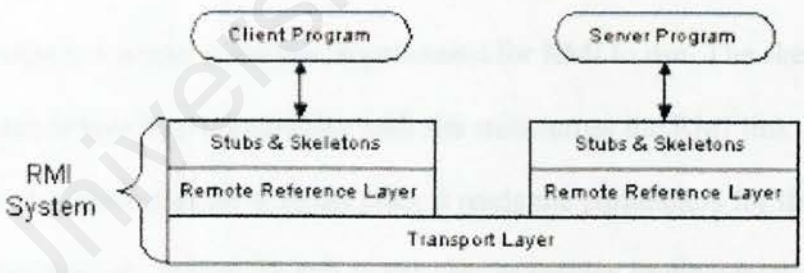


Figure 21 RMI Architecture

By using a layered architecture each of the layers could be enhanced or replaced without affecting the rest of the system. For example, the transport layer could be

replaced by a UDP/IP layer without affecting the upper layers. So, things are simple.

6.2.1.2 Stub and Skeleton Layer

The stub and skeleton layer of RMI lie just beneath the view of the Java developer.

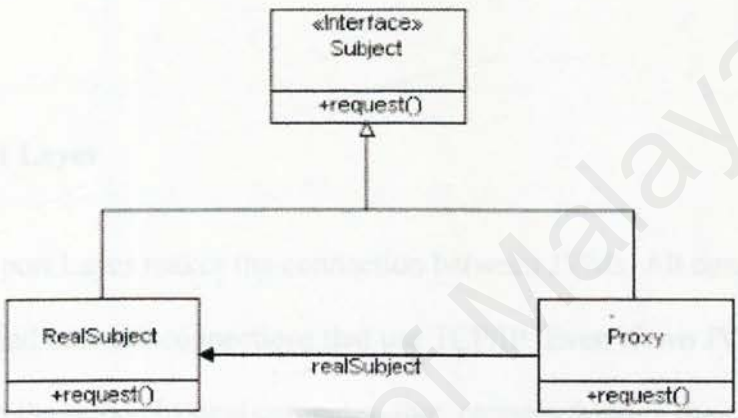


Figure 22 Stubs and Skel

A skeleton is a helper class that is generated for RMI to use. The skeleton understands how to communicate with the stub across the RMI link. The skeleton carries on a conversation with the stub; it reads the parameters for the method call from the link, makes the call to the remote service implementation object, accepts the return value, and then writes the return value back to the stub.

6.2.1.3 Remote Reference Layer

The Remote Reference Layers defines and supports the invocation semantics of the RMI connection. This layer provides a RemoteRef object that represents the link to the remote service implementation object.

The stub objects use the invoke() method in RemoteRef to forward the method call. The RemoteRef object understands the invocation semantics for remote services.

6.2.1.4 Transport Layer

The Transport Layer makes the connection between JVMs. All connections are stream-based network connections that use TCP/IP. Even if two JVMs are running on the same physical computer, they connect through their host computer's TCP/IP network protocol stack.

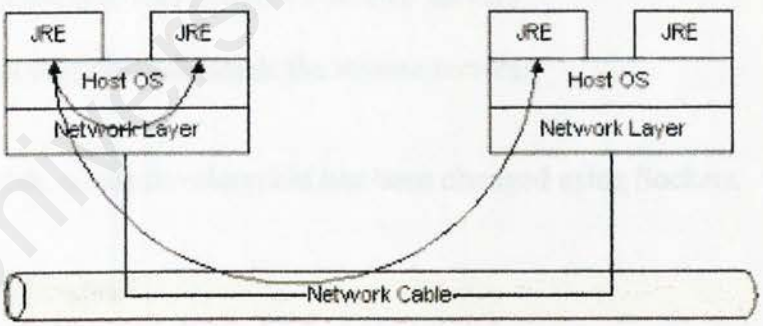


Figure 23 Transport Layer

The RMI transport layer is designed to make a connection between clients and server, even in the face of networking obstacles. While the transport layer prefers

to use multiple TCP/IP connections, some network configurations only allow a single TCP/IP connection between a client and server (some browsers restrict applets to a single network connection back to their hosting server).

In this case, the transport layer multiplexes multiple virtual connections within a single TCP/IP connection.

6.2.1.5 Using RMI

A working RMI system is composed of several parts.

- Interface definitions for the remote services
- Implementations of the remote services
- Stub and Skeleton files
- A server to host the remote services
- An RMI Naming service that allows clients to find the remote services
- A class file provider (an HTTP or FTP server)
- A client program that needs the remote services

This isn't work, so the development has been changed using Sockets.

Why these isn't work?

The reason is because the RMI is not really suitable for developing this application. The application need to change a little of its code so that it can be implemented the RMI techniques. The RMI uses Object which is really hard to

deal with the registry and need to do lots of re-configuration. There are also "unknown problems" which sometime give headache and need "brain-storming".

6.3 Java Message Passing – Sockets

In Java it is possible to "serialize" a data structure. This means that the data as well as its structure is packaged by the language to be stored on an external device, such as a disk drive, or sent over a network to a remote host. Several common data structures, including arrays and dynamic lists, can be serialized. To do socket communication in any language the programmer must establish the socket number to communicate through the server. The client, requesting services from the server then writes to that socket number. Java, through its ObjectInputStream and readObject facilities can send and receive a serialized data structure, essentially packing and unpacking, or "deserialize" the data communicated. Writing to a socket is much like writing to a file. Normally, when communicating through a socket a programmer must send small units of data, but with serialization Java is able to send complete data structures. To keep the implementations consistent, this development stage did not use this serialization feature, but rather only the features provided by the communication mechanism. Thus, small portions of data were repeatedly written to socket.

6.4 Development Processes

In this stage, the main process is to develop a java file. Below are the files that have been developed to fulfill the requirements. The java file using RMI:

1. IPGServer.java

2. IPGClient.java
3. IPGManager.java
4. RMIManager.java
5. ServerAdministrator.java
6. StoppableServer.java
7. ClientGUI.java
8. ServerImpl.java
9. ConnectListener.java
10. DisconnectListener.java

After tested (will be discuss in Chapter 7), the above codes have been changed to Sockets based. Below are the java files:

1. Puzzle.java
2. PuzzleServer.java
3. PuzzleFrame.java
4. PuzzlePanel.java
5. PuzzlePanel_Success.java

6.4.0 The Primary Classes

This section provides a few diagrams that show the most important classes in the game applications and how they related. This section also provides a description of the classes in the diagrams focusing on their relationships and their purpose.

6.4.0.1 Descriptions of Every Class

Puzzle.java

Main program: initialization; connection

Uses: PuzzleFrame, PuzzlePanel

Used by: PuzzleServer to receive connections from client and load Client Interface of game.

Do: Web page online interface, asks for username, connect with server.

PuzzleServer.java

Main program: Overall Control

Uses: Puzzle, PuzzleFrame, PuzzlePanel

Used by: Puzzle to make connections and ask to load Client Interface of game.

