







In the name of ALLAH,  
The Most Beneficent,  
The Most Merciful.

**Department of Computer Science,  
International Islamic University, Islamabad.**

June 30, 2004

**Final Approval**

It is certified that we have read the thesis; entitled “Code Mobility (Runtime Process Migration)” submitted by **Muhammad Kamran Naseem** University Reg. No. **19-CS/MS-01**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad, for the Degree of **MS (Computer)**.

**Committee**

**External Examiner**

Dr. Abdul Sattar  
Consultant  
Allama Iqbal Open University,  
Islamabad.

  
\_\_\_\_\_

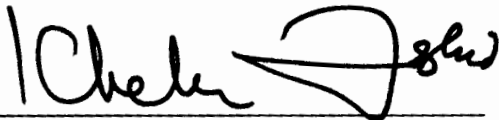
**Internal Examiner**

Asim Munir  
Lecturer,  
Department of Computer Science,  
International Islamic University,  
Islamabad.

  
\_\_\_\_\_

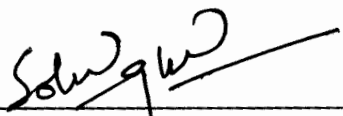
**Supervisors**

Prof. Dr. Khalid Rashid  
Dean,  
Faculty of Applied Sciences,  
International Islamic University,  
Islamabad.

  
\_\_\_\_\_

**Sohail Iqbal Ayubi**

Lecturer,  
Department of Computer Science,  
International Islamic University,  
Islamabad.

  
\_\_\_\_\_

A dissertation submitted to the  
Department of Computer Science,  
International Islamic University, Islamabad  
as a partial fulfillment of the requirements  
for the award of the degree of  
**MS (Computer)**

## **Declaration**

I hereby declare that this software, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that I have developed this software entirely on the basis of my personal efforts made under the sincere guidance of our teachers. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Muhammad Kamran Naseem  
19-CS/MS-01

## **Dedication**

To Muslim Ummah, whose dynasty's pendulum,  
I wish to see, again marking the peak.  
To my parents who pray for me unconditionally, and  
To my family and friends for their enthusiastic company.

Muhammad Kamran Naseem,  
Islamabad,  
29-01-2004

## **Acknowledgements**

All praise to the Almighty Allah, the most Merciful, the most Gracious, without whose blessings, I was unable to complete the project.

To my parents for their supreme support and patience, buttressed me during my struggling period and the status today I gained, is due to them.

I would like to thank sincerely my supervisors Dr. Khalid Rashid and Mr. Sohail Iqbal Ayubi for all their help over the past number of months, their sincere efforts helped me to complete my project successfully. I would also like to thank my class fellows for their assistance and support I grasped, when required. In the end, special thanks go to Muhammad Israr of Streaming Networks, Islamabad, whose facilitation made my efforts confident enough to complete a major portion of this document.

Muhammad Kamran Naseem  
19-CS/MS-01



# Project in Brief

Project Title:	Code Mobility (Runtime Process Migration)
Objective:	To Develop a technique of process migration to boost distributed computing.
Undertaken By:	<b>Muhammad Kamran Naseem</b>
Supervised By:	<b>Prof. Dr. Khalid Rashid</b> Dean, Faculty of Applied Sciences & Faculty of Management Sciences, International Islamic University, Islamabad.  <b>Sohail Iqbal Ayubi</b> Lecturer, Department of Computer Science, International Islamic University, Islamabad.
Technologies Used:	C Microsoft® Visual C++ 6.0
System Used:	Intel Pentium® III
Operating System Used:	Microsoft® Windows® 2000 Professional Microsoft® Windows® XP Professional
Testing	Tested on TCP/IP based star network
Date Started:	15 <sup>th</sup> February, 2003
Date Completed:	1 <sup>st</sup> January, 2004

## Revision History

<b>Date</b>	<b>Changes</b>	<b>Recommended by</b>
17-02-2004	Grammatical & Critical Errors	Dr. Khalid Rashid
09-03-2004	Technical Errors in Chapter 2	Dr. Khalid Rashid
20-03-2004	Finalizing of Chapter 2 & 6	Dr. Khalid Rashid
30-06-2004	Inclusion of Test Result Tables and Charts.	Dr. Abdul Sattar

## **Abstract**

Traditional architectures, technologies and methodologies used to develop large scale distributed applications reveal diversity of limitations and drawbacks in configurability, scalability and customizability. Only Java Applets provide code mobility for common platforms. Even then the code mobility provided through Java applets is the weaker one. This thesis describes the idea of implementing strong code mobility in terms of platform independence, microprocessor architecture reliance and resource management. Our proposed system establishes a shared connection with the resources and its surrounding environment based on distributed structured XML-based knowledge. The resources managed by the process are transparently shared between the nodes, so that the developer can program in a centralized setting. The goal is to present a solution for strong mobility for commonly used platforms. The Research work presents a conceptual framework for understanding code mobility. The framework is centered on a classification that introduces three dimensions: technologies, design paradigms, and applications. The contribution of the research work is multi fold. First, it provides a set of terms and concepts to understand the code mobility of Type-III (introduced in this thesis). Second, it introduces criteria and strategy that support the developer in implementing the strong code mobility. Thirdly, the issues that must be addressed for making code mobility embedded part of operating systems and developing environment. Fourthly, the identification of the classes of applications has been enlightened for the developers to recognize the importance of code mobility, also helping them in designing their applications about mobile code, and, finally, the discovered mobility Type-III has been exemplified through a simulation.

## **Abbreviations**

<b>CSPEC</b>	Control Specification
<b>DFD</b>	Data Flow diagram
<b>ERD</b>	Entity Relationship diagram
<b>OS</b>	Operating System
<b>TMEFA</b>	Type-III Mobility Enable File Search Application
<b>N/W</b>	Network
<b>CS</b>	Client-Server
<b>REV</b>	Remote Evaluation
<b>COD</b>	Code On Demand
<b>CM</b>	Code Mobility
<b>JVM</b>	Java Virtual Machine
<b>PM</b>	Process Migration
<b>LB</b>	Load Balancing
<b>MCL</b>	Mobile Code Language
<b>MCA</b>	Mobile Code Applications
<b>EU</b>	Execution Unit
<b>CE</b>	Computational Environment

## TABLE OF CONTENTS

Ch. No.	Contents	Page No.
	FINAL APPROVAL .....	iii
	DECLARATION .....	v
	DEDICATION .....	vi
	ACKNOWLEDGEMENTS .....	vii
	PROJECT IN BRIEF .....	viii
	REVISION HISTORY .....	ix
	ABSTRACT .....	x
	ABBREVIATIONS .....	xi
	LIST OF FIGURES .....	xv
<b>1.</b>	<b>INTRODUCTION.....</b>	<b>1</b>
1.1	CODE MOBILITY .....	1
1.1.1	STRONG MOBILITY .....	2
1.1.2	WEAK MOBILITY .....	2
1.2	PROCESS MIGRATION .....	3
1.3	DRAW BACKS WITH CURRENT SYSTEM .....	3
1.4	CONFUSING TERMS .....	3
1.5	NEED OF PROCESS MIGRATION.....	4
1.6	THE PROJECT.....	7
1.7	PROJECT SCOPE .....	8
1.7.1	TYPES OF MOBILE CODE .....	8
1.8	JOB TRANSFER.....	9
1.9	PLATFORM INDEPENDENT CODE MOBILITY SOLUTION.....	9
1.9.1	APPLICATIN LEVEL MECHANISM.....	9
1.10	OBJECTIVES.....	10
<b>2.</b>	<b>SYSTEM ANALYSIS .....</b>	<b>13</b>
2.1	STRUCTURED ANALYSIS .....	12
2.1.1	OBJECT DESCRIPTION.....	13
2.1.2	DATA FLOW DIAGRAM.....	15
2.1.3	PROCESS SPECIFICATIONS .....	21
2.1.3.1	START .....	21
2.1.3.2	SEARCH .....	21
2.1.3.3	MIGRATION .....	22
2.1.3.3.1	CAPTURE PROCESS STATE .....	23
2.1.3.4	REACTIVATION AT SERVER .....	23
2.1.3.5	ERROR HANDLER .....	24
2.1.4	STATE TRANSITION DIAGRAM.....	24
2.1.5	CONTROL SPECIFICATIONS .....	25
2.1.6	DATA DICTIONARY .....	25

<b>3. SYSTEM DESIGN .....</b>	<b>28</b>
3.1 RELATIONSHIP OF ANALYSIS TO DESIGN .....	29
3.2 DESIGN TYPES .....	30
3.2.1 DATA DESIGN .....	30
3.2.2 INTERFACE DESIGN .....	31
3.2.3 ARCHITECTURAL DESIGN .....	32
3.2.4 PROCEDURAL DESIGN .....	38
<b>4. SYSTEM DEVELOPMENT AND IMPLEMENTATION .....</b>	<b>45</b>
4.1 CLIENT APPLICATION AND MIGRATION HOST .....	46
4.1.1 SHARED CONNECTION .....	49
4.1.2 NON-SHARED CONNECTION .....	49
4.1.3 SHARED DISCRETE CONNECTION .....	49
4.2 WIN32 API .....	49
4.2.1 SOCKET API'S DESCRIPTION .....	50
4.2.2 FILE FUNCTIONS .....	51
4.3 FUNCTIONALITY ADDED TO SOFTWARE USING VISUAL C++ .....	51
4.3.1 SPLASH SCREEN .....	52
4.3.2 SINGLE INSTANCE OF APPLICATION .....	53
4.3.3 USER FRIENDLY INTERFACE AND FUNCTIONALITY .....	53
4.3.3.1 SEARCH/SUSPEND RESUME INTERFACE .....	53
4.3.3.2 OPTIONS INTERFACE .....	58
4.3.3.3 SERVER INTERFACE .....	62
4.3.4 SYSTEM TRAY INFORMER .....	65
4.3.5 PROPER MESSAGES .....	67
4.3.6 HELP FILE .....	68
4.3.7 FILE NAME AND PATH SELECTION .....	68
4.3.8 DISABLING INTERFACE CONTROLS DURING EXECUTION .....	69
<b>5. TESTING AND RESULTS .....</b>	<b>71</b>
5.1 OBJECTIVES OF TESTING .....	72
5.2 OBJECT ORIENTED TESTING STRATEGIES .....	72
5.3 TYPES OF TESTING DONE .....	73
5.4 EVALUATION .....	74
5.5 BENCHMARKING .....	75
5.6 TYPE-III NETWORK TRAFFIC .....	76
5.7 ENHANCEMENT .....	77
<b>6. FUTURE ENHANCEMENTS .....</b>	<b>78</b>
6.1 REQUIREMENTS FROM THE OPERATING SYSTEM .....	79
6.2 PROPOSED LANGUAGE SPECIFICATIONS .....	80
6.3 FOCUSED IMPLEMENTATION SOLUTION FOR STRONG MOBILITY .....	81
6.3.1 INTEL MEMORY ADDRESSING TECHNIQUE .....	81
6.3.2 SEGMENT AND OFFSET ADDRESSING SCHEME .....	82
6.3.3 FETCHING DATA .....	83
6.3.4 MIGRATING THE FETCHED DATA .....	83
6.3.5 PROTOCOL ENCAPSULATION .....	84
6.3.6 CODE RETRIEVAL AND REACTIVATION .....	84

6.3.7 RESOURCE MANAGEMENT .....	85
6.3.8 IMPLEMENTATION STEPS OF STRONG MOBILITY .....	85
6.3.9 FINALE .....	86
<b>BIBLIOGRAPHY AND REFERENCES .....</b>	<b>87</b>
<b>APPENDIX-A (USER MANUAL) .....</b>	<b>90</b>
<b>APPENDIX-B (BASIC CONCEPTS) .....</b>	<b>103</b>
<b>APPENDIX-C (PUBLICATIONS) .....</b>	<b>110</b>

## List of Figures

<b>Figure No.</b>	<b>Caption</b>	<b>Page No.</b>
<b>Figure 1.1</b>	Code Mobility Technique	1
<b>Figure 1.2</b>	Strong Code Mobility	2
<b>Figure 1.3</b>	Service Migration	6
<b>Figure 2.1</b>	Analysis Model	14
<b>Figure 2.2</b>	Process Image Object	15
<b>Figure 2.3</b>	0 DFD for Type-III mobility enabled File Search Application	17
<b>Figure 2.4</b>	Level 1 DFD of Type-III mobility enabled File Search Application	18
<b>Figure 2.5</b>	Level 2 DFD of Process Searching	19
<b>Figure 2.6</b>	Level 2 DFD of Migration Process	20
<b>Figure 2.7</b>	Level 3 DFD of Capturing Process State	20
<b>Figure 2.8</b>	Level 0 DFD of Remote Service	21
<b>Figure 2.9</b>	Level 1 DFD of Reactivation Process at Server	21
<b>Figure 2.10</b>	State Transition Diagram	25
<b>Figure 3.1</b>	Relation of Analysis model to design model	31
<b>Figure 3.2</b>	ProcessInfo Object	32
<b>Figure 3.3</b>	Program structure	33
<b>Figure 3.4</b>	Program Structure of User Interaction process	34
<b>Figure 3.5</b>	Program structure of Search process	35
<b>Figure 3.6</b>	Program structure of Migration process	36
<b>Figure 3.7</b>	Program structure of State Capturing process	37
<b>Figure 3.8</b>	Program structure of Process Revival	38
<b>Figure 3.9</b>	Procedural Design	39
<b>Figure 3.10</b>	Procedural Design of Searching	40



<b>Figure 3.11</b>	Procedural Design of Migration	41
<b>Figure 3.12</b>	Procedural Design of Capturing Data Space	42
<b>Figure 3.13</b>	Procedural Design of Process Revival	43
<b>Figure 3.14</b>	Overall Procedural Design of Type-III Mobility, including client and Server	44
<b>Figure 5.1</b>	Comparison of Different Mobility Types	75
<b>Figure 5.2</b>	Graphical Representation of different mobility types	76

Chapter 1  
**Introduction**

# 1. INTRODUCTION

As the power of individual workstations increase, distributed systems are becoming more popular. Users have all the capabilities that are provided by their workstations, and at most times, this is sufficient. However, there are cases when the power of one workstation is not sufficient to complete all the tasks at hand. One solution to this problem is to distribute some of the tasks to an idle workstation. Studies have shown that over 65% of workstations are idle at any given time. Distributing processes over all of the workstations in the network balances the load at each machine so the overall time needed to complete tasks is reduced.

This project deals with the discovery and development of a different paradigm of distributed computing. Through the framework provided, it has been made possible for the running processes of one machine to migrate to another machine when necessary. It is a successful approach towards load balancing, fault tolerance, extension of server capabilities, avoiding distribution of state, remote device control and configuration and others as discussed in section 1.5.

The future enhancements include the implementation of strong code mobility and the extension of Type-III mobility for a platform independent process migration.

## 1.1 Code Mobility

Code mobility is the capability of software systems to dynamically reconfigure the bindings between the software components of an application and their physical locations (nodes) within a computer network (Figure 1.1)[1]. Code Mobility is of two types.

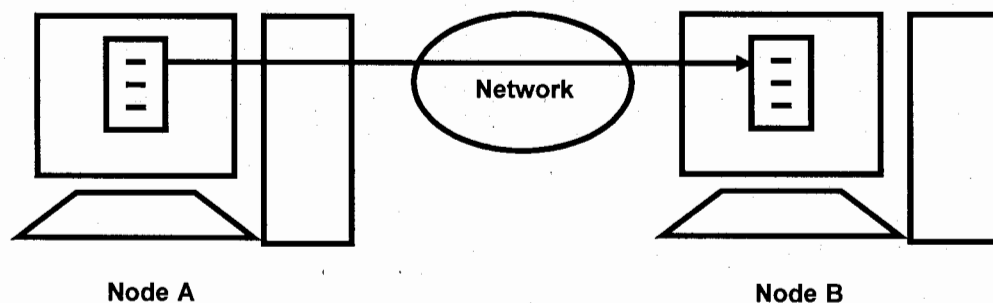


Figure 1.1 Code Mobility Systems

### 1.1.1 Strong Code Mobility

The code along with data and state of the process travels from source machine to the destination machine during execution. Figure 1.2 elaborates Strong Code Mobility.

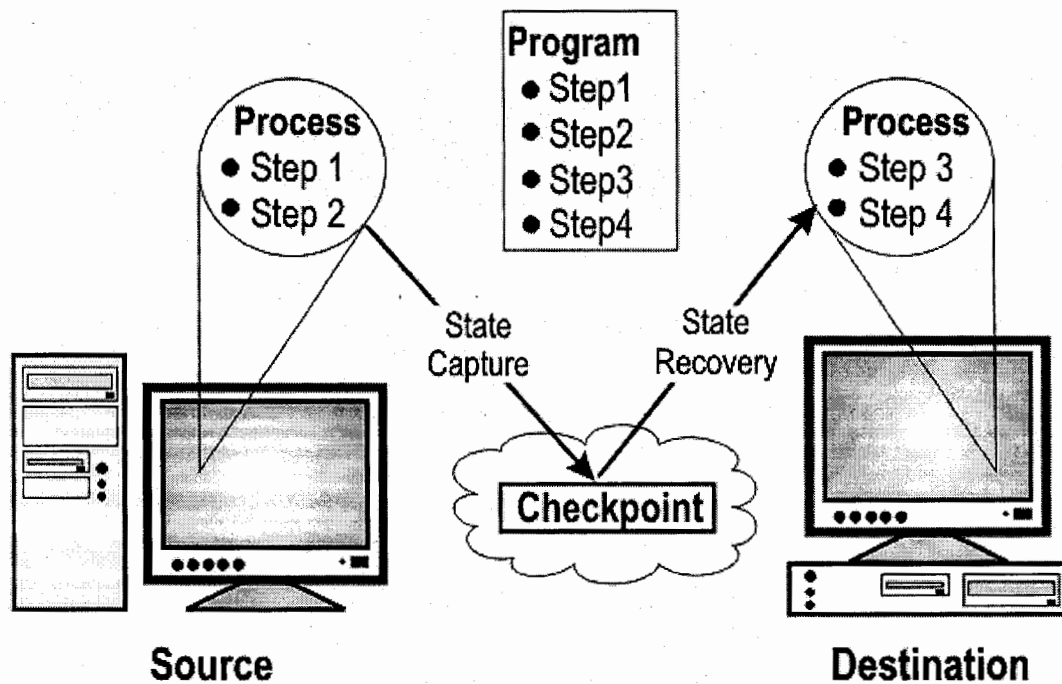


Figure 1.2 Strong Code Mobility

### 1.1.2 Weak code Mobility

Only code of the process is sent from one machine to the other and that code still has to start execution. Weak mobility is more commonly available in commercial and research systems and is the reason of

- Success of Java
- Serialization / Externalization mechanism

## 1.2 Process Migration

Process migration is the function which controls how code mobility is achieved [1]. As a result of successful process migration the suspended process on client resumes its execution on server.

## 1.3 Drawbacks with current system

The popularity of code mobility is increasing in industry and academia, however

- Applets in a Web browser are still the only pervasive application
- Research has been focusing almost completely on new technologies, thus leading to
  - Lack of quantitative evaluations and experimental evidence
- Java API has no support for accessing thread state
- Requires extension of JVM
- Loss of portability
- No specifications about how the code is to migrate
- What are the pre and post-conditions of the transfer operation?
- What are the triggers of the operation?

## 1.4 Confusing Terms

- Mobile Agents
- Agents

In the distributed system community the term “Mobile Agent” is used to denote a software component that is able to move between different execution environments. This definition has actually different interpretations. “Mobile Agents” are just code fragments associated with initialization data that can be shipped to a remote host. They do not have the ability to migrate once they have started their execution. On the other hand, in the Artificial Intelligence community the term “AGENT” denotes a software component that is able to achieve a goal by performing actions and reacting to events in a dynamic environment [1]. The behavior of this component is determined by the knowledge of the relationships among events, actions, and goals. Moreover, knowledge can be exchanged with other agents, or increased by some

inferential activity. Although mobility is not the most characterizing aspect of these entities, there is a tendency to blend this notion of intelligent agent with the one originating from distributed systems and thus assume implicitly that a mobile agent is also intelligent (and vice versa). This is actually generating confusion since there is a mix of concepts and notions that belong to two different layers, i.e., the layer providing code mobility and the one exploiting it. Finally, there is no definition or agreement about the distinguishing characteristics of languages supporting code mobility.

## **1.5 Need of Process Migration**

Code mobility represents a new way of building distributed software systems. Motivation for adopting the code mobility paradigm has been surveyed in great detail in the literature. The benefits of mobile code are appealing. We list several examples:

### **1.5.1 Real-time interaction with remote resources**

Most computing resources, like databases, file systems, or even physical displays, are not transportable. If such resources are located at a remote site, then computation that requires real-time interaction with the resources has to happen where the resources reside. Code mobility allows one to prescribe the location of computation to make real-time interaction possible. For example, active contents like Java applets prescribe interactive presentation that is to be rendered on the browser side.

### **1.5.2 Fault Tolerance**

The fault tolerance is much improved in code mobility than the conventional client-server systems. Migrating software components from their working nodes which have experienced a partial failure can improve the fault tolerance of the application system. A server program can move service code to a consumer device so that the device can be served even after it is disconnected from the network.

### **1.5.3 Reduction of Communication Traffic**

Mobile computers (e.g. hand-held computers or intelligent mobile phones) usually interact with servers through unreliable, low-bandwidth, high-latency, high-cost networks. Code mobility becomes an attractive alternative because network traffic can be reduced by migrating the client program to the server side, thus avoiding the potential cross-network communication bottlenecks.

### **1.5.4 Extension of server capabilities**

In traditional client-server applications, the server offers a predefined set of services. It is very difficult to extend the capability of the server without redefining its interface. Code mobility offers a flexible infrastructure for extensible server. Whenever new process is migrated to this server from a remote server, the capabilities of this server are extended by one. Because of this inherent connection to extensibility, many of the issues in code mobility security also appear in the study of extensible operating systems.

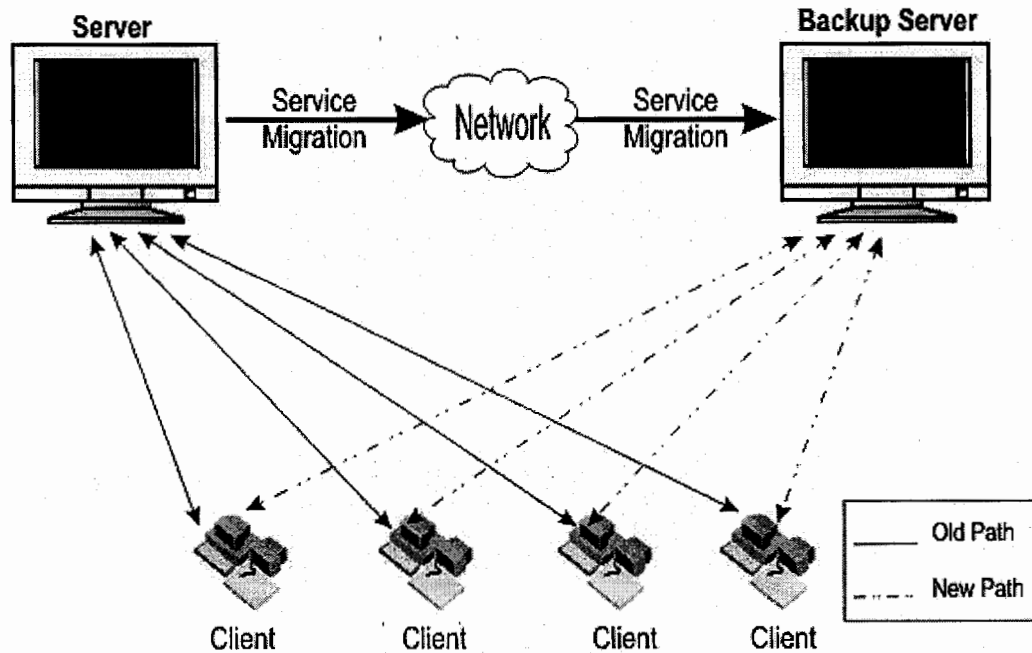
### **1.5.5 Reduction of server loads**

Traditional client-server model sometimes suffers from the fact that massive computational burden is imposed on a single server. Code mobility can transfer processing activities (those are servicing processes) from the server to the side-server/backup server. Clients wishing to access facilities of a server after that server is fully loaded, may request to side-server/backup server. This technique can address the overloading problem of servers. As shown in Figure 1.3, server programs can clone/move their services to a better execution environment to balance the load or to avoid bottlenecks of long waiting delays.

### **1.5.6 Avoiding distribution of state**

In traditional client-server applications, the state of computation is distributed among the servers and the clients. As a consequence, it is difficult to maintain consistency of the distributed states, and to express the correctness of the

computation. Code mobility localizes computation states in a single process. They offer a better abstraction that makes the crafting of distributed software a more manageable task.



**Figure 1.3** Service Migration

### 1.5.7 Protocol Encapsulation

No particular protocol is needed to achieve the code migration among distributed systems.

### 1.5.8 Immature Field

Field is still immature and lacks suitable methodologies and framework.

### 1.5.9 Service Customization

The interfaces or services are not statically defined. After a server receives process(es) from other server, its services are increased dynamically. This leads to dynamic extension in server capabilities.



### 1.5.10 More Flexibility and Up gradation

Supporting more flexible software deployment and maintenance. A server program can move service code to a consumer device so that the device can be served even after it is disconnected from the network. Automatic software up gradation without human interaction on the client side can be achieved. Code mobility can be used for on-line extension of application functionality or software upgrades. A software component can be delegated a task and sent to a remote device to perform its operation. This can be used for network management applications.

Any application that can be crafted under the mobile code paradigm can also be structured as a client-server application. However, mobile code systems offer many software engineering advantages that its client-server counterpart lacks. The above list illustrates several representative ones. Recently, *Gianpaolo* propose an abstract model for evaluating the potential benefit of adopting various mobile code paradigms. The result suggests that the advantages corrupt only for certain kinds of application domains. It is important to distinguish the implementation process and the paradigm.

## 1.6 The Project

Code Mobility under the definition of distributed computing is swiftly getting popularity. Though the idea of code mobility was delivered in 1998 by Fuggetta, Picco & Vigna, the field is still immature. No proper and suitable framework, methodology and design paradigms are available for code mobility. More over, only JAVA Applets have been introduced, which are also weak mobility accommodators. The project focuses on introducing other undefined type of code mobility, its suggested solutions and implementation along with new distributed technique, called "Job Transfer". The work done in this project provides framework, methodology and design paradigms, which open up new horizons of research in distributed computing. We implement "Type-III Mobility", which is the most straightforward, uncomplicated, economical and faster against, weak and strong mobility in terms of process selection, code fetching, transferring and re-executing. The "Type-III Mobility" framework presented in this thesis is platform independent and can be implemented for any operating system and network protocol.

## 1.7 Project Scope

Previously the work was done on weak mobility and introduction of new languages that could support code mobility. Most of the work has been done to prove that Java is the best mobility provider language, in terms of weak mobility. Many researchers consider that strong mobility is not impossible to implement but is extremely difficult to implement, because it requires greater time consumption, sound system level expertise and also involves insecurity.

“Understanding Code Mobility” by Fuggetta, Picco & Vigna,[1] the authors have introduced two basic types of code mobility, that is strong mobility and the weak mobility, but no solution of strong mobility has been provided. More than ninety percent work done under the definition of code mobility basically addresses Mobile Agents and weak code mobility(Applets), which should be defined under the heading of mobile code, not code mobility. The mechanisms defined against mobile code are defined as

### 1.7.1 Types of Mobile Code

- Mobile Agents (MA)
- Remote Evaluation (REV)
- Code on Demand (COD)
- Job Transfer (JT)
- Code Mobility (CM)

The project provides efficient process state capturing, migration and rehabilitation at the remote machine, by implementing Type-III mobility. Previously in 1998 Fuggetta, Picco & Vigna[1] have done some work on code mobility. No gigantic or immense accomplishment has been achieved during the middle era. This Project includes the following:

- Implementation solution of Strong mobility for Intel Architecture
- New mobile code mechanism i-e “Job Transfer”
- Introduction to Type-III mobility
- Implementation of Type-III mobility and Job Transfer
- Framework for Type-III code mobility implementation

## 1.8 Job Transfer

Our research has made us able to introduce this new mobility paradigm in Client/Server computing. Its details are as under.

Locations *A* and *B* both have the know-how and resources or a shared connection to the resource, but at any time *A* faces some vital problems in computations and gets unable to continue with its jobs. Therefore for a given process, it copies its Data and State, and transfers it to *B*. *B* takes the data and state, associates it with local instance of corresponding code and executes it.

## 1.9 Platform Independent Code Mobility Solution

Away from the two solutions provided for homogeneous and heterogeneous (those are kernel level and user level, refer to *Appendix-B*), we bring in a general solution that is applicable to both homogeneous and heterogeneous. This Application Level Mechanism is presented underneath. An implementation (*Chapter 3*) of this mechanism unlocks the potential of Application Level Mechanism.

### 1.9.1 Application level Mechanism

In the user level mechanism described in *Appendix-B*, wrapper routines and runtime libraries are written to achieve that mechanism. The solution we suggest under the definition of Application Level Mechanism, demonstrates that there is no need of writing runtime libraries and routines. As kernel level solution implementation is not an ordinary job (refer to *Appendix-B*), for majority of users and developers this is stumbling block. Further more critical information related to a process, is just kept with kernel or with the application itself. The only solution the user/developer is left with, are runtime libraries and routines. If runtime libraries and routines are put into action, though this is also a tough job for many programmers, every input/output, function call, parameters and return values are to be registered with mobility agent by the application or should be tracked by the runtime libraries/routines. This all becomes a complex process which itself demands more CPU cycles or causes extra loads on system. We show that as process information is just kept with kernel and the process itself, in addition kernel level process migration also has not been implemented so far, why process does not migrate itself, without any request to

kernel, and without any help of runtime libraries and wrapper routines? As we can track the process which should be migrated, we make that process enabled of migrating itself or its data space. In case of data space transfer remote node should have un-instantiated code of that process. Further more remote server should have any of these:

- The standalone resource that is being used by the migrated process at previous node
- The resource is transferred/ copied along with the data space/state
- A shared connection is established to the resource (can be called network reference)

This can not be considered as a limitation, because the server, to whom process is migrated, is obviously capable of executing this particular process. By capable we mean in terms of hardware and software. For example if a process which has been migrated, is performing a job of providing clients with their account usage details, will also provide the clients with account usage details on new side-server/ backup server. If the process can't execute at side-server, this means that either that server has no connection with the database or hardware in-capabilities of server obstruct in its job. This all has been concluded with the supposition that backup server's job is always to support the server in case of fault tolerance, load balancing and service customization, and a backup server is just like server but in idle state or with maximum CPU cycles available to use at anytime. In our project process itself copies its data space and migrates itself. The only condition we applied here is that, we just migrate process by sending data space to the remote node and the resource is already available at new node or a shared connection to that resource is available. In the next chapters we analyze, design and implement this technique, respectively.

## 1.10 Objectives

The objectives of the project are to introduce framework, design paradigms and methodologies which should be helpful in implementation and enhancement of different types of code mobility. This will ultimately provide the feasible solution available at commercial level and to everyday user to unbolt the screws of Code Mobility.

Chapter 2

**System Analysis**

## 2. SYSTEM ANALYSIS

At technical level, Software Engineering begins with a series of modeling tasks that lead to a complete specification of requirements and a comprehensive design representation for the software to be built. The Analysis model, actually a set of models, is the first technical representation of a system. Over the years many methods have been proposed for analysis modeling. However two of them now dominate the analysis modeling landscape. The first, *structured analysis* is a classical modeling method and the other approach is *object oriented* method. We have used the first modeling technique for the analysis of the software Type-III Mobility Enabled Process Migration.

### 2.1 Structured Analysis

Structured analysis is a model building activity. Using a notation that satisfies the operational analysis principles, we create models that illustrate information (data and control) contents and flow, we partition the system functionally and behaviorally, and we show the core of what must be built.

The Structured Analysis Model must achieve three primary objectives.

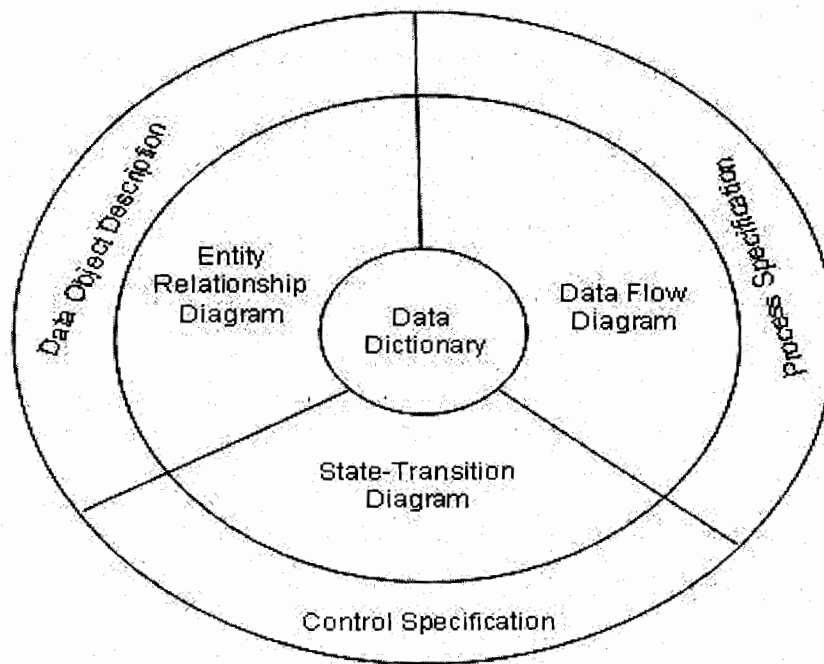
- Describe what the customer requires.
- Establish a basis for the creation of a software design.
- Define a set of requirements that can be validated once the software is built.

To accomplish these objectives, the analysis model derived during the structured analysis takes the form illustrated in Figure 2.1. At the heart of the model lies the data dictionary — a repository that contains description of all data items consumed or produced by the software. Three different diagrams surround the core. The entity-relationship diagram (ERD) depicts relationships between data objects. The ERD is the notation that is used to conduct the data modeling activity. The attributes of each data object noted in the ERD can be described using a data object description. The Data flow Diagram (DFD) provides an indication, how data are transformed as they

move through the system and depict the functions and sub functions that transform the data flow.

The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modeling of function. A description of each function presented in the DFD is contained in a process specification (PSPEC).

The State-transition diagram (STD) indicates how the system behaves at the participation of external events. To accomplish this, the STD represents the various modes of behavioral modeling. Additional information about control aspects of the software is contained in the control specification (CSPEC).



**Figure 2.1** Analysis Model

### 2.1.1 Object Description

In Object Description Section we describe the Objects. Object is a representation of almost any composite information that must be understood by the software. By composite, we mean something has a number of different attributes or properties. In

*Type-III Mobility Enabled File Search Application*, the most important object is *ProcessInfo Object*. It is used by the Application to perform the search and migration. The object is *partially opaque*. This means that we are only supposed to directly access or change certain fields in the structure. The opaque fields are similar to the private and protected members of a C++ class, and the non-opaque or accessible fields are analogous to public members. The important fields of the object are described below where non-opaque fields are shown as bold,

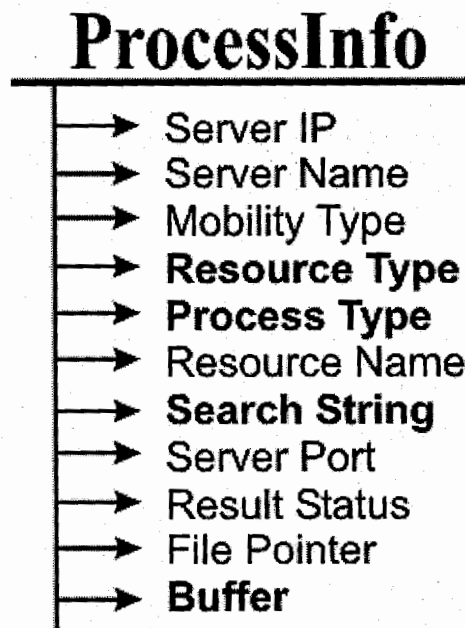


Figure 2.2 Process Image Object

- **Server IP** is the Internet protocol address of server, running the host service for process data space reception.
- **Server Name** is the Machine Name of server, running the host service for process data space reception.
- **Mobility Type** stores information to explain which type of mobility has to be performed for the current process, i-e Type-III, Strong or weak mobility.
- **Resource Type** tells the type of the resource, that is shared or standalone.
- **Process Type** notifies status of current process after mobility. That is clone or migrates. Clone means process continues its execution after



transferring its data space whereas after migration process' current instance is abandoned from memory.

- **Resource Name** stores name of the file through which search has to be performed.
- **Search String** is the string that has to be searched through file.
- **Server Port** is the server port at which data space reception server is listening for current process.
- **Results Status** is a Boolean variable which tells whether complete results, including results of this machine, are to be displayed at the destination machine. In other words, do current results also have to be dispatched along data space?
- **File Pointer** is the pointer variable which keeps the current status of the file/resource after the most recent instruction.
- **Buffer** is the information stream which consists of data space of the process after the state capturing. This buffer is dispatched to the server for re-invocation of this process at server side.

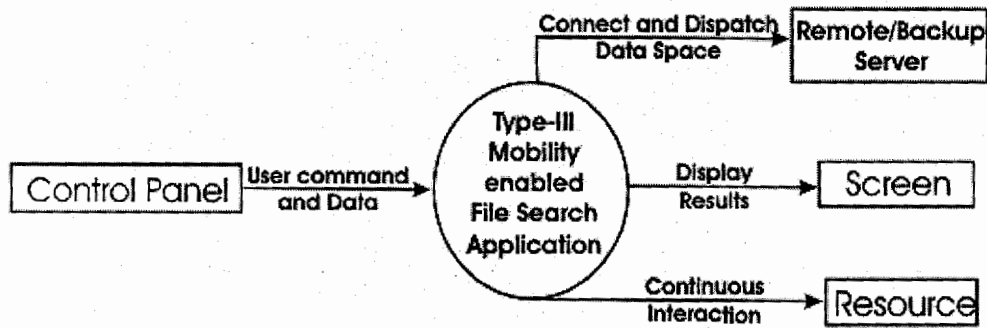
As there is only one object in our software so we can not make the ERD of the software.

### 2.1.2 Data Flow Diagrams (DFD)

As information moves through the software, it is modified by a series of transformations. A DFD is a graphical technique that depicts information flow and the transformations which are applied as data move from input to output. The DFD is also known as *Data flow graph* or a *bubble chart*.

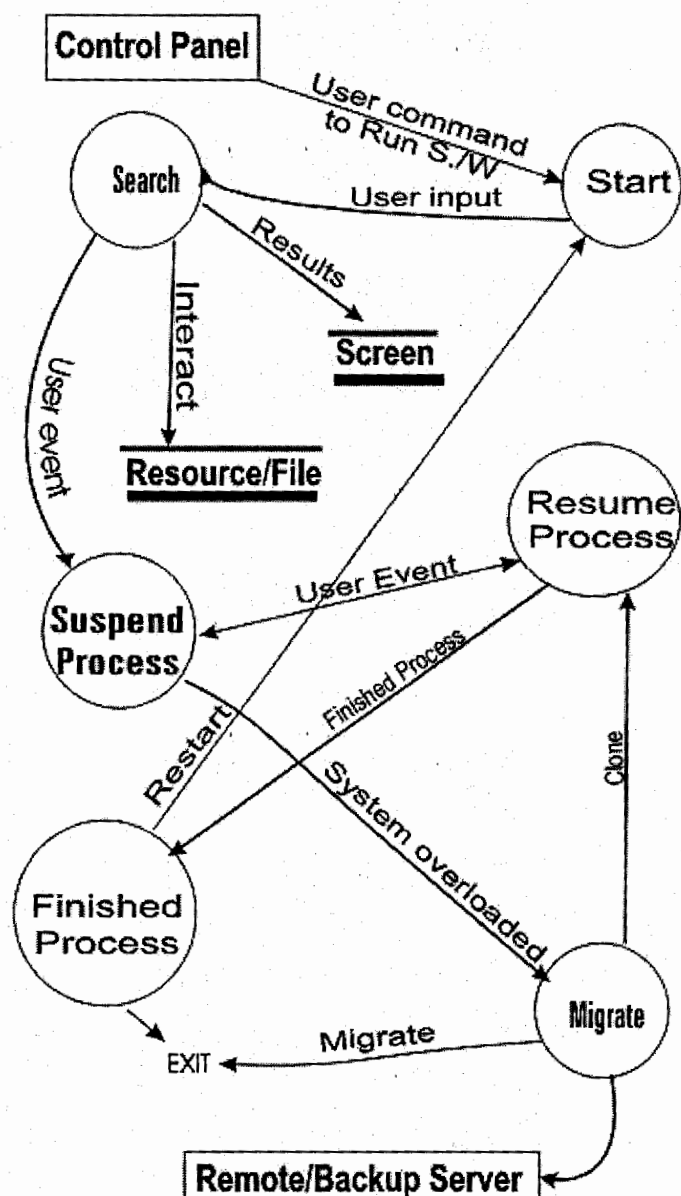
Our software is expanded up to the third level DFD and all the functions shown in the DFDs are described in PSPEC.

Figure 2.3 shows the Context level DFD for the software Type-III mobility enabled File Search Program. This level is the highest level of abstraction where no details are shown only the input to the software and output from software is shown. There is only one bubble which is the software and reveals no function of the software.



**Figure 2.3** Level 0 DFD for Type-III mobility enabled File Search Application

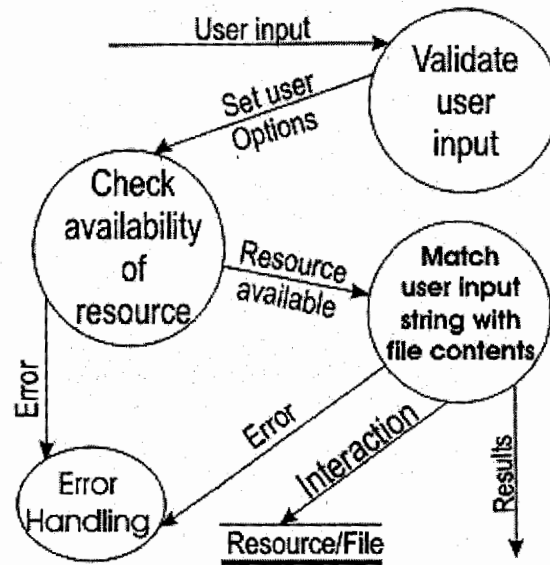
Now the DFD is expanded and level one shows the detail of the process or functions of the software. This level is called level 1 DFD for the software and reveals the function but not the sub function.