Do: Give; Load client puzzle frame included with puzzle panel.

PuzzleFrame.java

Main program: custom frame

Used by: Puzzle, PuzzlePanel, PuzzleServer

Uses: --

Do: Sets up custom frame handler

PuzzlePanel.java

Main program: Panel for puzzle window: graphing; keyboard and mouse event handling

Uses: PuzzleFrame, PuzzlePanel_Success

Used by: Puzzle, PuzzlePanel, PuzzleServer

Do: Initializes puzzle, custom-sized window with puzzle on top, buttons for restarting with different puzzle, or exiting window on bottom arrow keys change puzzle tests if puzzle finished

PuzzlePanel_Success.java

Main program: "success" window

Uses: --

Used by: PuzzlePanel

Do: Displays small window, informs how long it took to finish the puzzle button to exit the window

6.5 System Development Platform

In this section, the platform that being used is the windows environment and this doesn't cause much trouble because java is platform independent. The major problem here is the java developer that always encounters various conflicts. This really time consuming problem because it need to do some configurations and settings before the program successfully running.

6.6 Hardware and Software Configuration for Development Platform

Below is a list of software tools used for the development platform:

SOFTWARE	FUNCTION
Windows XP Professional, Windows NT	Operating system for development environment
Sun ONE Studio 4 Update 1, Community Edition Oracle9i JDeveloper	Java source code editor and developer
Internet Explorer Version 6.0	Running Applet and web page
J2sdk, SEv1.4.1 Jdk1.3.1 J2re, SE v1.4.1 Java Web Start	Compile and run Java source codes

The hardware is the personal computer (pc) or microcomputer.

Chapter 7: System Testing

7.0 Introduction

In this stage all the developed java files will be running with testing phase. The reason of this phase is to make sure the implemented program can run and make sure that it is free from any error.

7.1 Integrate, Compiling, Running and Debugging

After all the java files created, now the process to compile and running the file. Below are the steps:

1. Compile all files to create the generic class file (ex: PuzzleServer.class).
2. Test connection (using socket) of one client to one server. If this works, test with more clients and the thread.
3. Test the connections.Integrate these classes with the PuzzleServer class and Client class.
4. Test the integration of these classes to see whether they are able to function well.
5. Integrate Puzzle class (login function) with Client class (PuzzleFrame).
6. Test integration.
7. Integrate all files included PuzzlePanel_Success class.
8. Test overall. Try to solve puzzle and look whether the PuzzleFrame_Success connected or not.
9. Test the message passing between server and client.
10. Try to connect the client using web page and check whether the applet connected to the Server.

11. Implement the system (server and clients) in one computer using Microsoft Internet Information Server (IIS).
12. Test the implementation of the system between 2 clients (this currently didn't works)
13. Evaluate overall programs.

In this stage, the process repeated over and over again until it functions correctly. If there is class or an integration of classes doesn't work, then it need to modify to make sure they work.

7.2 Testing Environment

To make sure that this application runs properly, the computer must at least have Java Virtual Machine (JVM) installed and the JDK files. These files must be configured properly and the settings of its path and classpath correct. To run the applet which is the web page for online login, the specified user pc or client must have the Internet Explorer (latest version will be suitable).

7.3 Testing Phase

This section is really important to make sure that the system is running correctly. In this stage, the process of validation and debugging will be done separately into different stage.

This phase is divided into 3 steps:

1. Component testing
2. Integration testing
3. System Testing

7.3.0 Component Testing

Unit testing is testing stage that tests the very basic unit of the application and individual program components. This stage will test all the basics and simple source codes (hardcore). This will included the testing of program modules. The hardcore module is all the java files (ex; Puzzle.java). Each of these files will be compiled separately. This stage helps reducing errors in program module. Tests are derived from the developer's experience

7.3.1 Integration Testing

After integrate all the modules, it must be test the effectiveness whether they are connected or not. The interconnected module must be working properly. Sometimes in this stage the module when it is tested with the integration with other module, problems occurred and the modification has to be done on that module (this happened while implementing using the Java RMI). Integration testing can be a very tiresome process because of the logical errors sometime cannot be detected. Testing of groups of components integrated to create a system or sub-system. Tests are based on a system specification

7.3.2 System Testing

This stage will be doing only after the unit and integration testing done their job. This testing to make sure that the program can run to the various environments with the specification configuration software and hardware.

Chapter 8: System Evaluation

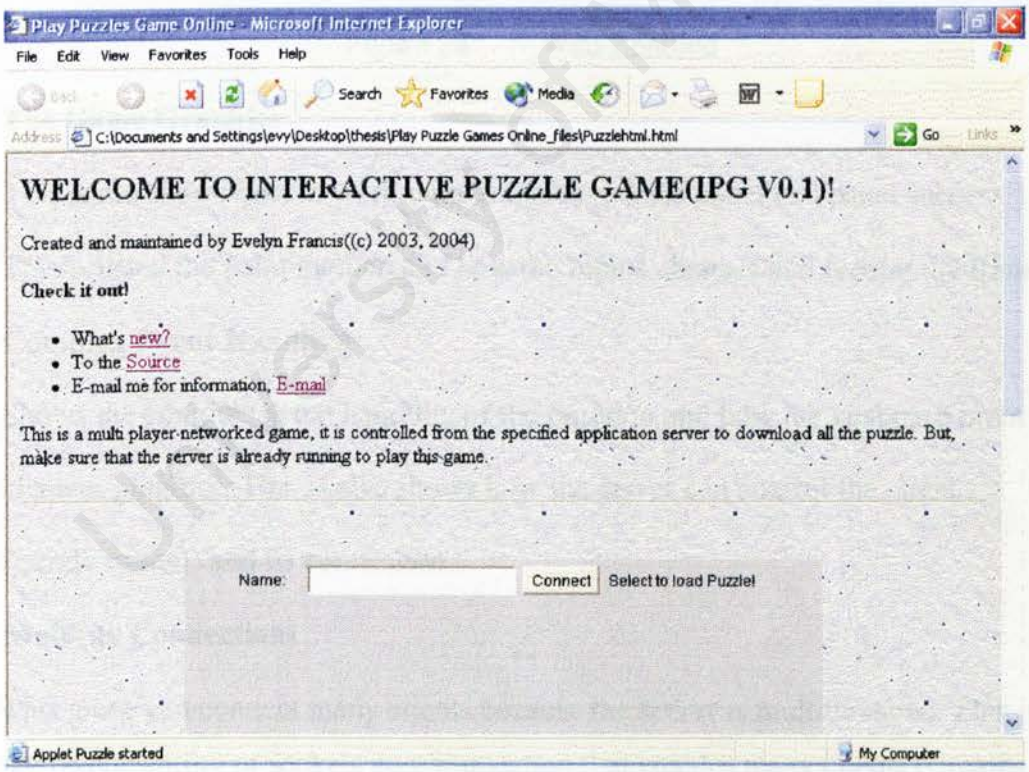
8.0 Introduction

This chapter will give some discussion of what the pros and cons of this application. This also shows how the system can be implemented in the future with better functions.

8.1 What's Interesting?

8.1.0 The web-enabled

This is web-enabled game because it is one of Java features. It enables the web browser and applet can easily embed in any web programming such as html or xml.



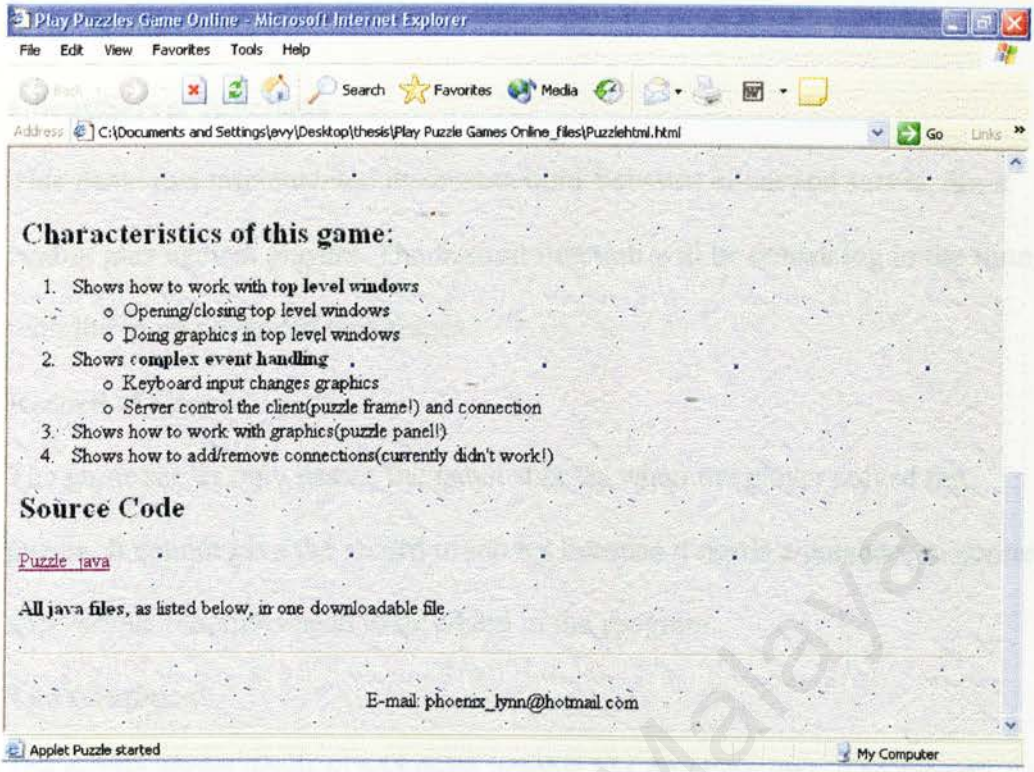


Figure 24 Web-Enabled

8.1.1 The Game Graphics

This game shows how to work with graphics (puzzle panel and panel success!). This is using the paint method and how the inputs changes and repaint the frame.

8.1.2 Complex Event Handling

Shows the complex event handling of the program and how the keyboard input changes graphics. This is also shows how the server can control the client (puzzle frame!) and its connection.

8.1.3 Multiple Connections

This game can connect many clients because the server is multithreaded. This is the responsibility of sockets communication that enables many clients (streams – TCP).

8.2 The Weaknesses

8.2.0 Client/Server functions

This game just implemented the connections between client and server. So, it cannot play against players. The desired function will be enhancing in the future with the more advanced techniques.

8.2.1 Record of Game

The game server only passes the panel success when the player solved the puzzle. It cannot save the record of moves because it needs a database to connect and various functions need to be added in the program.

8.2.2 The Graphical

The puzzle applet needs to add more graphic has few functions. The server application also needs to implement the function such as stop and restart the server. The puzzle frame and panels also need to add more graphics so that it will look nice.

8.3 Future Enhancements

8.3.0 Concurrently Play Puzzle

In the future, the puzzle can be played with other players (minimum 2 persons). Below is the code of moving the puzzle using mouse event. If this is used in client/server architecture based, the concurrent puzzle can be implemented.

```
public boolean handleEvent(Event evt) {
   if (evt.id == Event.MOUSE_UP) {// the core of mouse event
   /*argument here*/ }
```

```
if (evt.id == Event.MOUSE_DOWN) {
// argument is here
}
 if (evt.id == Event.MOUSE_MOVE && inst==0) {
 // your argument is here
}
 switch(evt.id) {
// example using mouse event action
case Event.ACTION_EVENT: {
    if("Solve".equals(evt.arg)) { if(inst==0) cheat=1; }
    if("Restart".equals(evt.arg)) { restart(); }
    if("Instructions".equals(evt.arg)) { instructions(); }
    if("Resume".equals(evt.arg)) { resume(); }
    if("Rotate".equals(evt.arg)) { rotate(); }
    }
  }
 return true;
 }}
```

8.3.1 Server With More Functions

Add more functions in the server such as can send stream messages or add function chat while playing puzzle with others. Follows may be useful,

```
// initializing the data input stream and data output stream
try {
        input = new DataInputStream( connection.getInputStream() );
        output = new DataOutputStream(connection.getOutputStream() );
         } catch( IOException e) {
              e.printStackTrace();
                 System.exit(1);
              }
// sending message
output.writeInt(221);
output.writeUTF("HELLO");

// receiving message
int Value = input.readInt( );
String Name = input.readUTF( );
```

8.3.2 Interactive Multimedia Features

This game will be more fun if the multimedia features such as sound effects while playing or when the connection created or when the message send and received included.

Chapter 9: Conclusion

9.0 Introduction

This chapter is the overall view of the project and also what are the problems during the development of this project.

9.1 Problems

9.1.0 The Experience of Using Java

This is really challenging because the java programming is not like basic programming language such as C or C++. To develop this project using java need more research or do lots of exercises in programming. The most important is that how to debug and identify problem if there is an error during the implementation. This also need to know how to deal with the event method (using event handling), when is the right time to use and so on. So, more practice and spend lots of time and focus on how to do programming in java. To makes thing easier is to study the others developed program and adapt the same concept from other programming language such as C++ and many more.

9.1.1 The Game Cannot Play Concurrently

This is the hardest part in developing the project. The problem is that this puzzle game never been done before play concurrently by using mouse. So, it needs more study on how to implement concurrently using mouse event. This is also hard because of java's complexity and experience of the developer.

9.1.2 The Database

Java provides connectivity (JDBC) in developing an application that uses more files and need to save information in the database. So, the problem occurs when connecting the database and make sure that it can run and work

properly together without any conflict. This is tough to do because the application will too large to fix if there is a problem.

9.2 Project Conclusion

As the benefits of client/server technologies are realized, the demand for high quality applications will increase. The real power of client/server applications comes from the developer's ability to tailor each application to suit the needs of the business and provide that service in an optimal way at the point of need without overly impacting the needs of other users.