**Figure 2.4** Level 1 DFD of Type-III mobility enabled File Search Application

Figure 2.4 shows the expansion of bubbles in level 0 DFD, here the level of abstraction decreases but only up to the functions still the sub functions are not revealed. It also shows the data storage and the arrows, to indicate which process stores the data and which process uses the stored data. The Control Panel is basically the user provided instructions/commands or data, and the interface through which user interact with the software and gives command for performing different actions at runtime, Process State Capture and Migration. While the Screen is an output device,

in our case Screen is the monitor screen and shows the results, errors and different messages describing process status at present.



**Figure 2.5** Level 2 DFD of Process Searching

Level 2 DFD for Searching process is shown in Figure 2.5. This level shows the sub functions of the file searching process and describes almost all the functionality or calculation involved in searching through file.

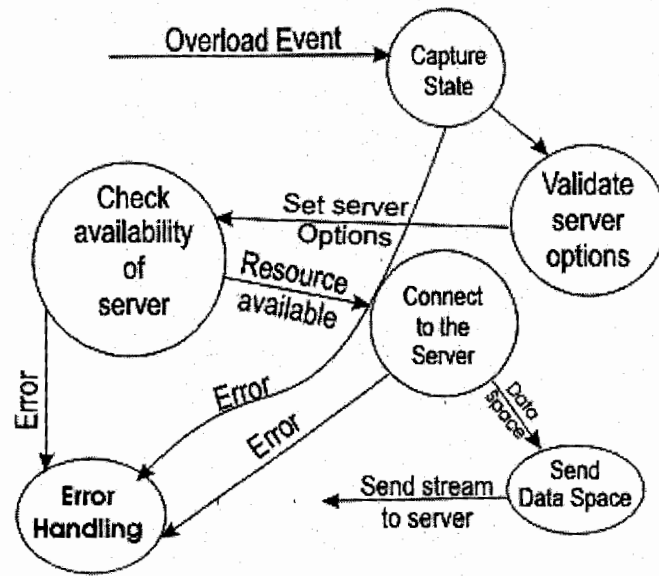


Figure 2.6 Level 2 DFD of Migration Process

Level 2 DFD of Migration process is shown in Figure 2.6. This level shows the sub functions of the migrating process and describes almost all the functionality or calculation involved in searching through. Level three of state capturing has been elaborated in Figure 2.7 to overcome any possible confusion.

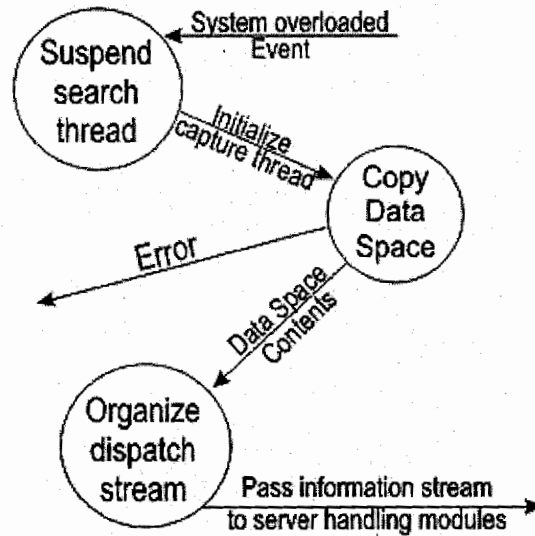
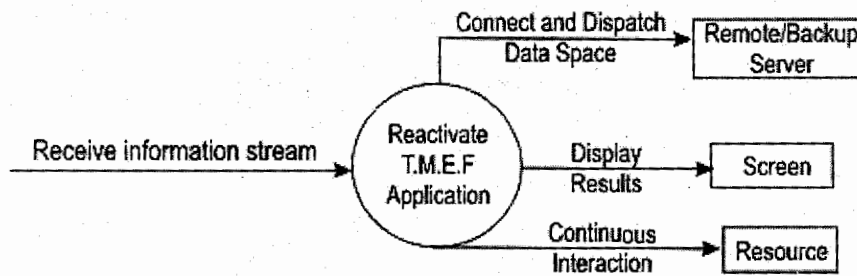


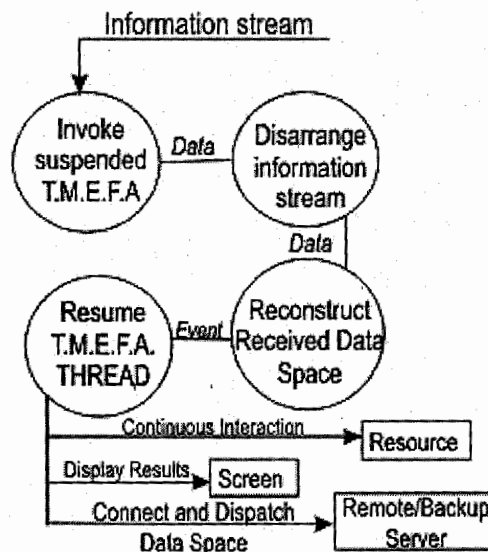
Figure 2.7 Level 3 DFD of Capturing Process State

This level shows the sub functions involved in capturing the state and describes almost whole of the functionality required. The information about the data space is stored into a stream after a successful collection of data. The important thing to note from the whole process of State Capturing is that the process of searching through file is suspended during state capturing.



**Figure 2.8** Level 0 DFD of Remote Service

The remote server's Level zero data flow diagram is represented in Figure 2.8. Though it is an abstract model, note that three outside interactions of process are same as of Level Zero of Type-III Mobility Enabled File Search Application in Figure 2.3.



**Figure 2.9** Level 1 DFD of Reactivation Process at Server

Level 0 DFD of Remote service has been expanded to Level 1. This is shown in Figure 2.9. This is the maximum expansion of functionality, the remote service

provides. This figure illustrates that migrated process can further migrate to another node, provided, that node is running the appropriate server.

### 2.1.3 Process Specification (PSPEC)

The PSPEC of our software in the form of PDL (Process definition language) are explained in section 2.1.3.1 to 2.1.3.5

#### 2.1.3.1 Start

**Procedure Start;**

    Read the input data;

**If** data is valid

**then** send user command to Search;

**Else** do not send user command to Search;

**End if;**

**Endproc**

#### 2.1.3.2 Search

**Procedure Search;**

    Get Basic Information about the search string and resource name;

**If** Basic Information is not valid **then** print error message;

**Else** Read the search string;

        Initialize resource;

        Display Process Status

**If** resource initialization failed **then** print error message;

**Else** Match string contents with file contents char by char;

**If** contents matched;

**Then** add increment char count;

**End if;**

**If** char count equals search string count;

**Then** Increment results found;

                Display the Process Status;

**End if;**

        End Displaying Process Status;

Display user required results based on display options;  
Display the completion message;  
**End if;**

**End if;**

**Endproc**

### 2.1.3.3 Migration

**Procedure Migration;**

Get Information about the Server;

Validate the Server Info;

**If** validation fails;

**Then** print error message;

**Else begin**

Suspend Search Process;

Read the Data Space;

Copy Data Space into information stream;

Check availability of server;

**If** Server not available;

**Then** print error message;

Return;

**Else** Connect to the server;

Send information stream;

**End if;**

**If** could not send;

**Then** Display Error message;

**Else** Continue;

**End if;**

**End begin;**

**End if;**

**Endproc**



### 2.1.3.3.1 Capturing Process State

**Procedure** Capture State;

Suspend the search Thread;

Initialize capture thread;

Read Data Space of Search Process;

**If** failed to copy

**Then** print error message;

**Else** Store read data space in information stream;

**End if**;

Pass information stream to server handling module;

**Endproc**

### 2.1.3.4 Reactivation at Server

**Procedure** Reactivation;

Get Information stream from client;

Validate the received Information stream;

**If** validation failed;

**Then** print error message;

**End if**;

Parse the information stream;

Invoke suspended T.M.E.F.S Application;

Call Error Handler;

Disarrange information stream;

Reconstruct data space according to information stream;

Call Error Handler;

Establish connection with resource;

Call Error Handler;

Display received results on screen;

Resume suspended T.M.E.F.S Application;

**Endproc**

### 2.1.3.5 Error Handler

**Procedure Error Handling;**

**If** error occurred during the process;

**Then** Set the required Icon;

Display Message;

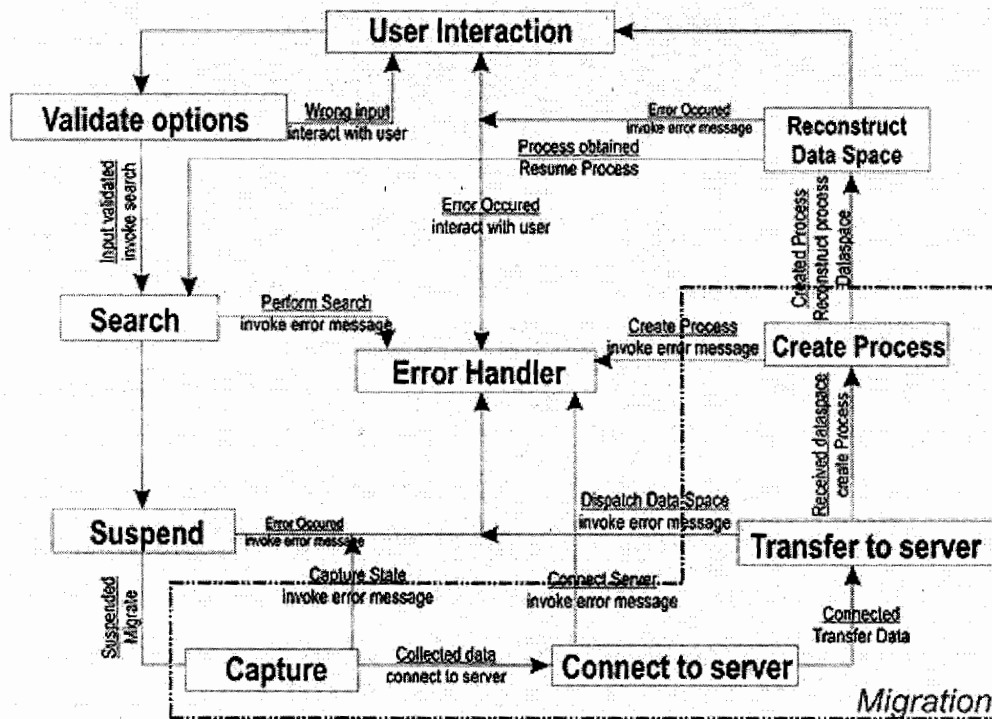
Get further command;

**End if;**

**Endproc**

### 2.1.4 State Transition Diagram (STD)

The State Transition Diagram indicates how the system behaves as consequence of external events. By studying STD, a software engineer can determine the behavior of the system and can ascertain whether there are “holes” in the specified behavior.



**Figure 2.10** State Transition Diagram

Figure 2.10 shows the State Transition Diagram for the software Type-III Mobility Enabled File Search Application.

### 2.1.5 Control Specification (CSPEC)

The Control Specification (CSPEC) represents the behavior of the system in two different ways. One is called specification behavior and the second one is the combinational specification. The CSPEC does not provide any information about the inner working of the processes that are activated as a result of this behavior.

### 2.1.6 Data Dictionary

The Data Dictionary is an organized listing of all data elements that are relevant to the system, with precise and exact definitions. The Data Dictionary of the software is:

*Name:* ServerIP  
*Aliases:* IPADDRESS  
*Where used/how used:* Connect2Server (input), Migrate (input)  
*Description:* ServerIP = "any valid IP address with four octets to locate server".

*Name:* ServerName  
*Aliases:* SERVER\_NAME  
*Where used/how used:* Connect2Server (input), Migrate (input)  
*Description:* ServerName = "any null terminated string representing the name of the server machine on network".

*Name:* MobilityType  
*Aliases:* MOBILITY\_TYPE  
*Where used/how used:* CaptureState (input), Migrate (input)  
*Description:* MobilityType = "numeric value amongst 0,1 & 2. 0 stands for weak, 1 stands for strong and 2 stands for Type-III mobility".

*Name:* ResourceType  
*Aliases:* RESOURCE\_TYPE

Where used/how used: Migrate (input)  
 Description: ResourceType = "Boolean value where 0 stands for shared resource and 1 stands for standalone".

Name: ProcessType  
 Aliases: PROCESS\_TYPE

Where used/how used: Migration(input)  
 Description: ProcessType = "Numeric value to recognize whether process has to be migrated or cloned".

Name: ResourceName  
 Aliases: FILENAME  
 Where used/how used: Search (input), CaptureState (input)  
 Description: ResourceName = "any valid null terminated path containing (optional) alphanumeric string describing resource name".

Name: SearchString  
 Aliases: SEARCH\_STRING  
 Where used/how used: Search (input), CaptureState (input)  
 Description: SearchString = "any valid alphanumeric null terminated string".

Name: ServerPort  
 Aliases: none  
 Where used/how used: Connect2Server (input)  
 Description: ServerPort = "numeric value".

Name: ResultStatus  
 Aliases: CHECK\_RESULTS  
 Where used/how used: CaptureState (input)  
 Description: ResultStatus = "Boolean variable".

Name: FilePointer

*Aliases:* none

*Where used/how used:* Search (input), CaptureState (input)

*Description:* FilePointer = "pointer variable to keep address of file/resource".

*Name:* Buffer

*Aliases:* none

*Where used/how used:* ReceiveDataSpace(input), ReconstructDataSpace(input)  
CaptureState(Output), Dispatch2Server(input),

*Description:* Buffer = "stream variable to store data space".

Chapter 3  
**System Design**

### 3. SYSTEM DESIGN

Design is an iterative process transforming requirements into a “blueprint” for constructing the software. It is the first step in the development phase for any engineered product or system. It can also be defined as “the process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization.”

The designer’s goal is to produce a model or representation of an entity that will later be built. The process by which the model is developed combines perception and judgment based on experience in building similar entities, a set of principles and/or heuristics that guide the way in which the model evolves and ultimately leads to a final design representation.

#### 3.1 Relationship of Analysis to Design

For design we need analysis’ results, which serve as base information for design. In fact we explore the analysis in detail and produce design such that it is directly mapped into coding.

Figure 3.1 shows the relation of Analysis model to Design model and the arrows shows which of the information from the analysis model is necessary for which design. Data Design is created using the Data Dictionary and Entity-Relationship Diagram information of Analysis model. Architectural Design is created using the information from Data Flow Diagram of the Analysis model. Interface Design is also created using the information from data flow diagram. Procedural Design uses the information from CSPEC, PSPEC and state-transition diagram of the Analysis model.

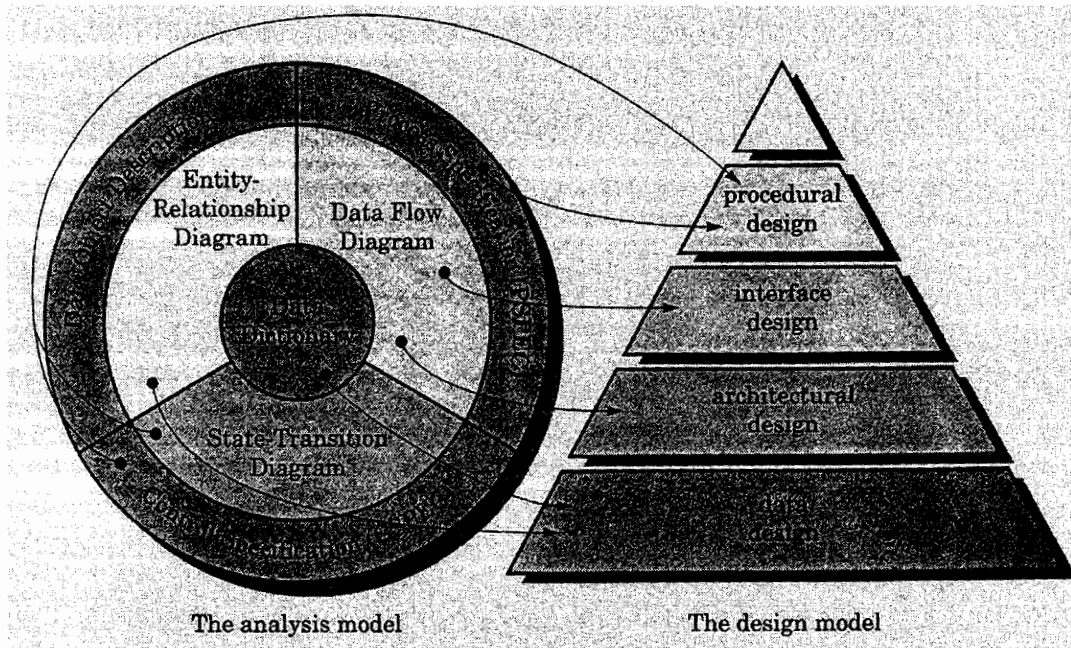


Figure 3.1 Relation of Analysis model to design model

## 3.2 Design Types

There are four types of Designs.

- Data Design
- Interface Design
- Architectural Design
- Procedural Design

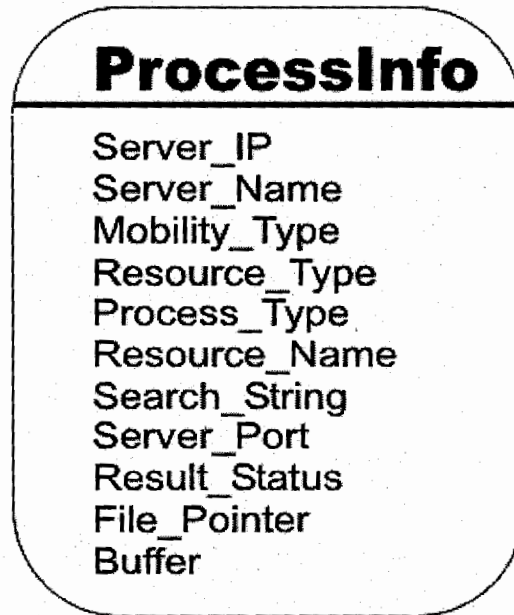
### 3.2.1 Data Design

The Data Design transform the information domain model created during analysis into the data structures that will be required to implement the software. The data objects and relationships defined in the entity-relationship diagram and the detailed data content depicted in the data dictionary provide the basis for the data design.

Data Design is the first of four design activities that are conducted during software engineering. The primary activity during data design is to select logical representation of data objects identified during the requirement definition and specification phase. The selection process may involve algorithmic analysis of alternative structures in



order to determine the most efficient design or may simply involve the use of a set module that provide the desired operations upon some representation of an object.



**Figure 3.2** ProcessInfo Object

As our software contains only one object because it is not a database project and contains information only about the process and server so the data design is very simple and holds information about only one object named ProcessInfo.

Figure 3.2 shows the Object ProcessInfo along with its attributes. This information is not stored in a database or in a file. This is the user input data at the start of process execution.

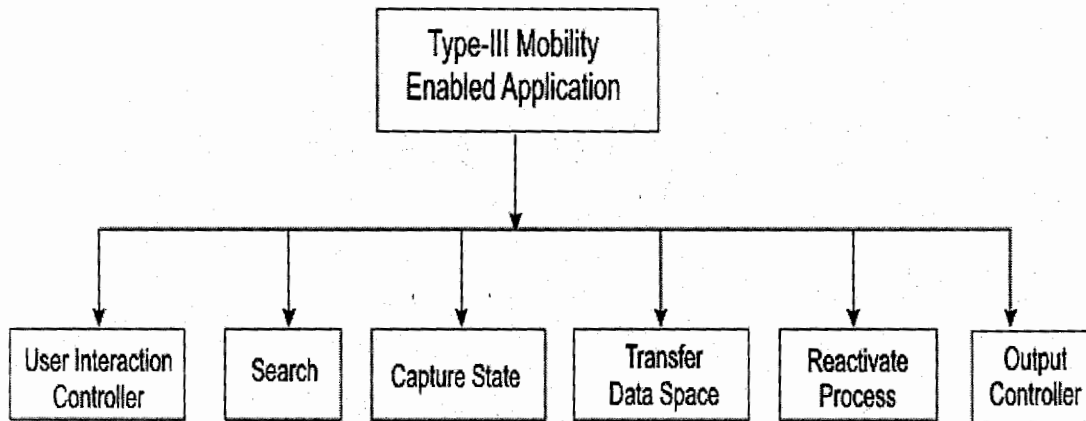
### **3.2.2 Interface Design**

The interface design describes how the software communicates with itself, to systems that interoperate with it and with humans who use it. An interface implies a flow of information (e.g. data and/or control). Therefore, the data and control flow diagrams provide the information required for interface design. The interface Design can be seen in Appendix A. The entire diagrams related to the interface are shown there.