During this development project, there are many processes of learning. It is especially in programming. This is because this project taught to be more openness of doing programming which is nobody's perfect. This project also taught to be patient especially while dealing with problems. The problems can be solved but because of lack of experience and time they cannot be tackling on time. It is also the decision on technique to develop (in this case; RMI) which really time consuming.

Even though many problems that happen during the development of this project, the experiences that gained is the most valuable thing and it might be useful in future. More skills and programming experience besides the knowledge of implementing the network computing. Therefore, this project achieved and utilizes all the knowledge and constructs a better application in the future.

Appendix

User Manual

User Manual

IPG v1.0

Faculty of Computer Science and Information Technology

This is a simple way to compile and running this program. If the class files (ex., PuzzleServer.class) are already in the same folder with java files (ex., PuzzleServer.java), you do not need to compile it anymore. Just follow these steps and you'll get the result.

[1] By using MSDOS, go to the path of folder containing puzzle game.
Just is case the copies of java files are not compiled, you need to compile it first by using this command,

-- javac {AllJavaFiles}.java

[2] Run the server by using this command. (Make sure that JDK file is already installed in your pc – minimum requirement is JDK1.2)

-- java PuzzleServer

[3] Then, run the applet. This is by either click on the html file - puzzle.html or run the appletviewer like below,

-- appletviewer puzzle.html

Then, enter username.

[4] If connected, the puzzle game will be loaded and it is ready!

[5] If success to solve the puzzle, the puzzle panel will be loaded and you will know your moves.

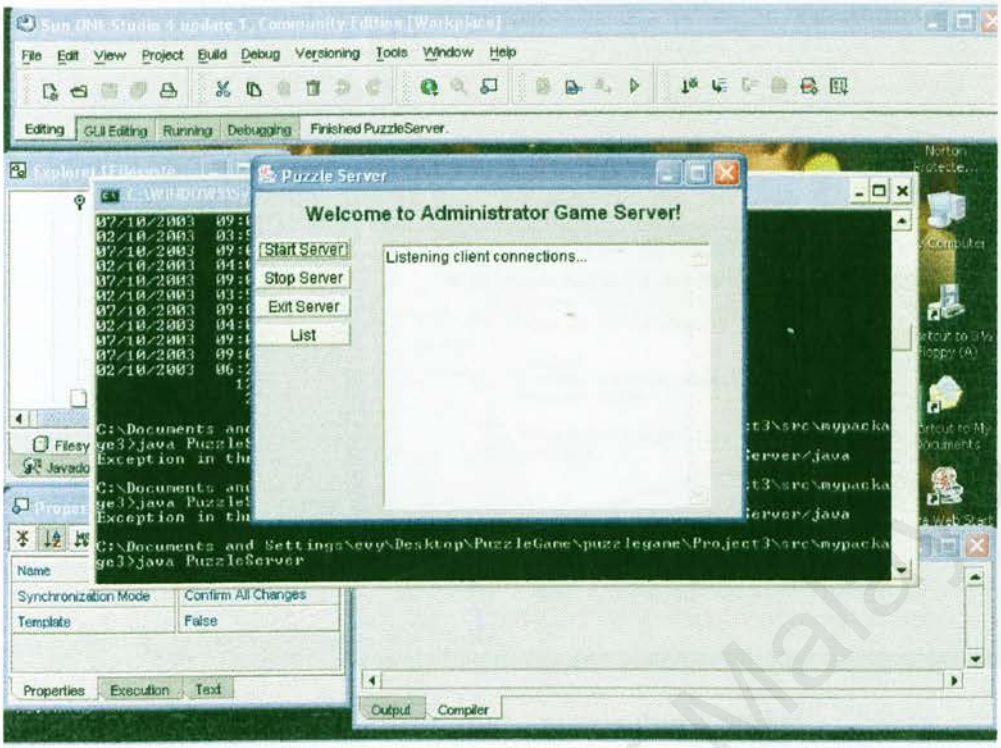
Try this and have fun!

Any enquiry, please contact me.

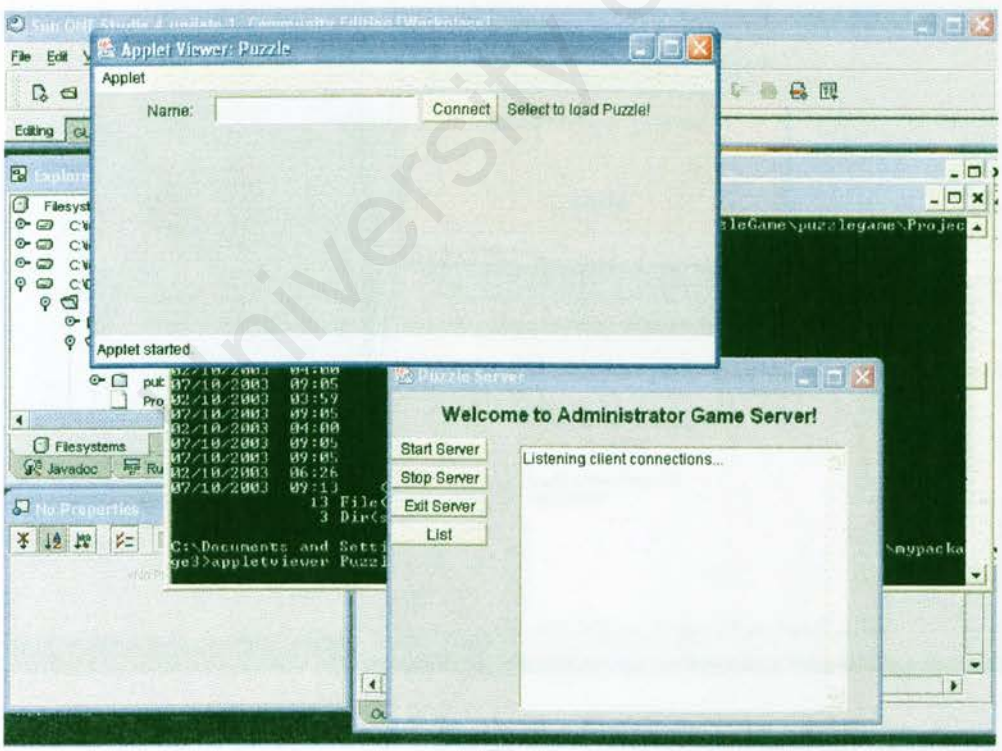
E-mail: phoenix_lynn@yahoo.com

The End

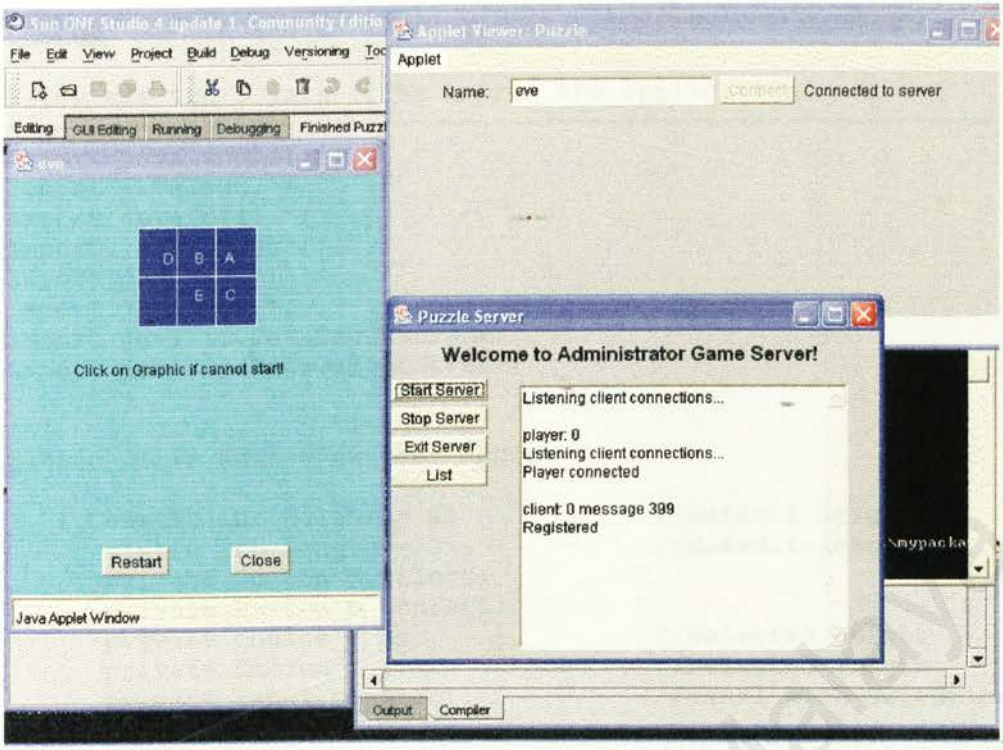
Pictures:



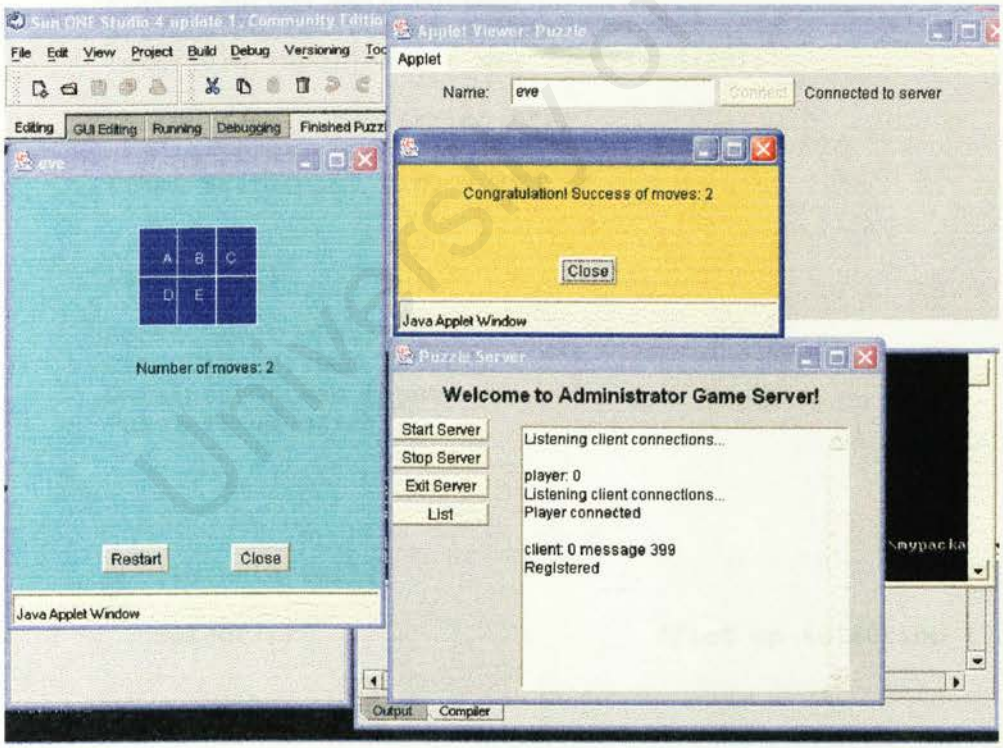
[1] Server running



[2] Applet loaded – server waiting connection



[3] Enter username – server connected – puzzle frame loaded



[4] Server connected – play puzzle – won – puzzle success loaded

Coding

Below are the 5 java files to run the application.

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import java.io.DataInputStream;
import java.io.DataOutputStream;


public class Puzzle extends Applet

   {  static int height = 2;              //default height
      static int length = 3;              //default length
      private Button b_select;
      private Button b_connect;
      private Choice c_h;                 //selected height
      private Choice c_l;                 //selected length
      private Panel p_select;             //panel with selection menu;
gets
                                          //replaced with...
      private Panel p_resume;             //after selection has been made
      static Button restart_button;
      static Button quit_button;
      private Button resume_button;
      TextField input_name = new TextField(20);

      String nama = new String("");

      final static int PING = 299;


      Label lab_1;

      Socket sock = null;
      DataOutputStream output;
      DataInputStream input;
      PuzzleFrame main;



      public void init()
      {
         Select();                        //set up selection frame

       }


      public void Select()
      {  int h, l;
         c_h = new Choice();              //selected height
```

```
    c_l = new Choice();                 //selected length
    b_select = new Button("  OK  ");
    b_select.setEnabled(false);
    b_connect = new Button("Connect");

    Panel pUpper = new Panel();
    Panel pLower = new Panel();
    Panel pCenter = new Panel();
    Panel pLeft = new Panel();
    Label name = new Label("Name:");

    lab_1 = new Label("Select to load Puzzle!");

    this.setBackground(Color.lightGray);
    this.setLayout(new BorderLayout());

    pUpper.add(name);
    pUpper.add(input_name);
    pUpper.add(b_connect);
    pUpper.add(lab_1);
    add("North",pUpper);
    nama = input_name.getText();

    main = new PuzzleFrame();

}


public boolean action(Event e, Object arg)
{
 /*
    if (e.target == c_h)            //height has been set
    {  String tt;
       tt = (String)e.arg;
       height = java.lang.Integer.parseInt(tt.substring(2,3));
       return true;
    }

    if (e.target == c_l)            //length has been set
    {  String tt;
       tt = (String)e.arg;
       length = java.lang.Integer.parseInt(tt.substring(2,3));
       return true;
    }

    if (e.target == b_select)       //selection process completed
    {  p_select.hide();
       this.remove(p_select);       //remove selection-panel

       this.add(p_resume);          //add resume-panel

       resume_button.show();
       p_resume.show();
       this.show();

       resume_button.validate();
```

```
            p_resume.validate();
            this.validate();