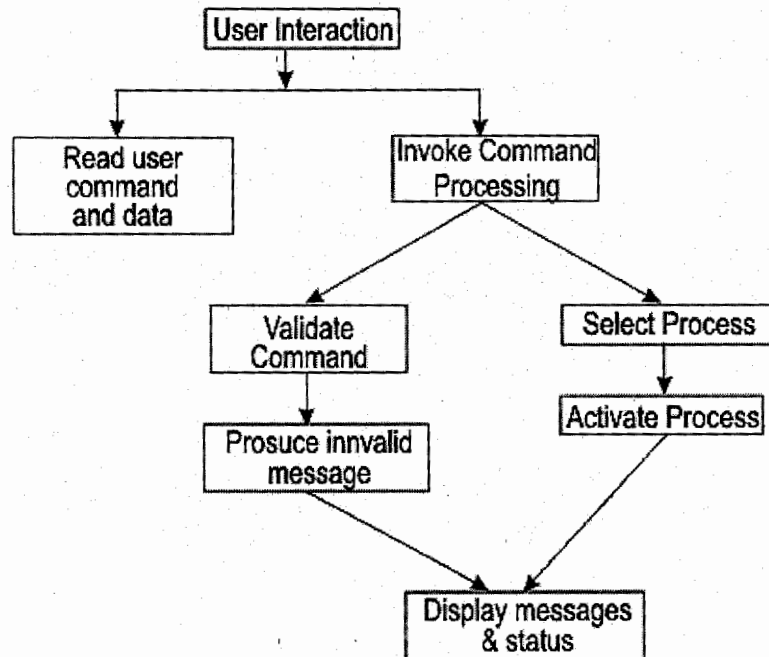
### 3.2.3 Architectural Design

The Architectural Design defines the relationship among major structural elements of the program. This design representation—the modular framework of a computer program — can be derived from the analysis model(s) and the interaction of subsystem defined within the analysis model.

The Program Structure of the software is show in Figure 3.3, 3.4, 3.5, 3.6, 3.7 of DFD level 1,2 and 3.



**Figure 3.3** Program structure



**Figure 3.4** Program Structure of User Interaction process

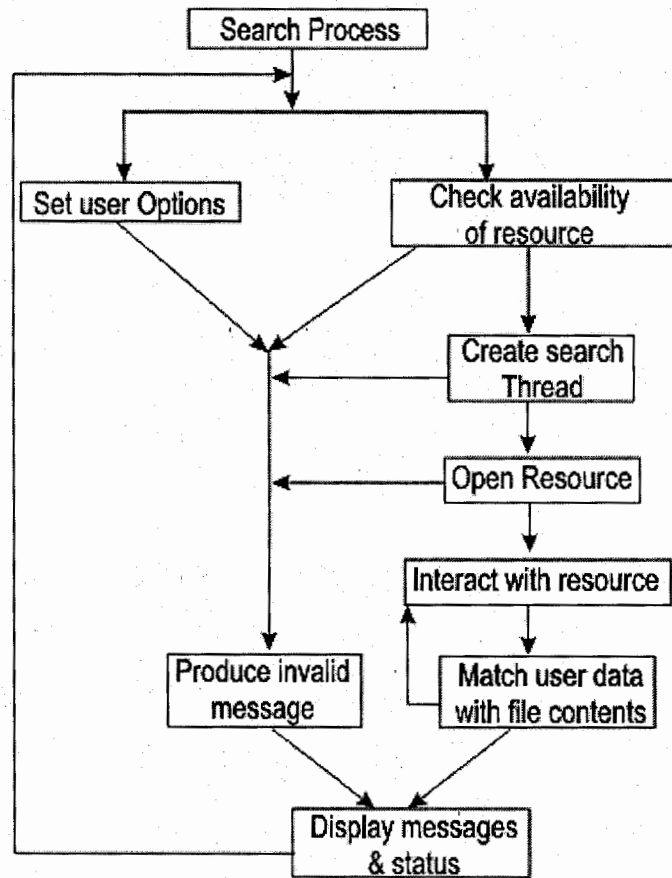
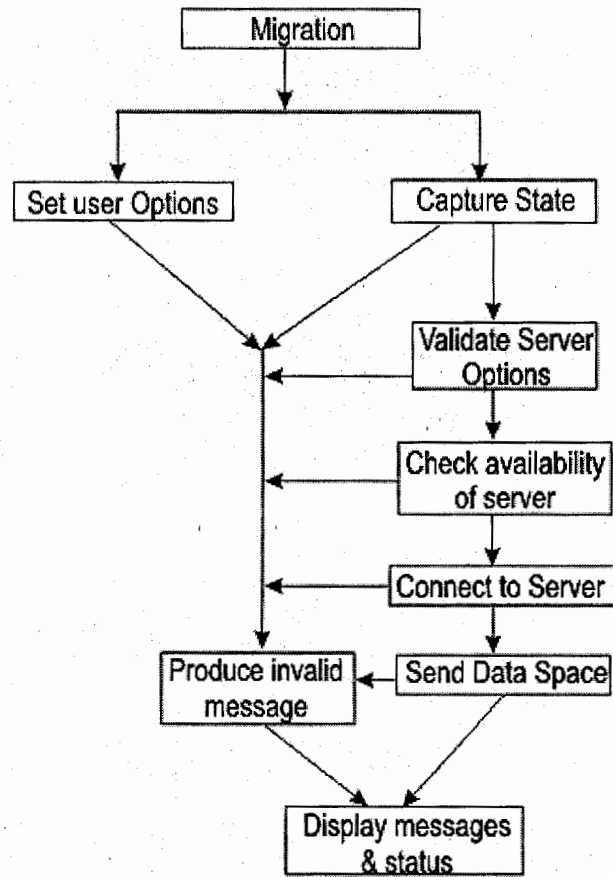
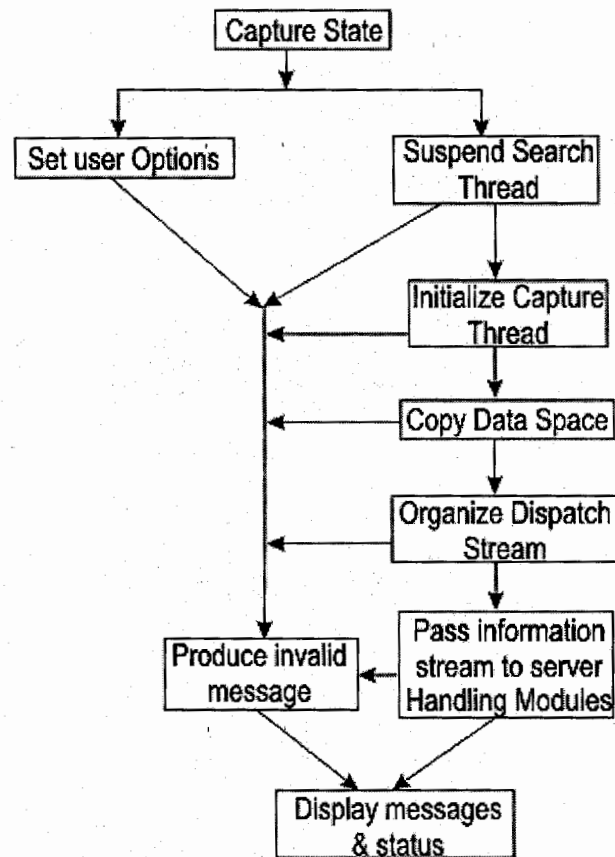


Figure 3.5 Program structure of Search process



**Figure 3.6** Program structure of Migration process



**Figure 3.7** Program structure of State Capturing process

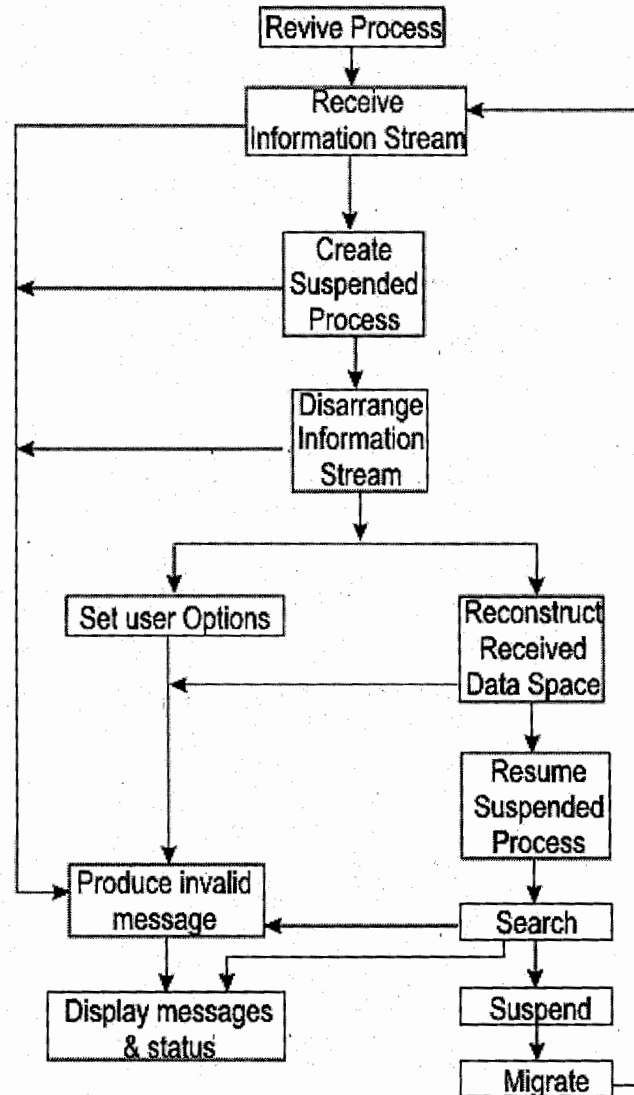


Figure 3.8 Program structure of Process Revival

### 3.2.4 Procedural Design

The procedural design transforms structural elements of the program architecture into a procedural description of software components. Information obtained from the PSPEC, CSPEC and STD serve as the basis for procedural design.

The Procedural Design for the software is shown in the Figures 3.8, 3.9, 3.10, 3.11, 3.12, 3.13 and 3.14.

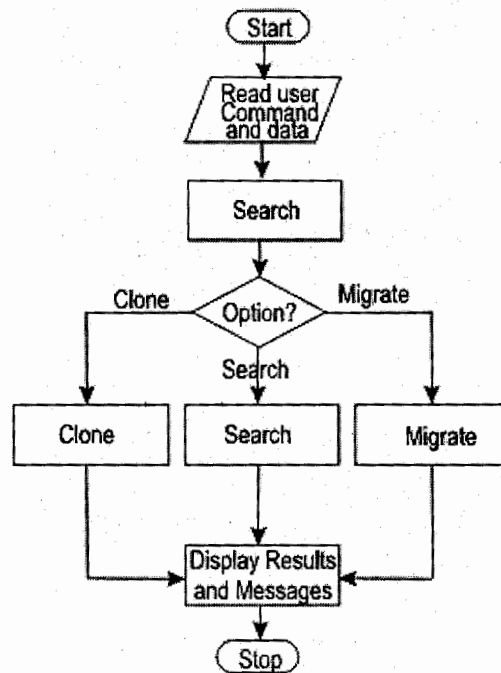


Figure 3.9 Procedural Design



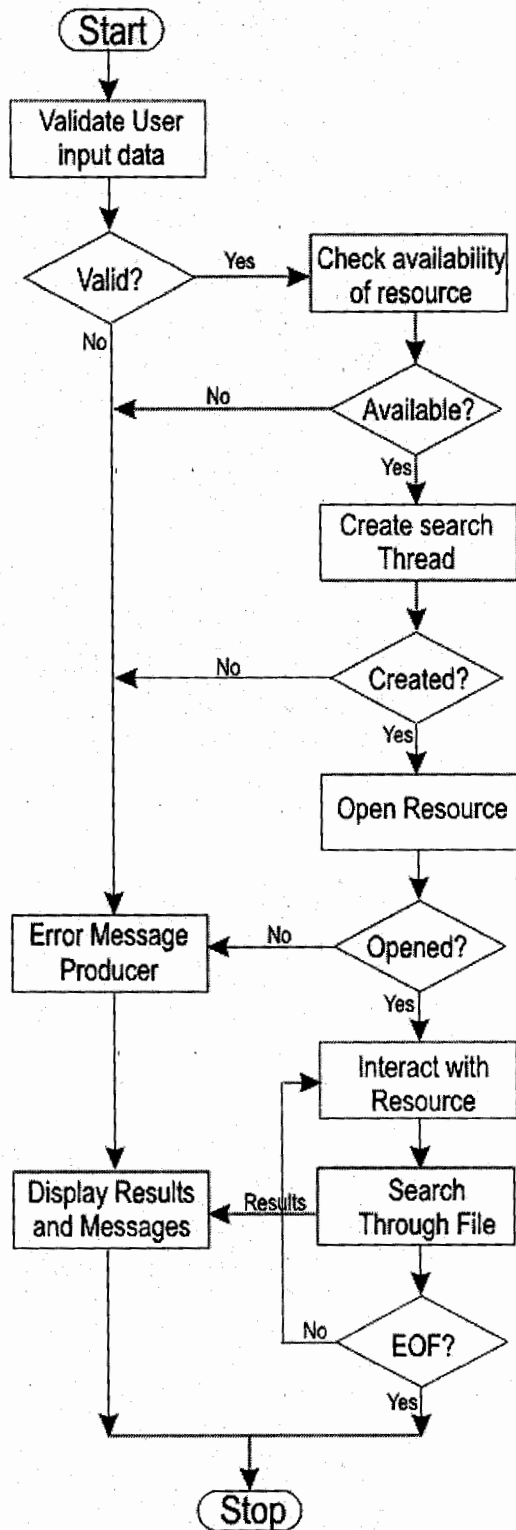


Figure 3.10 Procedural Design of Searching

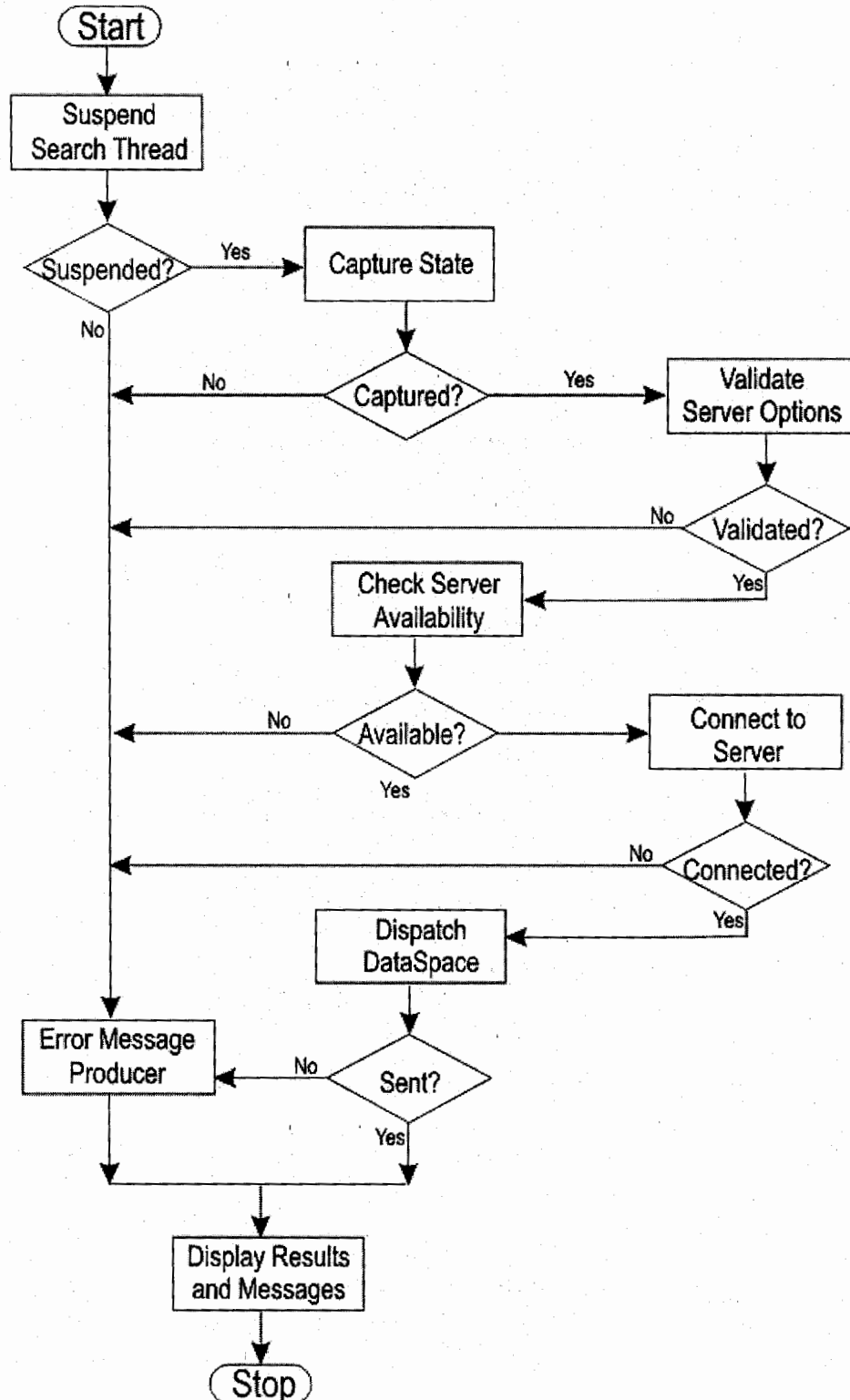


Figure 3.11 Procedural Design of Migration

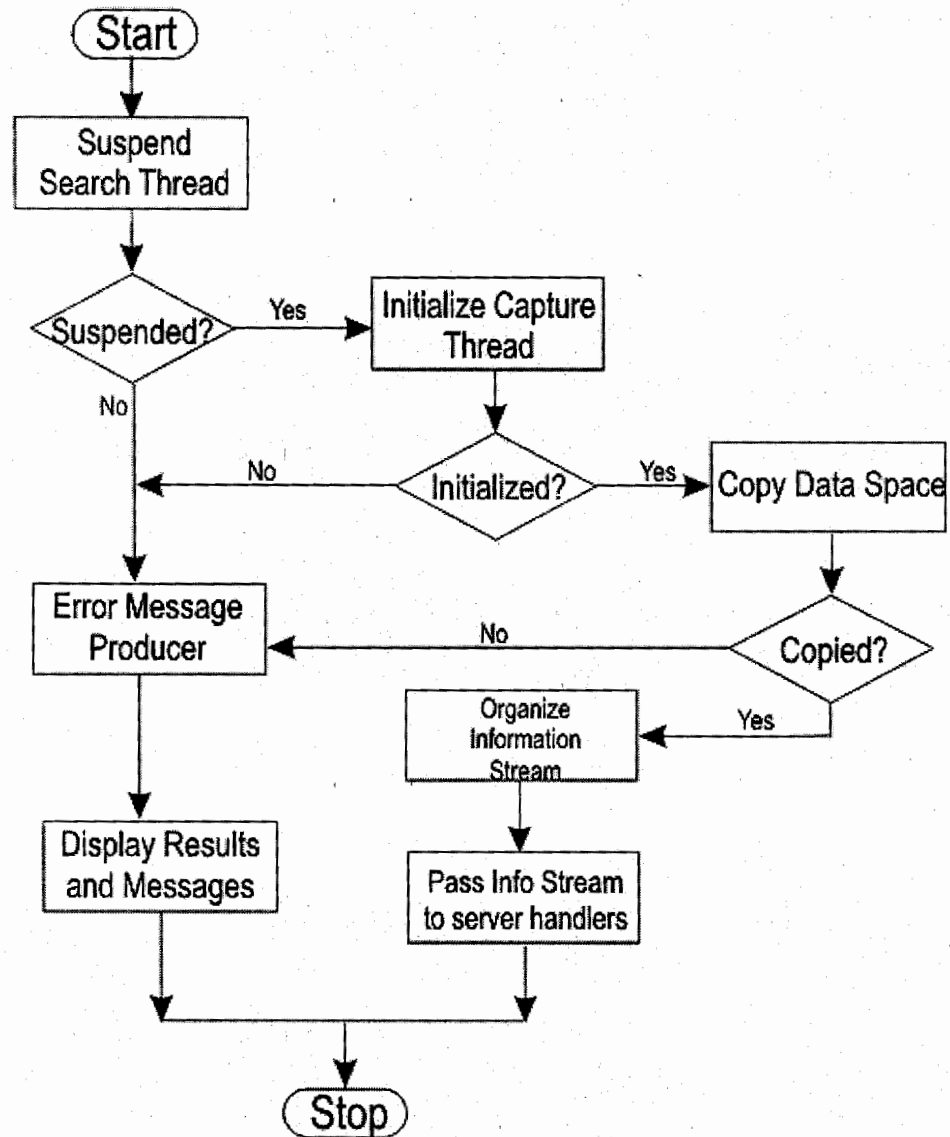
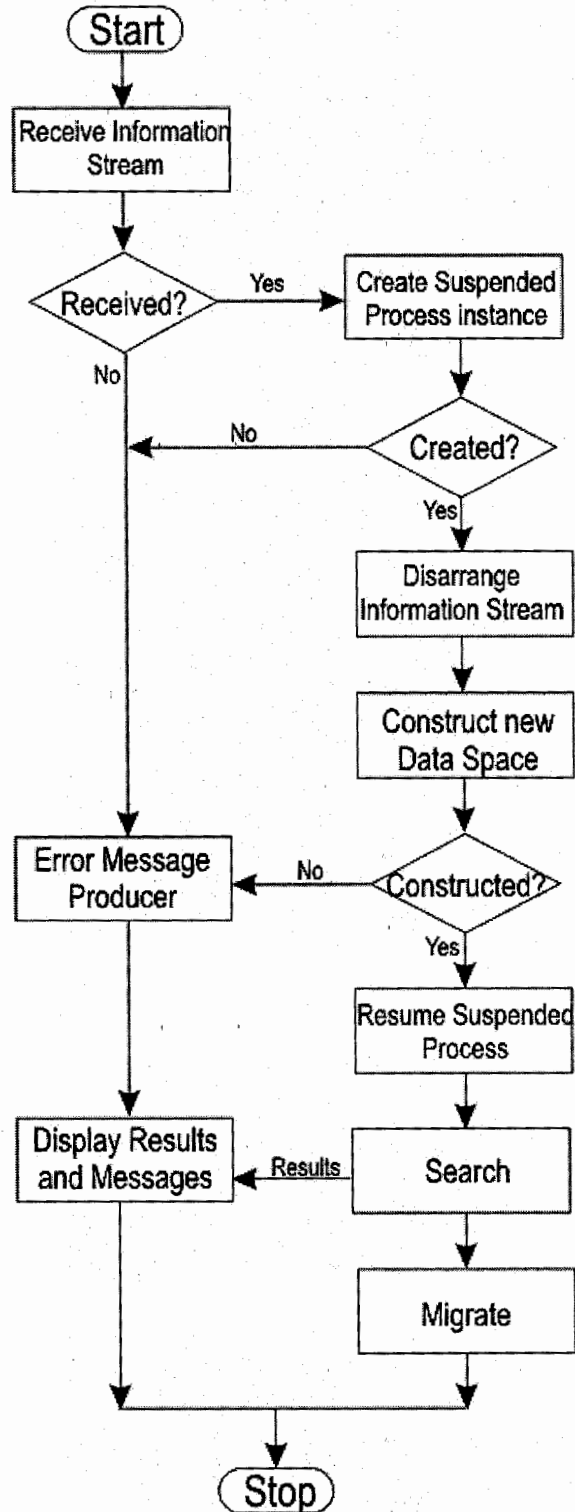
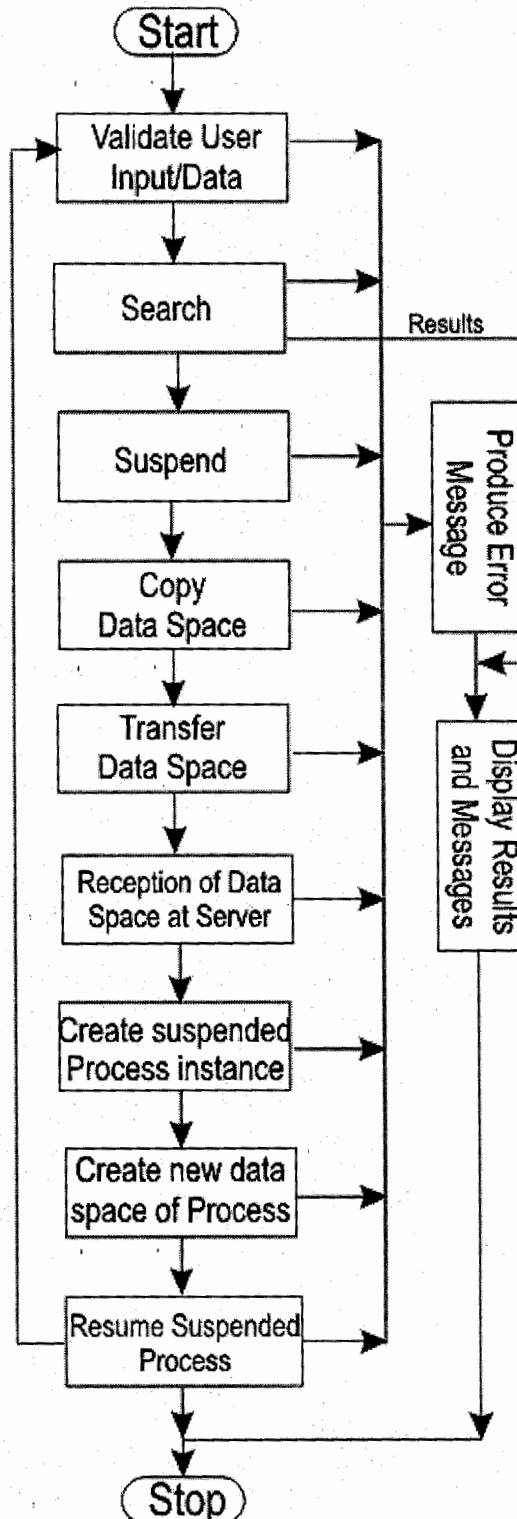