           // f = new PuzzleFrame("Welcome to Puzzle Frame!");



            return true;
         }

         if (e.target == resume_button) //new game!
         {  p_resume.hide();            //remove resume-panel
            this.remove(p_resume);
            p_select.show();
            this.add(p_select);         //replace with selection panel
            p_select.show();
            this.show();                //start all over again!
          }*/

        if(e.target == b_connect)
        {
        try{
          b_connect.setEnabled(false);
          sock = new Socket("localhost",5050);
          output = new DataOutputStream( sock.getOutputStream() );
          input = new DataInputStream( sock.getInputStream() );
          lab_1.setText("Connected to server");

          main.login(this,sock,"localhost");
          PuzzlePanel pp = new PuzzlePanel(length, height, main);
          main.setLayout(new BorderLayout(15, 15));
          main.add("North", pp);
          main.pack();
          main.resize(305,400);
          main.show();


        }catch(Exception ex){
        b_connect.setEnabled(true);
        lab_1.setText("Check Server!Try again");}
        }

         return super.action(e, arg);
    }


    public boolean handleEvent(Event e)
    {
        return super.handleEvent(e);
    }


}
```

```
import java.lang.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.event.WindowListener;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.Vector;


public class PuzzleServer extends Frame {



private ServerSocket server;
final int port = 5050;
int MAXPLAYER = 5;

Player[] players = new Player[MAXPLAYER];
 private int numOfPlayers =0;

//private List client_list;
public TextArea view = new TextArea(13,35);
Font font = new Font("Garamond", Font.BOLD, 16);
Label client_in = new Label("Welcome to Administrator Game Server!");
Button StopServer = new Button("Stop Server");
Button StartServer = new Button("Start Server");
Button ExitServer = new Button("Exit Server");
Button ClientList = new Button("List");

final static int PING = 299;

public PuzzleServer()
  {

    super("Puzzle Server");

     try {

              server = new ServerSocket(port);

      }
      catch(Exception ex){ }
      /* processing window events: */
        WindowListener L= new WindowAdapter () {
        public void windowClosing(WindowEvent e) {
        System.exit(1);
        /* closeApplication(); */
            }
        };
        addWindowListener(L);
```

```
    this.setSize(400,300);
    this.setLayout(new BorderLayout());
    this.setBackground(Color.lightGray);

    Panel upPanel = new Panel();
    Panel Middle = new Panel();
    Panel buttonPanel;
    buttonPanel = new Panel(new GridLayout(10,1,5,5));

    //StopServer.addActionListener(this);
    //StartServer.addActionListener(this);

    buttonPanel.add(StartServer);
    buttonPanel.add(StopServer);
    buttonPanel.add(ExitServer);
    buttonPanel.add(ClientList);
    add("West", buttonPanel);

    Middle.add(view);
    upPanel.add(client_in);
    client_in.setFont(font);


    add("Center",Middle);
    add("North",upPanel);
    this.show();
    execute();

}

public void execute()
{
 for (int i=0; i < players.length; i++)
 {
  try
  {
    view.append("Listening client connections...\n");
    players[i] = new Player(server.accept(), this, i);
    view.append("\nplayer: " + i + "\n");
    players[i].start();
    ++numOfPlayers;
  }
  catch(Exception e){}
 }

}

public void actionPerformed(ActionEvent e){

  Object src=e.getSource();
  String pemain = new String("");


  if (src == StartServer)
      {
```

```
                //msgFrame.show();
                //player_msg = to_player.getText();
                //msgsToPlayers(player_msg);

            }
        if(src == StopServer)
            {
              System.exit(1);
            }

        }


    public static void main(String args[]) {

                    new PuzzleServer();

        //execute();

           }


}

/*A thread class*/

class Player extends Thread
{
    Socket connection;
          DataInputStream input;
          DataOutputStream output;
          PuzzleServer control;
    int index = -1;
    String Name;

    final static int PING = 299;
    final static int REGISTER = 399;

    public Player(Socket sock, PuzzleServer puzzle, int i)
    {
      connection = sock;
      try
      {
      input = new DataInputStream(       connection.getInputStream() );
                  output = new
DataOutputStream(connection.getOutputStream() );
      }
      catch( IOException e) {e.printStackTrace();}

      control = puzzle;
      index = i;

    }


    public final void run()
```

```
  {
  final boolean done = false;
   try
   {
    control.view.append("Player" + " connected\n");
    output.writeInt(PING);
    while ( !done ) {
                int clientms = input.readInt();
          control.view.append("\n");
          control.view.append("client: " + index + " message " +
clientms);


      switch(clientms)
      {
        case REGISTER:
            Name = input.readUTF();
            control.view.append("\nRegistered " + Name);
            break;
      } //end switch
   }//end while
  }catch(IOException e){e.printStackTrace();}
  }//end run
}
/*end of thread class*/

-------------------------------------------------------------------------
-------------------------------------------------------------------------

import java.awt.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import java.net.*;


public class PuzzleFrame extends Frame implements Runnable {

   static int height;
   static int length;
   Socket accept;
   String servername;
   DataOutputStream output;
   DataInputStream input;
   Puzzle p_game;

   Thread outputThread;
   String name = new String("");

   final static int PING = 299;
   final static int REGISTER = 399;

   public void login(Puzzle puzzle, Socket receive, String host)
   {
     accept = receive;
     servername = host;
     try
```

```
    {
      output = new DataOutputStream(accept.getOutputStream());
      input = new DataInputStream(accept.getInputStream());
    }catch( IOException e) {e.printStackTrace();}

    p_game = puzzle;
    mainframe();
    outputThread =new Thread(this);
    outputThread.start();
  }

  public void mainframe()
  {

  String name = new String("");
  name = p_game.input_name.getText();
  setTitle(name);

  }

  public boolean handleEvent(Event e)
  {     if (e.id == Event.WINDOW_DESTROY)
        {   this.hide();
            this.dispose();
            return true;
        }


        return super.handleEvent(e);
    }


  public void run()
  {
    try{
    int message = input.readInt();
    switch(message)
    {
      case PING:
        output.writeInt(REGISTER);
        output.writeUTF(name);
    }
   }catch(IOException e){}
  }

}
-------------------------------------------------------------------------
-------------------------------------------------------------------------

import java.awt.*;
import java.util.*;
import java.lang.*;
import java.io.*;

public class PuzzlePanel extends Panel  {

   static int height;
```

```
   static int length;
   static int height_max = 6;              //of puzzle
   static int length_max = 9;              //of puzzle
   static int tot_x;                       //horizontal dim of window
   static int tot_y;                       //vertical dim of window
   static int center_x_extra;              //extra spacing
   static final int rect = 10;             //indent spacing
   static final int extra_vert = 20;       //extra spacing
   static int counter;                     //counts puzzle moves
   //actual state of puzzle
   static char state[] = new char[height_max*length_max+1];
   //possible labels for selected dimension of puzzle
   static char letter[][] = new char[height_max*length_max+1][2+1];
   //all possible labels for puzzle elements
   static char letter_hilf[] =
{'x','A','B','C','D','E','F','G','H','I','J','K','L','M','N', 'O', 'P',
'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd',
'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '*', '@'};
   static int blank;                       //position of empty element
   private Button restart_button;
   private Button close_button;
   static boolean success = false;         //T: puzzle finished; F: not
yet
   static Frame fpp;                       //this.frame


   public PuzzlePanel(int ll, int hh, Frame frame)
   {
      super();

      fpp = frame;

      this.setBackground(Color.cyan);

      height = hh;
      length = ll;

      tot_x = length * 25 + 1 * rect;
      tot_y = height * 30 + 2 * rect;
      center_x_extra = Math.round((9 - length) * 12);


      //initialize puzzle
      //1st: set possible element labels
      for(int i=1; i <= height*length; i++)
      {  this.letter[i][1] = this.letter_hilf[i];
         this.letter[i][2] = this.letter_hilf[i];
      }
      int i = 1;
      //2nd: use random number generator to fill state variable
      while(i <= height*length)
      {  int h;
         Random hr = new Random();
         h = hr.nextInt();
         h = Math.abs((h % (height*length) )) + 1;
```

```
            if (this.letter[h][1] != ' ')
            {  this.state[i] = this.letter[h][1];
               this.letter[h][1] = ' ';
               if (h == height*length)
               {  this.state[i] = ' ';
                  this.blank = i;
               }
               i++;
            }
         }

         this.letter[length*height][2] = ' ';
         counter = 0;

         this.setLayout(new FlowLayout(FlowLayout.CENTER, 50, 300));

         //set up button elements in window
         restart_button = new Button("Restart");
         restart_button.setBackground(Color.lightGray);
         this.add("North", restart_button);
         close_button = new Button("Close");
         close_button.setBackground(Color.lightGray);
         this.add("North", close_button);
         return;
      }


      //test if puzzle done
      public boolean finished()
      {  boolean ret = true;
         for(int i=1; i <= ((height * length) - 1); i++)
         {  if (state[i] != letter_hilf[i])
            {  ret = false;
            }
         }
         return ret;
      }


      //clears rectangle, draws blue rect, draws white lines
      static void _paint(char[] state, Graphics g)
      {  int x_disp;
         int y_disp;
         int tot_tot_x = tot_x + extra_vert - 10;
         int tot_tot_y = tot_y;
         x_disp = Math.round((tot_x + extra_vert - 10) / length);
         y_disp = Math.round(tot_y / height);

         g.setColor(Color.blue);
         g.clearRect(50 + center_x_extra - rect - 10, 30 + rect,
tot_tot_x, tot_tot_y);
         g.fillRect(50 + center_x_extra - rect - 10, 30 + rect, tot_tot_x,
tot_tot_y);
         g.setColor(Color.white);
         //draws horizontal lines --------
         for(int i=0; i < height; i++)
```

```
      {  g.drawLine(50 + center_x_extra - rect - 10, 30 + rect + i *
y_disp,(50 + center_x_extra - rect - 10) + tot_tot_x, 30 + rect + i *
y_disp);
      }
      //draws vertical lines | | |
      for(int j=0; j < length; j++)
      { g.drawLine(50 + center_x_extra - rect - 10 + j * x_disp, 30 +
rect,50 + center_x_extra - rect - 10 + j * x_disp, 30 + rect +
tot_tot_y);
      }
      //last lines exactly on border
      //draws horizontal lines----
      g.drawLine(50 + center_x_extra - rect - 10, 30 + rect +
tot_tot_y, 50 + center_x_extra - rect - 10 + tot_tot_x, 30 + rect +
tot_tot_y);
      //draws vertical lines | | |
      g.drawLine(50 + center_x_extra - rect - 10 + tot_tot_x, 30 +
rect,50 + center_x_extra - rect - 10 + tot_tot_x, 30 + rect +
tot_tot_y);

      return;
   }


   //draws counter
   static void _drawchar(int counter, Graphics g)
   {  String sCounter;
      sCounter = java.lang.Integer.toString(counter);
      g.setColor(Color.black);
      g.clearRect(50 - rect -10, height*30 + 30 + 2 * extra_vert,
length_max * 25 + rect + 10 + 1000, 40);
      if (counter == 0)
      {  g.drawString("Click on Graphic if cannot start!", 50,
height*30 + 60 + 2 * extra_vert);
      }
      else
      {  g.drawString("Number of moves: " + sCounter, 100, height*30 +
60 + 2 * extra_vert);
      }

      return;
   }

   //main function: does everything - rectangle and counter
   public void _paint_all(Graphics g)
   {  _paint(state, g);
      g.setColor(Color.white);
      for(int i=1; i <= height; i++)
      for(int j=1; j <= length; j++)
      {  g.drawChars(state, (i-1)*length+j, 1, 50 + center_x_extra + 25
* (j-1), i*30 + 2 * extra_vert);
      }
      _drawchar(counter, g);
      return;
   }
```

```
   public void paint(Graphics g)
   {  if (counter >= 0)
      {  _paint_all(g);
      }
      return;
   }


   public void update(Graphics g)
   {  paint(g);
      return;
   }


   public boolean handleEvent(Event e)
   {  if ((e.target == restart_button) & (e.id == e.ACTION_EVENT))
      {  success = false;                    //looser!
         for(int i=1; i <= height*length; i++)
         {  this.letter[i][1] = this.letter_hilf[i];
            this.letter[i][2] = this.letter_hilf[i];
         }
         counter = 0;
         int i = 1;
         //set up everything for new game
         while(i <= height*length)
         { int h;
           Random hr = new Random();
           h = hr.nextInt();
           h = Math.abs((h % (height*length) )) + 1;
           if (this.letter[h][1] != ' ')
           {  this.state[i] = this.letter[h][1];
              this.letter[h][1] = ' ';
              if (h == height*length)
              {  this.state[i] = ' ';
                 this.blank = i;
              }
              i++;
           }

         }

         this.letter[length*height][2] = ' ';

         Graphics g = this.getGraphics();
         _paint(state, g);

         g.setColor(Color.white);
         for(i=1; i <= height; i++)
         for(int j=1; j <= length; j++)
         {  g.drawChars(state, (i-1)*length+j, 1, 50 + center_x_extra +
25 * (j-1), i*30 +  2 * extra_vert);
         }
         _drawchar(counter, g);
         //update(g);              --not necessary!
         this.validate();                    //better!
```

```
        //this isn't nice
        Dimension you_are_so_hot = new Dimension();
        you_are_so_hot = getParent().size();
        int h;
        Random hr = new Random();
        h = hr.nextInt();
        h = Math.abs( h % 2 );
        if ((h == 0))
        { h = -1;
        }
        //so h = -1 or 1; randomly make frame a triffle bigger or
smaller
        getParent().resize(you_are_so_hot.width + 0,
you_are_so_hot.height + h);