Figure 3.12 Procedural Design of Capturing Data Space



**Figure 3.13** Procedural Design of Process Revival



**Figure 3.14** Overall Procedural Design of Type-III Mobility, including client and Server

Figure 3.10, 3.11, 3.12, 3.13 and 3.14 are the procedural diagrams of Searching, Migration, State Capturing, Process Revival at Server and Overall Procedural Design

of Type-III Mobility (including client and Server) processes respectively. Figure 3.14 makes it clear that process after migration to the server can further migrate to another server/node.

The Design Activity is completed here and can easily be mapped to coding in implementation section which is last activity of the software development Process.

## Chapter 4

# **System Development and Implementation**

## 4. SYSTEM DEVELOPMENT AND IMPLEMENTATION

This is the second last activity in the project and comes before testing of the whole program; however the partial testing can be done during implementation after completion of our every module.

This software is developed using C/C++ and Visual C++. The mobility type used to implement is Type-III mobility and has been selected considering efficiency, smaller in size to travel at network, accuracy in data space collection and revival with a minimum chances of error occurrence. The software is divided into three modules searching through File, Migration and Rehabilitation of received data space at server.

### 4.1 Client Application and Migration Host

This software works as a Client-Server Applications and uses the non-sharing connection. Where the application created using Visual C++ works as a client and handles all the functionality related to Graphical User Interface. Whereas backend computational competence of C/C++ has been used. The server interface for the remote machine as well, has been designed in Visual C++ and C/C++ accomplishment at backend. Server related complex computations are performed through C/C++.

There are two connection types available among Clients and Server. The connection types are listed below.[10].

- Shared Connection
- Non-shared Connection.
- Shared Discrete Connections

To exercise client-server architecture POSIX provides a mechanism of SOCKETS, a collection of Function calls through which we can create connection(s) among server and client application(s)/process(es). We can also create connections through other mechanisms available in Visual C++ and different frameworks [10]. Main techniques



available for this purpose are listed below along with their limitations and reasons of not using them to solve our problem.

- Pipes
- Mailslots
- Dynamic Data Exchange (DDE)
- Component Object Modeling (COM)
- Distributed Component Object Modeling (DCOM)
- Remote Method Invocation (RMI)
- Remote Procedural Call (RPC)
- Also P2P (peer 2 peer) connections establishment through
  - Serial port communication
  - Parallel port communication

Though we are not bothered about the protocol to be used, but the problems or limitations with majority of the above said techniques are listed.

**Component Object Modeling (COM) by VC++:** COM is a component based solution provided for Microsoft Windows based platforms. This model provides different application with sharing a single memory block or object on a single machine through an interface provided by the developer. That is both server and client applications or processes reside on the same machine. In our case we are in need of same solution but our client and server reside on two different machines on a network.

**Dynamic Data Exchange (DDE) by VC++:** Dynamic Data Exchange is the mechanism provided by Microsoft through which an application or process communicates among GUI (graphic user interface) controls and their associated variables. This architecture is incapable to cope with our problem domain.

**Distributed Component Object Modeling (DCOM):** DCOM is a component based solution provided for Microsoft Windows based platforms. This model provides different application with remote function calling through an interface provided by the developer. In this architecture server and client may reside on two different machines over a network. The limitation for our project with this architecture is that in DCOM after a server function calling server returns result(s) back to the client, depending upon the parameters passed. In our case we are in need of partially same solution but we need server to execute the rest of the process. The other thing to note about DCOM is that DCOM server always executes functions only which are already available at server. A client is just eligible to pass this function with the parameters and use server's code for completion of its process(es). This client-server paradigm is also known as REV (Remote Evaluation).

**Remote Method Invocation (RMI) by JAVA:** RMI is the technique provided by JAVA language of calling a method located remotely over internet. The problem with RMI is that parameter(s) are passed to the remote method while called and as a result of this, the remote method executes on the basis of those parameters and returns result(s).

**Remote Procedural Call (RPC) by VC++:** RPC is the technique provided by Visual C++ of calling a procedure located remotely over internet. The problem with RPC is that parameter(s) are passed to the remote method while called and at a result of this, the remote procedure executes using those parameters and returns result(s). This is the equivalent technology to the Java's RMI. One thing to be kept in mind is, "DCOM itself, at the back end uses RPC".

**Serial & Parallel port communication:** Serial & Parallel communication can also be used to transfer data between two computer nodes, with a limitation, that this creates a P2P (peer-to-peer) scenario, while we have to implement a solution for a local area network (LAN), more precisely for internet.

**Pipes & Mailslots:** Pipes and mailslots though fulfill our requirement of connecting and dispatching data among different nodes of network, as these are real-time network communicators/dispatchers. But these are more difficult than sockets, to handle, control, complex to implement and are not much common among application

developers to use for communication. Rather pipes are mostly used in real-time networks for fast communication and data transfer.

#### **4.1.1 Shared Connection**

In this type of connection, if the server is not started/initialized/connected by any other application/process, client will start the Server in the sharing environment. Any further client with a shared connection facility, interacts with the server without creating another instance of the server, rather it just directly links itself to the previously created instance or opened connection. As a result of this both applications share the same server workspace variables. Most of the times these type of connections are created when both server and clients reside on single machine. But there are the cases when these connections are created among applications/processes over network, for example a web server over internet providing user information services.

#### **4.1.2. Non-Shared connection**

In this type of connection, if the server is not started by any other client, the client will start the server. When any other client calls for a server connection a new instance or connection is created for that client. These two clients can not share the server workspace variables.

#### **4.1.3. Shared Discrete Connection**

This is the shared connection with the difference that just one instance for all clients is created but for every client request a different thread or sub process is created. First client creates the server instance while rest of the clients use that instance to create corresponding threads for them. No one owns the server process instance.

### **4.2 Win32 API**

The term *Win32* is used to describe an Application Programming Interface (API) that is common to all of the Microsoft 32-bit Windows platforms. These platforms currently include Windows 95, Windows 98, Windows NT, and Microsoft Windows CE. The Win32 API is a set of functions, structures, messages, macros, and interfaces

that provides a consistent interface to enable you to develop applications for any of the Win32 platforms. Table 4.1 lists some of the services provided by the Win32 API.

**Table 4.1 Win32 API Services**

Win32 API service	Description
Window Management	Provides the means to create and manage a user interface.
Window Controls	Provides a set of common user interface controls. Using the common controls helps keep an application's user interface consistent with that of the shell and other applications. It also saves a significant amount of development time.
Shell Features	Provides access to system objects and resources such as files, storage devices, printers, and network resources.
Graphics Device Interface	Provides functions and related structures used to generate graphical output for displays, printers, and other graphical devices.
System Services	Provides access to the resources of the computer via features of the underlying operating system.

#### 4.2.1 SOCKET API's Description

The socket API functions and their description used to establish a connection among a client and Server are listed in Table 4.2.

**Table 4.2 Socket API Functions**

Socket	The <b>socket</b> function creates a socket that is bound to a specific service provider.
Accept	The <b>accept</b> function permits an incoming connection attempt on a socket.
bind	The <b>bind</b> function associates a local address with a socket.
connect	The <b>connect</b> function establishes a connection to a specified socket.
listen	The <b>listen</b> function places a socket a state where it is listening for an incoming connection.
WSAStartup	The <b>WSAStartup</b> function initiates use of Ws2_32.dll by a process.
WSACleanup	The <b>WSACleanup</b> function terminates use of the Ws2_32.dll.
closesocket	The <b>closesocket</b> function closes an existing socket.
shutdown	The <b>shutdown</b> function disables sends or receives on a socket.

### 4.2.2 File Functions:

Table 4.3 lists the Functions and their description which are used for serialization in C.

**Table 4.3 C Serialization Structure and Functions**

<b>FILE</b> structure	Stores information about current state of stream; used in all stream I/O operations.
<code>Ftell</code>	Gets the current position of a file pointer.
<code>Fseek</code>	Moves the file pointer to a specified location.
<code>Rewind</code>	Repositions the file pointer to the beginning of a file.
<code>Fclose</code>	Closes a stream ( <b>fclose</b> ) or closes all open streams ( <b>_fcloseall</b> ).
<code>feof</code>	Tests for end-of-file on a stream.
<code>Ferror</code>	Tests for an error on a stream.
<code>fflush</code>	Flushes a stream.
<code>Fgetc</code>	Read a character from a stream
<code>_fgetchar</code>	Read a character from stdin
<code>Fputc</code>	Writes a character to a stream
<code>Putc</code>	Writes a character to a stream
<code>Fread</code>	Reads data from a stream.
<code>Fsetpos</code>	Sets the stream-position indicator.
<code>Fwrite</code>	Writes data to a stream.
<code>Getc</code>	Read a character from a stream

### 4.3 Functionality Added to Software Using Visual C++

All the errors related to the software except computational errors are handled Using Visual C++ and it show all the error messages and other interactive message to the user. Whole Graphical Interface is designed using Visual C++ and *Screen Shots* of

which are provided in User Manual (See *Appendix A*). Following functionalities are added to the software using visual C++.

- Splash Screen at the start of the software (Both Applications).
- Single Instance of Application (Server).
- System tray Informer (Server)
- User friendly interface for user ease and functionality.
- Directory Selection Dialog box for selecting the path.
- Disabling operational interface during execution.
- Proper Error and interactive message for the user.
- Help file (Client).

### 4.3.1 Splash Screen

The Splash Screen or Welcome Screen Window is shown during Client and server initialization it also shows the status of the Application. It is a very good technique to show something on the screen while preprocessing of application is going on.

```

CSplash* m_WelcomeDlg;
m_WelcomeDlg = new CSplash();
m_WelcomeDlg->Create(IDD_SPLASH, this);
m_WelcomeDlg->ShowWindow(SW_SHOW);
if ( //pre conditions )
{
    m_WelcomeDlg->m_Message.SetWindowText("Starting
    Application...");
    UpdateData(false);
    . . . . .
    //Add initialization code here
    . . . . .
}
m_WelcomeDlg->DestroyWindow();
delete m_WelcomeDlg;

```

### 4.3.2 Single Instance of Application

Only one instance of Server Application is allowed to run in order to maintain consistency between the whole Process Migration.

```
HANDLE hmutex =
CreateMutex(NULL, TRUE, "JUSTTOCHECK");
// then check for that name whether it already
exists or not by following code,
// if exists release mutex and return false
otherwise just release the mutex
{
    if ( hmutex && GetLastError () ==
        ERROR_ALREADY_EXISTS)
    {
        ReleaseMutex (hmutex);
        exit(1);
    }
}
```

### 4.3.3 User Friendly Interface and Functionality

Server interface has been divided into three main sections.

- Search/Suspend/Resume
- Options
- Server
- Informer

#### 4.3.3.1 Search/Suspend/Resume Interface

This is the first and the nearly everyone important part of the software. This interface lets the user interact with the software throughout the lifecycle of Application/process. Search process searches for a user provided string through a file provided by user. This is the base process of our application which has to be migrated. As process continuously interacts with the resource/file during its life cycle therefore “searching through file” has been chosen for migration purpose. Before migration of this process its execution is necessary. After the execution starts, corresponding results are displayed through associated controls, and migration option is enabled. If the user has run the software for the first time that is the process has not migrated from any other

node then unless and until he configures server and sets migration options, he will be unable to carry migration of current process during execution. It connects with the resource using the library functions, defined above. And also maintain a connection with the resource during whole life cycle of search process. This area defines the following things: The first Edit Box Labeled "File Name" requires the user to specify the file/resource through which a string has to be searched. This is not an editable field through keyboard. To specify the file name user presses the "Browse" button.

1. Browse button brings the *File Open* Dialog in front of the main application and user can select the file/resource and path through this dialogue. When user presses the button following code is executed to perform the operation.

```

CFileDialog m_fileDlg(true, NULL, "Resource",
OFN_FILEMUSTEXIST, "TextFiles (*.txt)|*.txt|",
NULL);
if(m_fileDlg.DoModal()==IDOK)
{
    . . .
    // Set Resource name
    // Set Resource path
    // Enable search field
}

```

Before this code the instance of the Dialog is created. The *DoModal()* shows the dialog in front of the screen and when after selection of valid path user press ok then *SetWindowText()* sets the path to the edit box of the main application window. At the end the instance of the Dialog is deleted.

2. The search button is enabled after all validations of resource and search string. After the user presses the search button following main tasks and code lines are executed:



```

m_hSearchThread= CreateThread
(NULL, 0, (LPTHREAD_START_ROUTINE)
SearchThread, this, CREATE_SUSPENDED, &dw);
// Disable Options Button
// Enable Suspend Button

```

3. Following important lines of code are executed when the suspend button is pressed:

```

ptr->GetDlgItem(IDC_BUTTON_SEARCH)-
    >SetWindowText("Resume");
ptr->GetDlgItem(IDC_BUTTON_MIGRATE)->EnableWindow(true);
SuspendThread(ptr->m_hSearchThread);

```

4. Lines of code listed below are executed on press of "Resume":

```

ResumeThread(m_hSearchThread);
toggle=~toggle;
loop=0;
GetDlgItem(IDC_BUTTON_SEARCH)-
    >SetWindowText("Pause");
GetDlgItem(IDC_BUTTON_MIGRATE)->EnableWindow(FALSE);
GetDlgItem(IDC_BUTTON_HELP)->EnableWindow(FALSE);
GetDlgItem(IDC_BUTTON_ABOUT)->EnableWindow(FALSE);

```

The search thread includes following vital lines of code:

```

int len;
if( ptr->m_serial_no==0)
    ptr->m_results.DeleteAllItems();
CString strText, searchtext;
ptr->m_searchstring.GetWindowText(searchtext);

len=searchtext.GetLength();

ptr->fp=fopen(ptr->m_FileName, "r");
if (ptr->fp==NULL)
{
    ptr->CWnd::MessageBox("Resource is unavailable\nor invalid
resource transfer\nRetry with another resource", "Invalid
Resource", MB_ICONSTOP);
}

```

```

        ptr->OnButtonBrowse();
    }
else if(len==0)
{
    ptr->CWnd::MessageBox("Please Enter a string to
search","Missing Search String ",MB_ICONQUESTION);
    ptr->m_searchstring.SetFocus();
}
else
{
    char ch;
    fseek(ptr->fp,ptr->position,SEEK_SET);
    if( ferror( ptr->fp ) )
    {
        perror( "Read error" );
        AfxMessageBox("The resource has the same name as on
previous node but with different
attributes",MB_ICONSTOP);
        ExitThread(1);
    }

    Sleep(ptr->sleep);
    while(ch!=EOF)
    {
        Sleep(1);
        ch=fgetc(ptr->fp);
        if(ch=='\n')
        {
            if(ptr->loop==1)
            {
                ptr->GetDlgItem(IDC_BUTTON_SEARCH)-
                >SetWindowText("Resume");
                ptr->GetDlgItem(IDC_BUTTON_MIGRATE)-
                >EnableWindow(true);
                SuspendThread(ptr->m_hSearchThread);
            }
            ptr->lines++;
            ptr->pos=1;
            ptr->x=0;
            continue;
        }
    }
}

```

```

else if(ch==searchtext.GetAt(ptr->x))
    ptr->x++;
else
{
    if(ptr->loop==1)
    {
        ptr->GetDlgItem(IDC_BUTTON_SEARCH)-
        >SetWindowText("Resume");
        ptr->GetDlgItem(IDC_BUTTON_MIGRATE)-
        >EnableWindow(true);
        SuspendThread(ptr->m_hSearchThread);
    }
    ptr->pos+=ptr->x+1;
    ptr->x=0;
    continue;
}

if(ptr->x==len)
{
    strText.Format(TEXT("%d"), ptr->m_serial_no+1);
    ptr->m_results.InsertItem(ptr-
    >m_serial_no, strText);
    strText.Format(TEXT("%d"), ptr->lines);
    ptr->m_results.SetItemText(ptr->m_serial_no,    1,
    strText);
    strText.Format(TEXT("%d"), ptr->pos);
    ptr->m_results.SetItemText(ptr->m_serial_no,    2,
    strText);
    ptr->m_serial_no++;
    ptr->pos+=ptr->x;
    ptr->x=0;
    if(ptr->loop==1)
    {
        ptr->GetDlgItem(IDC_BUTTON_SEARCH)-
        >SetWindowText("Resume");
        ptr->GetDlgItem(IDC_BUTTON_MIGRATE)-
        >EnableWindow(true);
        SuspendThread(ptr->m_hSearchThread);
    }
}
}

```

```

    }
}
ptr->GetDlgItem(IDC_BUTTON_SEARCH)->SetWindowText("Search");
ptr->GetDlgItem(IDC_BUTTON_MIGRATE)->EnableWindow(FALSE);
ptr->GetDlgItem(IDC_BUTTON_BROWSE)->EnableWindow(TRUE);
ptr->GetDlgItem(IDC_BUTTON_OPTIONS)->EnableWindow(TRUE);
ptr->GetDlgItem(IDC_BUTTON_HELP)->EnableWindow(TRUE);
ptr->GetDlgItem(IDC_BUTTON_ABOUT)->EnableWindow(TRUE);
ptr->created=0;
ptr->toggle=0;
ptr->position=0;
ptr->lines=1;
ptr->pos=0;
ptr->x=0;
ptr->m_serial_no=0;
ExitThread(0);

```

### 4.3.3.2 Options Interface

This is the 2nd division of the software interface and is used to set the migration options and server configurations. User can set here the type of process resettlement to be accomplished, type of mobility to be carried out and other ones elaborated below.

#### Process. Type of the Process Migration

**Migrate.** Means the current instance of the program terminates after it tries to relocate it on another node/ backup server.

**Clone.** If selected this option the process makes a copy on host and keeps executing on the current machine as well.

**Show Complete Results on Destination.** If checked means that results found on this machine will also be displayed along with the results found on the server machine.

#### Resource. Properties of the resource/File being used

**Shared.** The resource is shared among the server and the client.

**Standalone.** Both nodes have separate resources on same addresses/ paths.

**Destination.\*** Server/destination identification which is running the service for reception of the data space of current application.

**IP.** IP address of the server.

**Name.** Name of the server

---

*\*User can provide with anyone of these two i-e IP or Name.*

**Mobility.** Type of the mobility which has to be carried out

**Weak.** Only code is dispatched to the destination.

**Strong.** Code along with data and state travels to the destination.

**Type-III.** Data and State travels to the server.

**Sleep Element.** Holdup time during searching process instruction. The reason of providing this feature is that user or tester can experience the migration process during searching. Default and the minimum value of sleep element are set to 15 milliseconds.

Option dialogue is brought on screen through the following code. This code also includes data exchange before and after the dialogue modal.

```

OptDlg dlg;
dlg.m_sleep = sleep;
dlg.m_migrate = process;
dlg.m_results = results;
dlg.m_resource = resource;
dlg.m_mobility = mobility;
dlg.m_servername = servername;
dlg.m_ipAddress = serverip;
UpdateData(true);

if(dlg.DoModal()==IDOK)
{
    sleep = dlg.m_sleep;
    process= dlg.m_migrate;
    results = dlg.m_results;
}

```

```

        resource = dlg.m_resource;
        mobility = dlg.m_mobility;
        servername = dlg.m_servername;
        serverip = dlg.m_ipAddress;
    }
    if(sleep < 1 )
        sleep=15;

```

1. First items of the dialogue box are of radio type which enables the user to set process migration type. Here

```

MIGRATE = 0;
CLONE = 1

```

Where `m_migrate` is the variable used for this radio control.

2. Second item on dialogue is a check box control which lets the user to set the results option on server. Its description has already been mentioned above. The variable used for check box control is `m_results`. Where

```

TRUE = show complete results on server
FALSE = do not show complete results on server

```

3. Then there is a Resource type selection radio control. Through this control user can specify which type of resource is this, which is being used by the process. The control variables values are handled as below

```

0 = SHARED
1 = STANDALONE

```

4. Fourth item is a group box which contains the server IP and name. First item is an IPCONTROL and the other one is an edit box. User can specify server IP or name by entering IP values or name in IP control or edit box respectively. On changing any of the field the other field gets disabled. The code for this functionality is written below.

```

void OptDlg::OnFieldchangedIpAddress (NMHDR*
    pNMHDR, LRESULT* pResult)
{

```

```

        if(m_ip.IsBlank())
            m_name.EnableWindow(true);
        else
        {
            m_name.SetWindowText("");
            m_name.EnableWindow(false);
        }
    }

    /*****/

void OptDlg::OnChangeEditName()
{
    CString str;
    m_name.GetWindowText(str);
    if(str.GetLength() > 0 )
    {
        m_ip.EnableWindow(false);
        m_ip.SetWindowText("");
    }
    else
    {
        m_ip.EnableWindow(true);
    }
}

```

5. The second last item of the dialogue is the setter for mobility type to be took placed upon user inclination. This radio type control let the user to select any three types of code mobility that is weak mobility, strong mobility and Type-III mobility. The constants used here are specified below

```

0 = WEAK_MOBILITY
1 = STRONG_MOBILITY

```

```
2 = TYPE3_MOBILITY
```

The above mentioned code of dialogue used to get information from user and direct the process functionalities depending upon these data. The tasks are performed well enough and appropriate messages are displayed on proper times.

### 4.3.3.3 Server Interface

Third part of the software interface is the server interface and used for receiving images or data spaces of processes from remote clients and recreate those processes on the basis of those images and data spaces. The server interface consists of a list control and a button which provides the following information about the images and data spaces.

1. The list control consist of the following fields
  - Serial number
  - Name
  - Size
  - Client name
  - Time of arrival
  - Time if re-execution
2. *Terminate the service* button which actually terminates the server instance from the machine. After this button's function is performed successfully, this server is no more available to the clients for connecting and receiving data space or images and rehabilitating them.

The server tries to reconstruct new data space on the basis of received information stream from the client. The code for server listening, receiving and reconstructing new process is hinted below.

```
retcode = listen(socket_descriptor,0);
if (retcode == SOCKET_ERROR)
    AfxMessageBox("Listen Failed");
    /*****/
DWORD dw;
```



```

m_AcceptThread=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
Accept,this,0,&dw);
        /*****/

DWORD dw;
m_RcvThread=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
Receive,this,0,&dw);
        /*****/

```

### **Receive Thread Code Hints.**

```

ptr->length= recv(ptr->new_socket,ptr->Buffer,sizeof ptr-
>Buffer, 0);
.
.
.
if (ptr->length == SOCKET_ERROR)
{
    continue;
    AfxMessageBox("Failed to Receive Data Space");
}
else
{
    .
    .
    //temporarily store data
    if (fpt==NULL)
    {
        AfxMessageBox("Abnormal Fixed Storage Media\ncant
        handle received Data Space");
        continue;
    }
    else
    {
        .
        .
        // parse here the received data space and //extract
        all information
        .
        .
        strText.Format(TEXT("%d"), ptr-
        >ProcessesReceived);
        ptr->m_results.InsertItem(ptr-
        >ProcessesReceived,strText);
        strText.Format(TEXT("%s"), ptr->clientname);
        ptr->m_results.SetItemText(ptr->ProcessesReceived,
        1, strText);
    }
}

```

```

        ptr->m_results.SetItemText(ptr->ProcessesReceived,
        2, ptr->received image name);
        .
        .
        strText.Format(TEXT("%d"), ptr->image_size);
        ptr->m_results.SetItemText(ptr->ProcessesReceived,
        3, strText);
        ptr->ProcessesReceived++;
    }
    Sleep (1);
}

```

### Accept Thread Code Hints.

```

while(1)
{
    ptr->addrlen = sizeof(ptr->new_sin);
    ptr->new_socket = accept(ptr->socket_descriptor, (struct
    sockaddr *) &(ptr->new_sin), &(ptr->addrlen));
    .
    .
    if (ptr->new_socket == INVALID_SOCKET)
    {
        .
        .
        AfxMessageBox("Unable to accept a connection from
        host\n Possible abnormal network conditions");
    }
    Sleep (11);
    memcpy(ptr->ClientAddr, &(ptr-
    >new_sin).sin_addr, sizeof(ptr->ClientAddr));
    ptr->pClientHostEnt = gethostbyaddr(ptr-
    >ClientAddr, sizeof(ptr->ClientAddr), PF_INET);

    if(ptr->pClientHostEnt == NULL)
    {
        .
        .
        AfxMessageBox("Get host by address failed");
        strcpy(ptr->clientname, "Not Available");
    }
    else
        strcpy(ptr->clientname, ptr->pClientHostEnt-
        >h_name);
    Sleep(5);
}

```

### 4.3.4 System Tray Informer

Whenever a new image/data space is received a message window from system tray pops up to inform that an image/data space has been received. Though have not implemented any extra processing to be done at this stage, further options for the received image can be applied here. Following function is used to create the message window before displaying.

```
int CTaskbarNotifier::Create(CWnd *pWndParent)
{
    m_pWndParent=pWndParent;
    CString      strWndClass=AfxRegisterWndClass(0,AfxGetApp()-
>LoadStandardCursor(IDC_ARROW),GetSysColorBrush(COLOR_WIN-
DOW),NULL);
    return
    CreateEx(0,strWndClass,NULL,WS_POPUP,0,0,0,0,pWndParent-
>m_hWnd,NULL);
}
```

Following function is used to show the message window.

```
ptr->m_wndTaskbarNotifier.Show("Process Received");
```

Implementation details of the above mentioned *Show* function are given below:

```
void CTaskbarNotifier::Show(LPCTSTR szCaption,DWORD
dwTimeToShow,DWORD dwTimeToLive,DWORD dwTimeToHide,int
nIncrement)
{
    unsigned int nDesktopHeight;
    .
    .
    .
    m_strCaption=szCaption;
    m_dwTimeToShow=dwTimeToShow;
    m_dwTimeToLive=dwTimeToLive;
    m_dwTimeToHide=dwTimeToHide;

    ::SystemParametersInfo(SPI_GETWORKAREA,0,&rcDesktop,0);
    nDesktopWidth=rcDesktop.right-rcDesktop.left;
    nDesktopHeight=rcDesktop.bottom-rcDesktop.top;
    nScreenWidth=::GetSystemMetrics(SM_CXSCREEN);
    nScreenHeight=::GetSystemMetrics(SM_CYSCREEN);
```

```

BOOL bTaskbarOnRight=nDesktopWidth<nScreenWidth &&
rcDesktop.left==0;
BOOL bTaskbarOnLeft=nDesktopWidth<nScreenWidth &&
rcDesktop.left!=0;
BOOL bTaskBarOnTop=nDesktopHeight<nScreenHeight &&
rcDesktop.top!=0;
BOOL bTaskbarOnBottom=nDesktopHeight<nScreenHeight &&
rcDesktop.top==0;

switch (m_nAnimStatus)
{
    case IDT_HIDDEN:
        ShowWindow(SW_SHOW);
        if (bTaskbarOnRight)
        {
            . . .
        }
        else if (bTaskbarOnLeft)
        {
            . . .
        }
        else if (bTaskBarOnTop)
        {
            . . .
        }
        else //if (bTaskbarOnBottom)
        {
            // Taskbar is on the bottom or Invisible
            m_dwDelayBetweenShowEvents=m_dwTimeToShow/(m_nSkinHeight/
            m_nIncrement);
            . . .
        }

SetTimer(IDT_APPEARING,m_dwDelayBetweenShowEvents,NULL);
    break;

    case IDT_WAITING:
        . . .

    case IDT_APPEARING:
        . . .

    case IDT_DISAPPEARING:

```

```

        SetWindowPos(NULL,m_nCurrentPosX,m_nCurrentPosY,m_nSkinWidth,m_nSkinHeight,SWP_NOOWNERZORDER | SWP_NOZORDER | SWP_NOACTIVATE);
        RedrawWindow();
        break;
    }
}

```

### 4.3.5 Proper Messages

Proper Error, Information and Question Messages are shown at proper time along with the ICONS so at first sight user can guess which kind of message is displayed by the software. Sample Messages are:

- CWnd::MessageBox("Invalid File Extension","Invalid Resource",MB\_ICONSTOP);
- CWnd::MessageBox("Could not connect to server","Server unavailable",MB\_ICONSTOP);
- CWnd::MessageBox("Supported version is too low\n for communication","Low Version",MB\_ICONSTOP);
- CWnd::MessageBox("Connection to server failed","No Connection",MB\_ICONSTOP);
- CWnd::MessageBox("Cant get Localhost entry","Bad Local Resources",MB\_ICONSTOP);
- CWnd::MessageBox("Cant get Remote Host entry","Server Name Problem",MB\_ICONSTOP);
- CWnd::MessageBox("Cant Get RemoteHost Entry","Server IP Problem",MB\_ICONSTOP);
- CWnd::MessageBox("Connection to server could not be established","No Connection",MB\_ICONSTOP);
- CWnd::MessageBox("Could not dispatch dataspace","Data Transfer Failed",MB\_ICONSTOP);
- AfxMessageBox("Abnormal Fixed Storage Media\ncant handle received Data Space");
- AfxMessageBox("Unable to accept a connection from host\n Possible abnormal network conditions");
- AfxMessageBox("Failed to Receive Data Space");

- `AfxMessageBox("Abnormal Fixed Storage Media\ncant handle received Data Space");`

### 4.3.6 Help File

A Help file is being provided along with the software to the user to consult with, in case he faces any difficulty. It has been tried our best that, the help file completely elaborates the usage of software interface and working. There is a button of Help and also user can call the Help File by *Pressing F1*. `OnHelpInfo()` Function is overwrite to invoke help on F1.

```
OnHelpInfo(HELPIFINFO* pHelpInfo)
//Function to overwrite for help
{
    OnHelpinformation();

    return false;
}

void OnHelpinformation()
{
    WinHelp(0,HELP_FINDER);
}
```

### 4.3.7 File Name and Path Selection

This is the dialog box which is created for the user to select the file/resource name and its path. Following are the important lines of code to create and display this dialog.

```
CFileDialog          m_fileDlg(true,          NULL,
"Resource",OFN_FILEMUSTEXIST,"TextFiles    (*.txt)|*.txt||",
NULL);
if(m_fileDlg.DoModal()==IDOK)
{
    CString str;
    m_searchstring.EnableWindow(true);
    str= m_fileDlg.GetPathName();
    m_filename.SetWindowText(str);
    CString str2;
    str2 = m_fileDlg.GetFileExt();
    if(str2.CompareNoCase("txt")== -1 )
    {
```

```

        CWnd::MessageBox("Invalid File
        Extension", "Invalid Resource", MB_ICONSTOP);
    }
    else
    {
        int j=0;
        int len=str.GetLength();
        str2.Format("len =%d", len);
        for (int i=0; i < len; i++)
        {
            Sleep(1);
            m_FileName[i]=str.GetAt(j);
            if(str.GetAt(j)=='\\')
            {
                m_FileName[++i]='\\';
                len++;
            }
            j++;
        }
        m_FileName[i]='\0';
    }
}
}
}

```

#### 4.3.8 Disabling operational interface controls during execution

When the system is in specific module that is searching or migration, rest of the controls, regarding to different processes are disabled to protect the user from illogical and instantaneous input to the software. Some of the code lines concerning this activity are written below.

```

GetDlgItem(IDC_BUTTON_SEARCH)->EnableWindow(FALSE);

GetDlgItem(IDC_BUTTON_MIGRATE)->EnableWindow(FALSE);

GetDlgItem(IDC_BUTTON_HELP)->EnableWindow(true);

GetDlgItem(IDC_BUTTON_ABOUT)->EnableWindow(true);

GetDlgItem(IDC_EDIT_SEARCH)->EnableWindow(FALSE);

ptr->GetDlgItem(IDC_BUTTON_SEARCH)-
>SetWindowText("Search");

```

```
ptr->GetDlgItem(IDC_BUTTON_MIGRATE)->EnableWindow(FALSE);  
  
ptr->GetDlgItem(IDC_BUTTON_BROWSE)->EnableWindow(TRUE);  
  
ptr->GetDlgItem(IDC_BUTTON_OPTIONS)->EnableWindow(TRUE);  
  
ptr->GetDlgItem(IDC_BUTTON_HELP)->EnableWindow(TRUE);  
  
ptr->GetDlgItem(IDC_BUTTON_ABOUT)->EnableWindow(TRUE);
```



Chapter 5

**Testing and Results**

## 5. TESTING AND RESULTS

Testing is an important phase during software development life cycle, and shows the stability of the product. Also it helps in comparing the final product with the objectives. Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.

Testing should focus upon the system's external behavior; a secondary purpose of testing is pushing the limits of the system in order to understand how it fails under certain conditions. A design must be testable and an implementation must be tested.

The Software basically has been divided into three components or modules (Searching, Migration and Rehabilitation & Revival). Initially unit test was performed on every unit. Syntax errors were removed and the validation checks were tested and corrected entirely. For semantic errors, every program unit was tested with the help of test data.

After this all three modules were combined to form complete software as the integration completed. Again the Syntax and semantic errors were checked and removed. After the completion of individual testing of all the modules, the modules were integrated and testing phases were applied, then errors were checked and removed.

### 5.1 Objectives of Testing

The overall objective of the testing is to find the maximum number of errors in minimum amount of effort.

### 5.2 Testing Strategies

Testing begins with unit testing, then progress towards integration testing, and finishes with validation and system testing. In unit testing single modules are tested first. Once each module is tested individually, it is integrated into a program structure while a series of regression tests are run to uncover errors due to interfacing of

modules and side effects caused by addition of new units. Finally the system as a whole is tested.

### 5.3 Types of Testing Done

We conducted various types of testing to make the software stable and error free.

- **Code Inspection**

Reviews and walk through.

- **Unit Testing**

All the modules of the project were first tested individually by inserting invalid values. Exceptions thrown were properly handled.

- **Integration Testing**

After the modules were tested individually, they were combined to form the final product. All the links and paths were tested. Invalid values were also checked and measure taken to handle them successfully.

Tests of inter object and inter process coordination should be built at several granularity levels. For example, tests of two or three interacting objects/modules, dozens of objects, and thousands of them are all needed.

- **Black Box Testing**

The software was tested on Windows 2000 Professional and measures taken that expected output is generated on input.

- **System Testing**

The Software was checked under different versions of Windows Operating Systems

- **White Box Testing**

Prior testing is part of white box testing in which we look inside the code. Here we can often find errors. This includes tests, those force most or all

computation paths to be visited, and especially those that place components near the edges of their operating conditions form classic test strategies.

- **Beta testing**

Use by outsiders rather than developers often makes up for lack of imagination about possible error paths by testers. Beta testing was done by International Islamic University, Islamabad's students.

- **Regression testing**

Tests should never be thrown out (unless the tests are wrong). Any changes in classes, modules etc., should be accompanied by a rerun of tests. Most regression tests begin their lives as bug reports.

## 5.4 Evaluation

Evaluation of the software is carried out to check the stability and usability of the product being developed. We took measures to ensure that the developed software becomes effective and easy to use. Some of the features of the product are given below.

- **Efficiency and Effectiveness**

The product developed is effective and efficient.

- **Accuracy**

The Software provides reliable results. Only Microsoft Windows based Operating systems can be used as Platform for this software (different compatible versions are mentioned on *Page viii*).

- **Easy to use**

The product is easy to use. All a deprived client has to do is,

- Connect to the server
- Send their data spaces for restoration

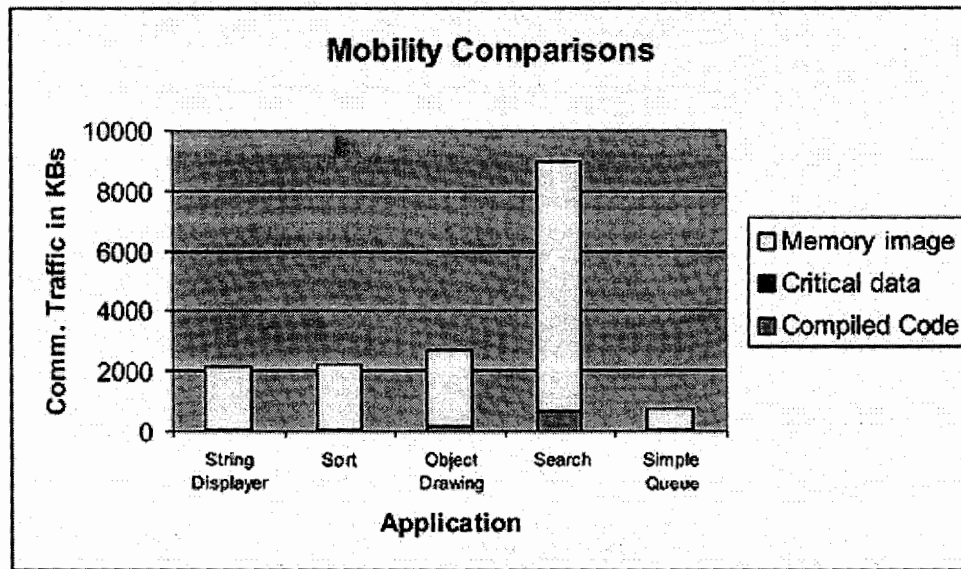


Figure 5.2 Graphical Representation of different mobility types

## 5.6 Type-III Network Traffic

The network load to achieve the Type-III code mobility is detailed through the following equation which gives the Bytes to travel on network.

$$* \text{ sizeof } \left( \sum_{i=1}^n \text{ tostring}(v_n) + \sum_{i=1}^n \text{ tostring}(v_v) \right) + NS + \text{ sizeof}(\text{PI\_id}) + \text{ sizeof}(\text{P\_H})$$

Where

<b>v_n</b>	= variable name
<b>v_v</b>	= variable name
<b>n</b>	= No. of variables
<b>N</b>	= No. of Separators
<b>S</b>	= Separator bytes
<b>PI_id</b>	= Process image identifier
<b>P_H</b>	= Protocol Header
Always ( <b>N &gt; n</b> )	

\* This Complexity is valid for applications having variable based state.

## 5.7 Enhancement

The software we have developed provides the basic functionality of code mobility on top of weak code mobility. It introduces Type-III mobility implementations. Further functionality that can be added includes:

- Strong Mobility
- Portable Data Space Migration

The overall objective of the testing process is to identify the maximum number of errors in the code with a minimum amount of efforts. Finding an error is thus considered a success rather than failure. On finding an error, efforts were made to handle and correct it.

Chapter 6

**Conclusion &  
Future Enhancements**

## 6. Conclusion & Future Enhancements

As with any project, several ideas came to light that might improve the performance of system but have not been implemented. Some of these did not seem to provide additional benefit, while others were not implemented due to lack of time. Now, we describe some of these ideas and our intuition as to whether their implementation would be beneficial.