    }

    //do what you're supposed to do after arrow-key pressed
    if (e.id == Event.KEY_ACTION)
    {  if (e.key == Event.DOWN)
       {  if (blank >= (length + 1))
          {  state[blank] = state[blank - length];
             state[blank - length] = ' ';
             counter++;
             blank = blank - length;
             Graphics g = this.getGraphics();
             _paint(state, g);
          }
       }
       if (e.key == Event.UP)
       {  if (blank <= ((height - 1) * length))
          {  state[blank] = state[blank + length];
             state[blank + length] = ' ';
             counter++;
             blank = blank + length;
             Graphics g = this.getGraphics();
             _paint(state, g);
          }
       }
       if (e.key == Event.LEFT)
       {  if ((blank % length) != 0)
          {  state[blank] = state[blank + 1];
             state[blank + 1] = ' ';
             counter++;
             blank = blank + 1;
             Graphics g = this.getGraphics();
             _paint(state, g);
          }
       }
       if (e.key == Event.RIGHT)
       {  if ((blank % length) != 1)
          {  state[blank] = state[blank - 1];
             state[blank - 1] = ' ';
             counter++;
             blank = blank - 1;
             Graphics g = this.getGraphics();
             _paint(state, g);
```

```
            }
         }
      }
      //redraw everything nicely if key has been pressed
      if (e.id == Event.KEY_ACTION)
      {  Graphics g = getGraphics();
         g.setColor(Color.white);
         for(int i=1; i <= height; i++)
         for(int j=1; j <= length; j++)
         {  g.drawChars(state, (i-1)*length+j, 1, 50 + center_x_extra +
25 * (j-1), i*30 +  2 * extra_vert);
         }
         _drawchar(counter, g);
      }

      //done - finally!
      //display success window
      if ((finished() == true) & (success == false))
      {  success = true;
         Frame fs = new PuzzleFrame();
         PuzzlePanel_Success pps = new PuzzlePanel_Success(counter,
fs);

         GridBagLayout gbls = new GridBagLayout();
         fs.setLayout(gbls);

         GridBagConstraints gbcs = new GridBagConstraints();
         gbcs.weightx=1.0;
         gbcs.weighty=1.0;
         gbcs.gridx=0;
         gbcs.gridy=0;
         gbcs.anchor = GridBagConstraints.NORTH;
         gbcs.fill = GridBagConstraints.BOTH;
         gbcs.gridwidth = GridBagConstraints.REMAINDER;
         gbls.setConstraints(pps, gbcs);
         fs.add(pps);

         fs.pack();
         fs.resize(140,140);
         fs.show();
      }

      if ((e.target == close_button) & (e.id == e.ACTION_EVENT))
      {  fpp.hide();
         fpp.dispose();
         return true;
      }


      return super.handleEvent(e);
   }


}
```

```
---------------------------------------------------------------------
import java.awt.*;
import java.util.*;
import java.lang.*;
import java.io.*;


public class PuzzlePanel_Success extends Panel
{

   private Button close_button;
   private Label ls;                    //how long it took
   private Frame f;

   public PuzzlePanel_Success(int counter, Frame fs)
   {
         super();

         f = fs;
         String sCounter;
         sCounter = java.lang.Integer.toString(counter);
         ls = new Label("Congratulation! Success of moves: " +
sCounter, Label.CENTER);

         close_button = new Button("Close");
         close_button.setBackground(Color.lightGray);

         this.setBackground(Color.orange);

         GridBagLayout gblps = new GridBagLayout();
         this.setLayout(gblps);

         GridBagConstraints gbcps = new GridBagConstraints();
         gbcps.gridx=0;
         gbcps.gridy=0;
         gbcps.anchor = GridBagConstraints.NORTH;
         gbcps.fill = GridBagConstraints.BOTH;
         gblps.setConstraints(ls, gbcps);
         this.add(ls);

         gbcps.gridx=0;
         gbcps.gridy=GridBagConstraints.RELATIVE;
         gbcps.anchor = GridBagConstraints.SOUTH;
         gbcps.fill = GridBagConstraints.NONE;
         gbcps.gridwidth = GridBagConstraints.REMAINDER;
         Insets yada_yada_yada = new Insets(40, 0, 0, 0);
         gbcps.insets = yada_yada_yada;
         gblps.setConstraints(close_button, gbcps);
         this.add(close_button);

   }

   public boolean handleEvent(Event e)
   {

         if ((e.target == close_button) & (e.id == e.ACTION_EVENT))
         {  f.hide();
```

```
            f.dispose();
            return true;
        }

        if (e.id == e.WINDOW_DESTROY)
        {   f.hide();
            f.dispose();
            return true;
        }

        return super.handleEvent(e);
    }
}
```

References

Colouris, G., Dollimore, .J. and Kindberg, I. (Eds) (2001). *Distributed Systems, Concepts and Design,* 3rd ed, Addison Wesley. (DSCD)

Roberta M. Roth (2000). Alan Dennis and Barbara Haley Wixom John Wiley & Sons, Inc. *System Analysis and Design, 2nd ed.*

Roger A. Lurie (2003). *A Recipe for Success: Departmental Client Server Application Development.* Arizona State University.

P. Sellapan (2000). *Database Management. Theory and Practice.* Sejana Publishing, 2nd ed.

Halvorson, M. (2003). *Microsoft Visual Basic 6.0 Professional step by step. 2nd* ed. Microsoft Press publisher.

Halvorson, M. (1998). *Microsoft Visual Basic 6.0 Step by step: Quick and self-paced Training in Microsoft Visual 6.0.* Microsoft Press publisher.

Pfleeger, L. 5. (2001). *Software engineering Theory and Practical. 2nd* ed. Prentice Hall publisher.

H. M. Dietel, P. J. Dietel (2002). *Java: How to Program.* Prentice Hall, 4th ed.

H. M. Dietel, P. J. Dietel, S. E Santry (2002). *Advanced Java 2 Platlbrm: How to Program*. Prentice Hall, 1st ed.

Andrew S. Tanenbaum (2003). Computer Networks. Pearson Education Inc, 4th ed.

DATANET. (2002). *Internet For Business.*

URL: http ://www.Datanet.uk/terrninology.htm

Sliders Puzzle for Java.

URL: http ://www.mazeworks .com/sliders/index.htm

URL: http ://www.ecst.csuchico.edu/~tfsmiles/java/server/

Bibliography

Niu Swee Ling (1999/ 2000). Thesis report: *Digital Network Learning Server (Digital Network Learning Environment).*

Mani Subramaniam (2000). *Network Management: Principles and Practice.* Addison Wesley. (29-04-2003).

URL: http://computer.howstuffworks.comlau2mented-reality.htm/ (21-04-2003)

URL: http://www.unix. ualberta.ca/ADSM/admgde/a45eaaO3 .htrn (30-04-2003)

URL: http://www benchmarkresources.com/database_vendors.htinl (30-04-2003)