This initial work will be extended in three directions: first, we will implement the solution for Strong Code Mobility and will cover other language and operating system features that could not be covered here. Second, we shall extend and refine our proposed Type-III model to provide a platform independent implementation. Eventually, we wish to develop a runtime library for supporting code mobility on Windows based platforms and in the next step, for independent platforms. This portion of document also includes some suggested solutions for the interest of research scholars of Code mobility. The solutions are:

### 6.1 Requirements from the Operating System

A better solution would allow the system to automatically decide whether a process can be executed remotely. If the system maintains a history of the past performance of a process, it can use a number of different metrics to decide whether a process should be executed remotely. One drawback with this approach is that the system is guessing whether the process should be executed remotely. However, every time a process is executed, the system will have more information concerning the behavior of the process. In most cases, after a relatively few number of executions, the history of a process will be mature enough to accurately predict whether the process should be executed remotely.

The following services are vital in order to support a kernel-level implementation of a migration

- Virtual address space that is arranged identically for each instance of an application/process. Namely, the code and the static data reside at the same virtual addresses in each copy of a program.



- Protection of pages in virtual memory and exception handling on a protection fault.
- Interface for the creation and management of threads, including a mechanism for obtaining and updating a thread's state.
- Some mechanism for resetting the location of threads' stacks. It should be possible to reserve a range of virtual addresses for the stack of a thread.

## 6.2 The proposed language specifications

Programming languages concerns are portability, safety, efficiency, security, confidentiality, integrity, availability and authenticity. These all issues must be achieved as in conventional languages. The concern we focus here are portability and availability. Most of the MCL's rather all of them introduced so far lack portability or availability. Therefore we take into account enhancements in Java, as it is getting popular day by day because of its portability and availability.

Given that Java Virtual Machine (JVM) provides a platform independent support to the processes written in java, also exposing Applets the only pervasive application of code mobility. Even applets are also weak mobility implementers. We propose extension in Java language such that at the next step it supports strong mobility as well. The primary features of new language are to support

- Capture of the execution state of a single Java thread, thread group, or all threads (complete process) in the JVM.
- Capture of the execution state at fine levels of granularity (ideally, between any two Java byte code instructions).
- Capture of the execution state as transparently to the Java code executing in the JVM as possible.
- Cross-platform compatibility for the execution state information.
- Flexibility in how much information is captured (in particular whether to include the definitions of Java classes).
- Easy portability to a variety of platforms (at least Win32 and various UNIX/Linux platforms).
- Flexible usage in different contexts and inside different applications.

- Enforcement of fine-grained and dynamically changing limits of access to resources such as the CPU, memory, disk, network, and GUI.

### 6.3 Focused Implementation Solution for Strong Mobility

In connection with current languages and technologies we propose a platform independent but architecture based strong mobility implementation. This solution is provided below.

Code mobility is not a new concept. The research work on distributed operating systems has followed a more structured approach. In this research area, main problem is to support the migration of active processes and objects, along with their code and data, at the operating system level over a network. In particular, process migration concerns the transfer of a process being run by an operating system, from machine where it is running to a different one [1]. Though Java Applets have gained great popularity under the definition of code mobility but they also provide weak code mobility, where only code is transferred, which yet has to start execution. There are no such common technologies which provide strong code mobility. In this paper we present an implementation of strong code mobility which is platform independent but microprocessor architecture specific. This paper discusses a technique for user level process migration between computers, for collecting the memory contents of a process on one computer in an information stream, and for restoring the data content from the information stream to the memory space of a new process on a different computer. Unlike Java feature of object serialization [2], the data fetching and re-establishment method enables complicated data structures such as indirect memory references that is pointers, to be migrated properly between two computer nodes. Study is based on Intel 386 and onward microprocessor architecture.

#### 6.3.1 Intel memory addressing technique

The microprocessor has a set of rules that apply to segments whenever memory is addressed. The rules, which apply in the real and protected mode, define the segment register and offset register combination [3]. For example, the code segment register is always used with the instruction pointer to address the next instruction in the program. This combination is **CS: IP** or **CS: EIP**, depending upon the microprocessor's mode

of operation. The **code segment** register defines the start of the code segment and the **instruction pointer** locates the next instruction within the code segment.

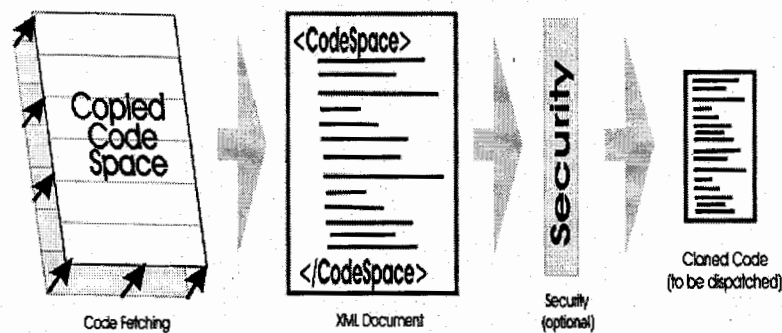


Figure 1 Mechanism from Code Fetching to Code Dispatching

### 6.3.2 Segment and Offset Addressing Scheme Allows Relocation

Our basic process model is a standard, procedural, stack based model of computation. We use the complicated scheme of segment plus offset which allows programs to be relocated in the memory system [3]. Even it also allows programs written to function in the real mode to operate in a protected mode system. A relocateable program and relocateable data are program and data that can be placed in any area of memory and are used without any change in their contents. The segment and offset addressing scheme allows both programs and data to be relocated without changing the instructions or contents of program or data, respectively. This is ideal for use in a general-purpose computer system in which not all machines contain the same memory areas. Because memory is addressed within a segment by an offset address, the memory segment can be moved to any place in the memory system without changing the offset addresses [3]. This is accomplished by moving the entire program, as a block, to a new area and then changing the contents of the segment registers.

Process migration occurs when a process is transferred between two machines which differ in software environments such as compiler, operating system, or software tools. Our study says that, if memory contents can be cloned or relocated on the same system then they can also be cloned on another system's memory over network. This network can be a set-up from p2p to internet. In this way the programs picked from real mode can also be instantiated in protected mode.

### 6.3.3 Fetching Data

In the approach to process migration, there is a need for efficient methods to identify, assemble and restore data contents of a process. From now onwards we refer data, code and state as process. For migrating a process, all data necessary for future execution of the process is collected (Figure 1) and then restored in the segments of the new process on another machine. We copy the binary contents of process from the system memory into an information stream and store them into a buffer. Since there are two basic types of data objects that can be contained in the memory space of a process, the storage object and the memory reference object [4].

Due to different operating system memory management and loading operations, a memory address used in a process on one computer may be worthless to a process on a different computer. When the executable file is loaded into the computer's main memory, its contents are placed at particular memory locations, depending on the memory management scheme used by the operating system of that node. Therefore, while a memory address in one process refers to particular data, most probably the same memory address might be undefined or may refer to any unrelated data when used by a process on another machine. For this purpose a Value-to-Pointer table is maintained to store the data addressed by any pointer. These pointers can be identified while scanning the memory for shipment of data. Each pointer entry in the table is assigned with pointer's value and offset, which makes easy to assign data values to pointers on the destination machine.

### 6.3.4 Migrating the fetched data

The fetched data, code, execution state along with required register values is transferred to the destination machine (Figure 2). We use to dispatch the copied data (Code & data etc from the computer memory) in eXtensible Markup Language (XML) tags from one machine to the other (Figure 1).

The reason why to use XML for shipment and maintenance is its several potential advantages [5]. It provides

- a general, open and self-describing data format

- an open and interoperable environment for distributed applications which rely on the concept of code mobility. This opens up the possibility to separate the maintenance from the design and implementation aspects of code mobility.
- hierarchical and complex relationship between or within dispatched data segments.
- creating structured documents in flexible ways.
- designed for use on the internet and for exchanging data.

All the above said points are directly related to our requirements and we think XML most suitable for transferring data. As XML based data transfer is not secure over a network, we apply optional security measures before transferring the process (Figure 1).

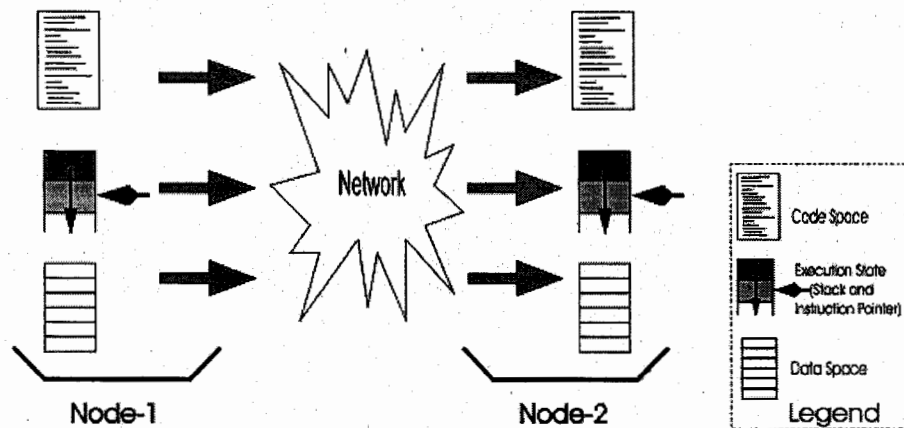


Figure 2 Abstract model of Code Migration

### 6.3.5 Protocol Encapsulation

Though we use TCP/IP for communication but did not bother about the protocol to be used while transferring the data. We are not concerned with which technique or protocol to be adopted for dispatching or receiving the information stream. That has to be decided by the programmer on the basis of operating system and software environment to be used. The process can be transferred via direct network-to-network communication (network migration) or by any other means of communication.

### 6.3.6 Code Retrieval and Reactivation

To restore the contents to the memory space on a different machine, the restoration mechanism must be able to extract the collected data from the received information

stream and reconstruct the data structure into the memory space. When we receive information stream of the migrating process on destination machine, we restore the data from the information stream and reactivate it until the end of its execution. The reverse of (Figure 1) is practiced on destination machine.

### 6.3.7 Resource Management

When a process is migrated from one machine to another there might be a case that the migrated process is using a resource at the previous node. We believe either that resource is already available at new node (Figure 3b) or resource on previous node is shared (Figure 3a). If the resource is not available at new node then that resource (Figure 3d) or a reference to that resource (Figure 3c) is dispatched along with the code, if possible. By resource we mean device(s) or file(s). The process rebinds with the resource already available at new node or previous node, if possible.

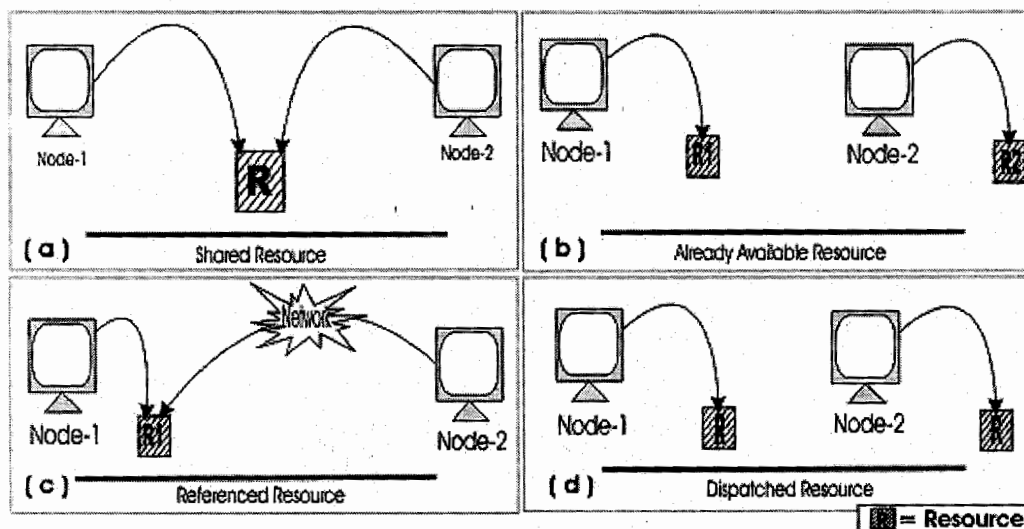


Figure 3 Resource Management

### 6.3.8 Implementation steps of Strong Mobility

Each operation involving code mobility is divided into the following steps:

- i) Determine the requested code in the system memory.
- ii) Fetch the required code along with state into a stream or buffer.
- iii) Transfer code.
- iv) Integrate code into the target system i.e.
  - a) Activate the instance of the code

- b) Connect it to the existing data or code or resource
- c) Continue its transfer over the network to yet another node if required.

These steps of implementing strong code mobility have been elaborated in Figure 4

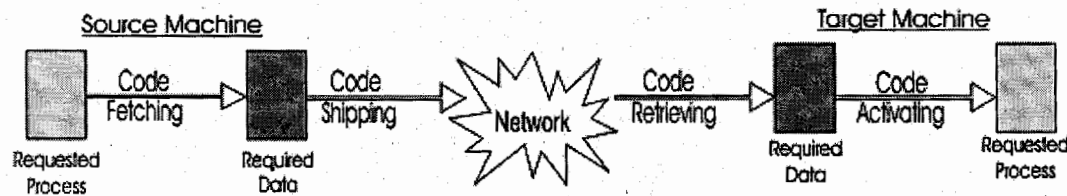


Figure 4 Four steps of Process Migration

### 6.3.9 Finale

We have uncovered a technique for collecting memory contents of a process on one computer into a platform-independent information stream, and for restoring the data content from that information stream to the memory space of a new process on a different computer. The mechanisms of data collection and restoration enable complicated data structures such as pointers to be migrated properly. This mechanism examines the current program state for migration of process and can be used in process migration, as well as in sequential and parallel distributed computing. These procedures may be used in any general solution of process migration over a network to carry out the following tasks automatically and effectively.

- Recognize the complex data structures like pointers of a migrating process for process migration.
- Replicate the data into a machine-independent format.
- Transmit the buffered information stream for a new process on the destination node.

Decode the transmitted information stream and retrieve the data in the memory space of the new process and reactivate it on the destination machine.

## **Bibliography and References**



## Bibliography and References

### Research Papers & Books

- [1] Fuggeta A, Picco G. P, Vigna G., Understanding Code Mobility: Transactions On Software Engineering. IEEE , v.24, (1998).
- [2] R. Riggs, J. Waldo, A. Wollrath, K. Bharat "Pickling State in the Java System", Computing Systems, 9(4), pp. 313-329, Fall 1996.
- [3] Barry B. Brey. The Intel Microprocessors: Fifth Edition, pp 55, 57. 2000.
- [4] Sun X, Chanchio K., Data collection and restoration for homogeneous or heterogeneous process migration: June 19, 1998.
- [5] Ralf-Dieter, Friedrich, Wolfgang K., On Maintaining Code Mobility: November 2001.
- [6] Iris Reinhartz-Berger, Dov Dori, Shmuel Katz. Modeling Code Mobility Paradigms in OPM/Web: July 2002.
- [7] William O., Automatic Process Selection for Load Balancing June 1992.
- [8] Yeshayahu A, Raphael F., Designing a process migration facility: The Charlotte experience. 1998.
- [9] Niranjana S, Jeffrey M. B, Maggie R. B, Paul T. G, Gregory A. H, and Renia J, Strong Mobility and Fine-Grained Resource, Control in NOMADS, 2000.
- [10] Adam J. F, Process State Capture and Recovery in High Performance Heterogeneous Distributed Computing Systems, 1998.
- [11] Veaceslav C., Mobile Agents vs. Mobile Code.
- [12] Gian P. P.,  $\mu$ CODE: A Lightweight and, Flexible Mobile Code Toolkit, 1998.
- [13] Anna P., Programming Languages for Mobile Code.
- [14] Navid N, Meije, Reactive Autonomous Mobile Agent.
- [15] Gianpaolo C, Carlo G, Gian P. P, Giovanni V., Analyzing Mobile Code Languages, 1998.

### Books

- [16] Barry B. Brey. The Intel Microprocessors: Fifth Edition.
- [17] Software Engineering, A Practitioner's Approach (Fourth Edition) by Roger S. Pressman 1992, McGraw-Hill Companies Inc.
- [18] Desktop Applications with Microsoft Visual C++ 6.0 MCSD Training Kit, by Julian Lindars, Microsoft Press inc.
- [19] Network Programming for Microsoft Windows, by Anthony Jones and Jim Ohlund, Microsoft Press Inc.
- [20] The Art of Assembly Language.
- [21] PC Intern.

## **Others**

- MSDN
- <http://www.CiteSeer.org/cs>

Appendix-A  
**User Manual**

## Overview

This software is developed for coping with different distributed computing requirements. These type of software can be used anywhere to overcome the complexity of process mobility. This program exploits an easy to use interface and an extraordinarily accurate, fast and reliable technique called **Type-III Mobility** to migrate the processes over a network and internet in broad-spectrum.

This software is developed in C/C++ and Visual C++.

This Program is developed by: Software Development Team

To get this software Contact: Contact us at <http://www.iiui.edu.pk>

To learn about the program see the Main Windows (Applications)

## Splash Screen

Whenever the application starts (unless it is migrated from a remote location) a splash screen appears on the screen. During this period necessary initializations and pre computations are carried out. Figure 1 and Figure 2 show File Search clients and Migration Host's splash Screens respectively.



Figure 1 File Search Client Splash Screen

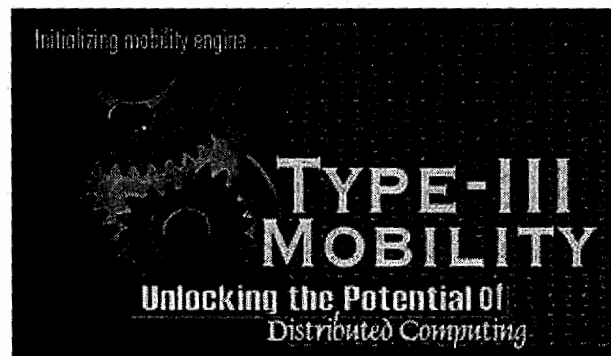


Figure 2 Migration Host Splash Screen

## Main Window (File Search Application)

This is what the main program window looks like (Figure 3)

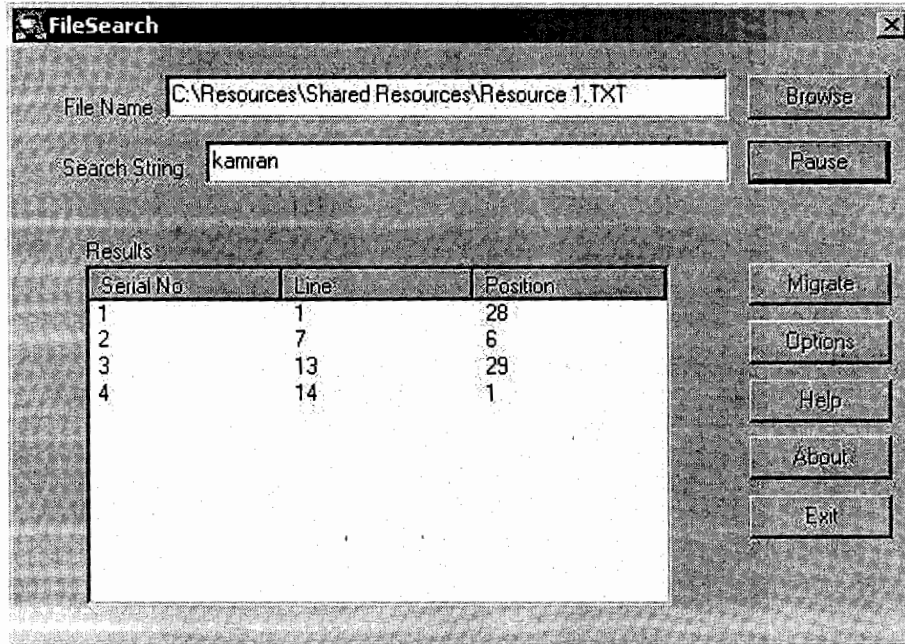


Figure 3 Client Main Window

**File Name** It shows the path to the resource/file along with name (Figure 4). User can not type the file name here directly rather he has to select the file with the help of dialog, appeared by pressing browse button.

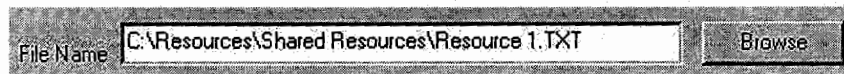


Figure 4 File Name

**Open File Dialog** User presses the browse button to locate the resource/file through which a string has to be performed. Dialog appearing in Figure 5 is popped up on pressing of browse button.

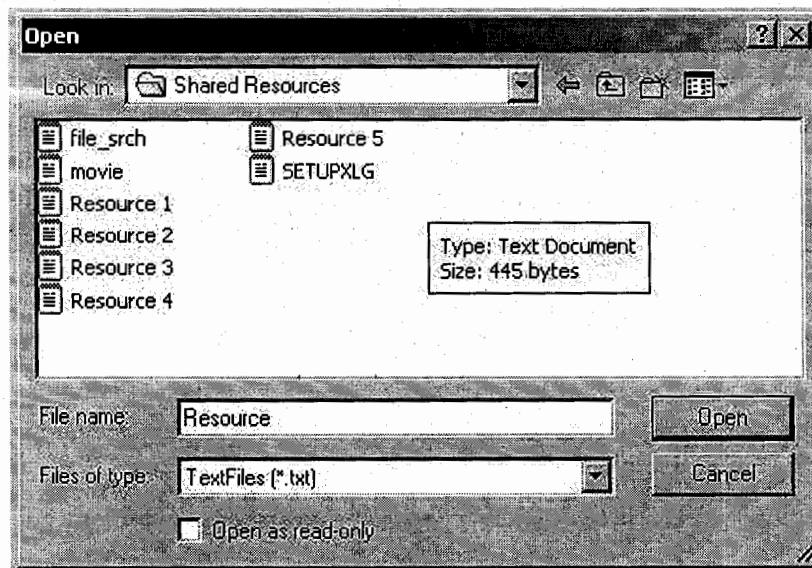


Figure 5 Open File Dialog



Figure 6 Search String

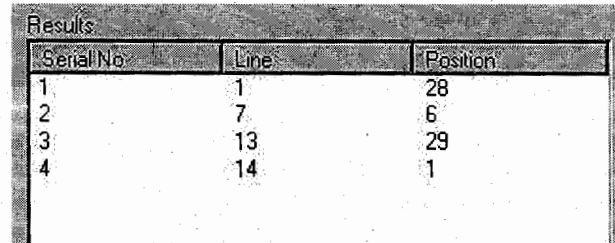
**Search String** In the edit box labeled “Search String” (Figure 6), user enters the string which has to be searched through file (resource).

**Pause/Resume** After the searching has been started the text of control button labeled “Search” is changed to “Pause”. On pressing of pause searching process is suspended and the control button’s label alters to “Resume”. On pressing of resume the searching process again starts execution where it was suspended. Keep in mind that it is necessary to suspend the process before we want it to migrate. Pause/Resume button captions are shown in Figure 7.



Figure 7 Pause (i) &amp; Resume (ii) Buttons

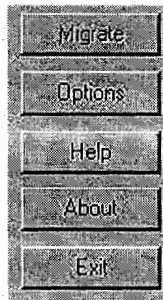
**Results** This is a list control, which is used to show the search results. Whenever a match of user provided string is confined in file (resource), corresponding results are immediately shown in result area. Controls used for results are shown in Figure 8.



Serial No	Line	Position
1	1	28
2	7	6
3	13	29
4	14	1

**Figure 8** Search Results Display

Where serial number is the number of occurrence, line is the line number on which occurrence spotted and the position tells the column number or the character number in that line from where the search string started.



**Figure 9** Control Pane

**Control Pane** Figure 9 demonstrates the different controls provided to the user for handling the software rather using the software in an efficient way. These include options, help, about and exit button. All these cause an options dialog, help file, about box and exit from the current instance of application, respectively.

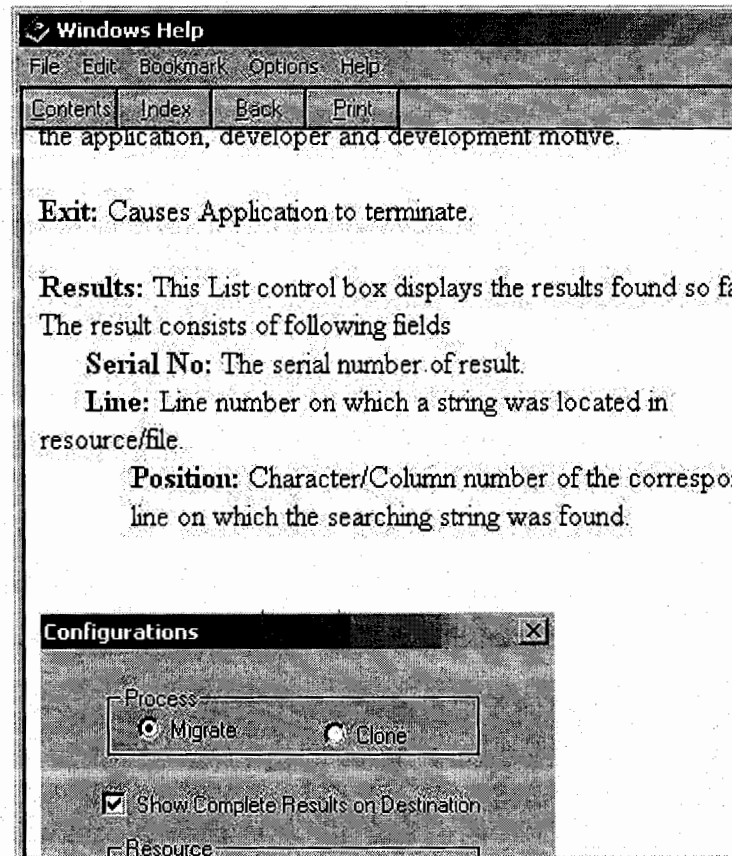
**About Box** About box consists of the information regarding the development team, motive of development and some other information. A screenshot of About Box is shown in Following Figure 10.





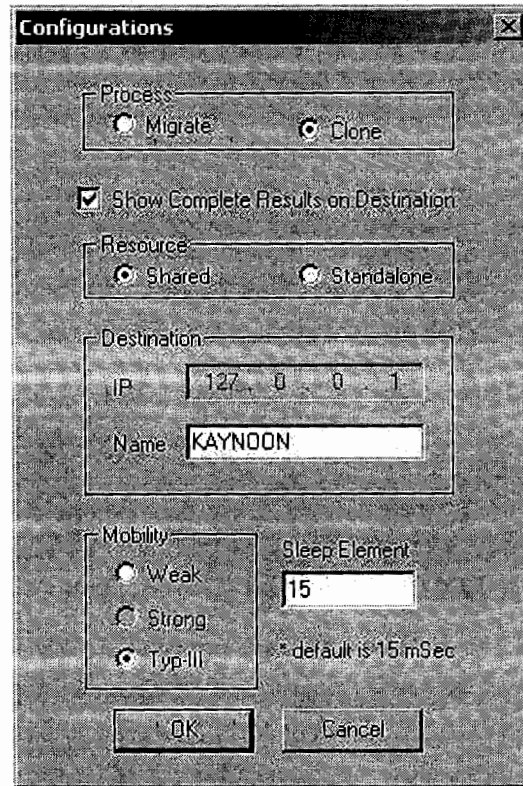
**Figure 10** Client's About Box

**Help File** Whenever during execution user needs help about the software, he may press help button to appear online help regarding this software. This feature works only if the process has not been migrated from any other computer. Figure 11 shows a glimpse of help file.



**Figure 11** Client Help File

**Options** Pressing the “Options” button on control pane caused the options dialog, shown in Figure 12, to be appeared. Through this dialog user can set many options concerning mobility, process and server. Options are elaborated in more detail in subsequent headings coming on.

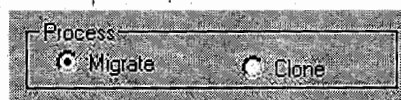


**Figure 12** Options Dialog

### **Process** Type of the Process Migration (Figure 13)

**Migrate** Means the current instance of the program terminates after it tries to relocate it on another node/ backup server.

**Clone** If selected this option the process makes a copy on host and keeps executing on the current machine as well.



**Figure 13** Process Type

**Show Complete Results on Destination** If checked means that results found on this machine will also be displayed along with the results found on the server machine (Figure 14).

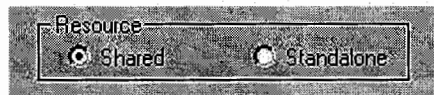


**Figure 14.** Result Display options

**Resource** Properties of the resource/File being used (Figure 15)

**Shared** The resource is shared among the server and the client.

**Standalone** Both nodes have separate resources on same addresses/ paths.



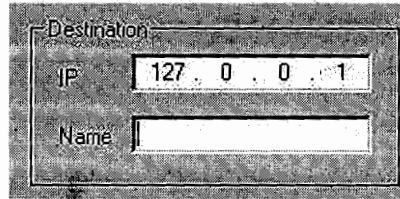
**Figure 15** Resource Type

**Destination** Server/destination identification which is running the service for reception of the data space of current application (Figure 16).

**IP** IP address of the server.

**Name** Name of the server

\*user can provide only one that is Server IP or Server Name but not both at a time. When entry in edit box named "Name" is made IP field is disabled or vice versa.


 A dialog box titled "Destination" with two input fields. The first field is labeled "IP" and contains the text "127 . 0 . 0 . 1". The second field is labeled "Name" and is currently empty.

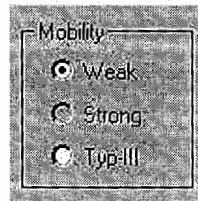
**Figure 16** Destination Selector

**Mobility** Type of the mobility which has to be carried out (Figure 17)

**Weak** Only code is dispatched to the destination.

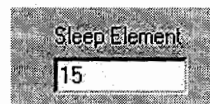
**Strong** Code along with data and state travels to the destination.

**Type-III** Data and State travels to the server.


 A dialog box titled "Mobility" with three radio button options: "Weak", "Strong", and "Type-III". The "Weak" option is selected.

**Figure 17** Mobility type Selector

**Sleep Element** Holdup time during searching process instruction. The reason of providing this feature is that user or tester can experience the migration process during searching. Default and the minimum value of sleep element is 15 milliseconds.

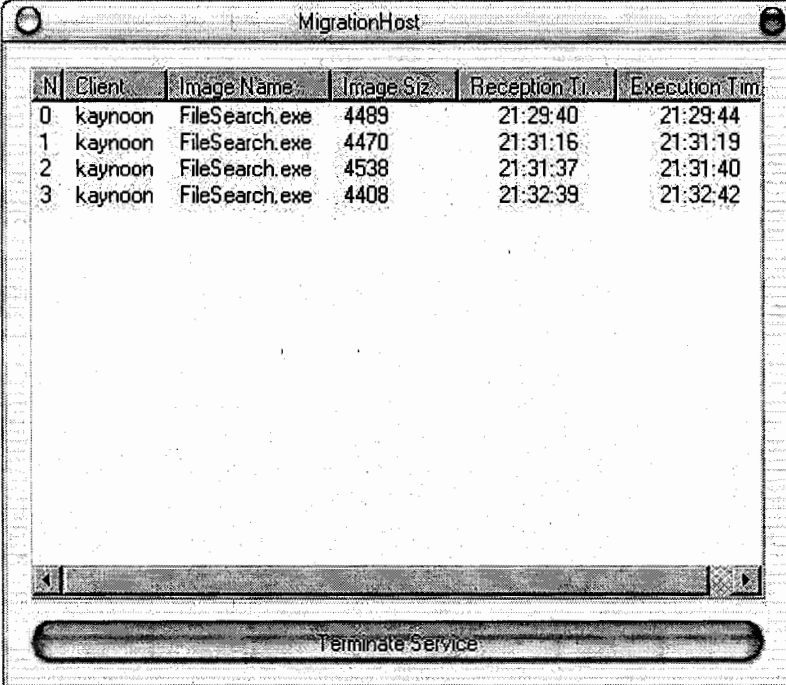

 A dialog box titled "Sleep Element" with a single text input field containing the number "15".

**Figure 18** Sleep/Delay element

User manual comes to an end here. Now we see the different aspects of server application, named Migration Host.

## Main Window (File Search Application)

This is what the main server program window looks like (Figure 19)

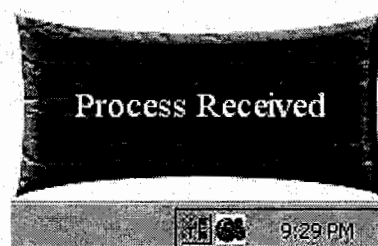


N	Client	Image Name	Image Siz	Reception Ti	Execution Tim
0	kaynoon	FileSearch.exe	4489	21:29:40	21:29:44
1	kaynoon	FileSearch.exe	4470	21:31:16	21:31:19
2	kaynoon	FileSearch.exe	4538	21:31:37	21:31:40
3	kaynoon	FileSearch.exe	4408	21:32:39	21:32:42

Terminate Service

**Figure 19** Migration Host interface

**System Tray Informer** Whenever a process image or data space is received by server a message window pops up from system tray to inform or notify to the user that a process image or data space has just been received. Though we have not but, here further options can be applied. An appearance of system tray informer has been shown in Figure 20.



**Figure 20** System Tray Informer

**Received Process' History** A complete list of process images/data spaces is organized by server. A quick look is shown in Figure 21 of this list.

This list includes

- **Serial No** Sequence number of received process/data space with respect of arrival.
- **Client Name** Name of client who dispatched its process to Migration Host for rehabilitation/revival.
- **Image name** Name of Process whose image/data space was received.
- **Image Size** Size of the memory image of process after revival/execution.
- **Reception Time** Time at which the process image/ data space was received
- **Execution Time** Time at which the process of received image/data space was rehabilitated of re-energized.

N	Client	Image Name	Image Siz...	Reception It...	Execution Tim
0	kaynoon	FileSearch.exe	4489	21:29:40	21:29:44
1	kaynoon	FileSearch.exe	4470	21:31:16	21:31:19
2	kaynoon	FileSearch.exe	4538	21:31:37	21:31:40
3	kaynoon	FileSearch.exe	4408	21:32:39	21:32:42

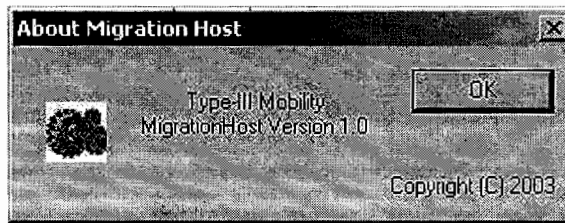
**Figure 21** Received Process Images History

**Service Terminator** The Figure 22 shown below shows the control button, the user is provided with to terminate the server that is Migration Host. After the migration host is terminated this particular system or node will not be able to receive the process images/data spaces anymore.



**Figure 22** Terminate Service

**About Box** About Box of the server consists of version information of the Migration Host. About Box is also shown in Figure 23



**Figure 23** Server About Box

Since Migration host was intended to be a service running endlessly on server machines, even then an interface to the server has been provided. The reason of not making it the service was just to demonstrate the user, the capabilities of Type-III mobility. Although an interface has been provided but it has kept in mind that

- No more than one instance of this application can exist at a time.
- Task bar entry has been removed
- System tray icon has been provided for interaction
- User can not close this application accidentally.

Appendix-B  
**Basic Concepts**



## 1. Process

Our basic process model is a standard, procedural, stack based model of computation. In a conventional sequential programming language, the runtime view of a program is an executing unit which consists of a *code segment*, which provides the static description of the program's behavior, and a program *state*. The state contains the local data of all active routines together with control information, such as the value of the instruction pointer and the value of return points for all active routines.

## 2. Process Migration

Process migration is the function which controls how code mobility is achieved. As a result of successful process migration the suspended process on client resumes its execution on server.

## 3. Migration Types

Migration can be either proactive or reactive. In *proactive* migration, the time and destination for migration are determined autonomously by the migrating EU. In *reactive* migration, movement is triggered by a different EU that has some kind of relationship with the EU to be migrated, e.g., an EU acting as a manager of roaming EUs. The remote cloning

## 4. Cloning

Cloning is just like migration but after migration current/source instance terminates its execution but not in case of cloning. Normally word cloning has been used in the sense of copying on remote node.

## 5. Execution Unit

EU informally describes a running program with an associated state of execution. Typical examples of EUs are single-threaded processes or individual threads of a multi-threaded process. Resources represent external entities that any process interacts with.

## 6. Computational Environment

CE is defined as computational environment provided to an EU for execution and dynamic linking.

## 7. Existing Mobile Code Paradigms

Fuggetta explains that traditional approaches to software design are not sufficient when designing large scale distributed applications that exploit code mobility and dynamic reconfiguration of software components. In these cases, he continues, the concepts of location, distribution of components among locations and migration of components to different locations need to be taken into account at the design stage.

The three main design paradigms exploiting code mobility are

1. Remote Evaluation
2. Code on Demand
3. Mobile Agents

<b>Common Mobile Code Paradigms extends the Client-Server approach from Data to Code</b>			
	<b>Remote Evaluation</b>	<b>Code on Demand</b>	<b>Mobile Agent</b>
<b>Client</b>	code	Data	Data + code
<b>Server</b>	Data	Code	-
<b>Initiator</b>	Client sends the code to server and gets back results	Client requests the code from server and executes it	The agent initiates its transfer from client to server

**Table 1:** Common Code Mobility Paradigms

In fact thrice of the remote evaluation, code on demand and mobile agent extends the client-server approach. In which one party in requester and the other is server. Any of these two initiate the process (see Table 1).

### 8. Which Processes Can Execute Remotely?

Once a load balancing algorithm is determined, how can we decide which processes should migrate to a target client? The goal of load balancing is to increase the throughput of each workstation so that the maximum numbers of processes are executed in the least amount of time. The goal of process selection for remote execution is to find those processes that, when executed remotely increase the overall throughput for the entire system. We must determine which processes can be migrated and which of those processes should be migrated.

- Which processes should be migrated?
- Should all processes be considered candidates for migration, or only a few particular CPU intensive processes?
- How are CPU-intensive processes to be identified?

While they provide the question, the issue is left open. We must find some means to determine which processes are the best candidates for remote execution.

The three main design paradigms exploiting code mobility are

1. Remote Evaluation
2. Code on Demand
3. Mobile Agents

<b>Common Mobile Code Paradigms extends the Client-Server approach from Data to Code</b>			
	<b>Remote Evaluation</b>	<b>Code on Demand</b>	<b>Mobile Agent</b>
<b>Client</b>	code	Data	Data + code
<b>Server</b>	Data	Code	-
<b>Initiator</b>	Client sends the code to server and gets back results	Client requests the code from server and executes it	The agent initiates its transfer from client to server

**Table 1:** Common Code Mobility Paradigms

In fact thrice of the remote evaluation, code on demand and mobile agent extends the client-server approach. In which one party in requester and the other is server. Any of these two initiate the process (see Table 1).

## 8. Which Processes Can Execute Remotely?

Once a load balancing algorithm is determined, how can we decide which processes should migrate to a target client? The goal of load balancing is to increase the throughput of each workstation so that the maximum numbers of processes are executed in the least amount of time. The goal of process selection for remote execution is to find those processes that, when executed remotely increase the overall throughput for the entire system. We must determine which processes can be migrated and which of those processes should be migrated.

- Which processes should be migrated?
- Should all processes be considered candidates for migration, or only a few particular CPU intensive processes?
- How are CPU-intensive processes to be identified?

While they provide the question, the issue is left open. We must find some means to determine which processes are the best candidates for remote execution.

First, we must define which processes are eligible for migration. Ha<sup>^</sup>c and Jin state: If, in the system not loaded by any additional processes, the mean response time of a process executed locally is greater than the mean response time caused by migration of this process and relative files, then this process is called *migratable* in the sense of load balancing. Otherwise, the process is called *nonmigratable*.

While Gait states that during execution, a process may be rescheduled on another Processor if:

- The local resident time slice becomes exhausted;
- An idle processor waiting for processes from other nodes makes entry into the shared file.
- The process is swapped out to make place for newly created (or high priority) processes.

Also system critical processes at any stage and at any cost, even fulfilling the above said conditions and postulates can not be migrated. Examples of these processes are smss, crss, winlogon, services, lsass (Accounts Managing services) and some more on WINDOWS XP and WINDOWS 2000 series operating systems.

## 9. Process Evaluation for Migration

Another method for selecting processes to execute remotely is to estimate from past performance whether the process should migrate. There are two problems that must be considered when using past performance as a guide to the migratability of a process. What characteristics should a process exhibit to be eligible for remote execution, and what data is available from previous executions of the process?

In *History*, proposed by Svennson, the amount of CPU time necessary to complete execution is used to determine whether a process should execute remotely. This approach is based on the assumption that the greatest cost associated with executing a process remotely is the amount of CPU time necessary to send the process to the target client. Upon completion, the amount of CPU time used by the process is recorded, and averaged in with all previously recorded times.

## 10. Solutions for implementing Code Mobility

For a convenient solutions scheme development or introduction we divide these solutions in two categories.

- Homogeneous Systems
- Heterogeneous Systems

### 10.1 Homogeneous Systems

Following text suggests solutions for homogenous systems.

#### 10.1.1 Kernel Level Mechanism

Process state capture mechanisms to support activities such as process migration and checkpoint/restart have been the subject of a great deal of research, both in terms of mechanisms and policies. Ideal homogeneous state capture mechanisms can be implemented inside operating systems at the kernel level due to efficiency concerns and because a process' external state is more readily available at this level. For example, systems such as Charlotte, Sprite utilize kernel-level state capture and recovery mechanisms to support process migration. Although these and other kernel-level homogeneous state capture mechanisms differ in certain performance related respects, they share a common basic approach to capturing the state of a process. The state of the process is commonly defined to consist of:

- **Virtual Memory:** code, stack, and data segments of the process's address space
- **Open Files:** file descriptors, file pointers, I/O buffers, etc.
- **Communication Buffers:** connection information, message buffer contents, etc.
- **Processor State:** current condition codes, program counter, stack pointer, general purpose registers, etc.
- **Environment Data:** process identifier, user name, etc.

All of these parts of the process state are accessible at the kernel level, and thus state capture involves marshalling or communicating this information in a well-defined format. For example, capturing a process' virtual memory might involve saving the

page table of the process along with the contents of any valid pages. Of course, the implementation of state capture operations varies widely depending on the intended use and context. For example, during process migration, an effort is often made to transfer the minimal state needed to restart the process at its destination first, and to transfer remaining state subsequently to reduce migration latency. Alternatively, for the purposes of check pointing, incremental schemes for saving a process's memory, such as periodically capturing only dirty pages, may improve time/space performance. Beyond the obvious issue of heterogeneity, kernel-level state capture schemes have a number of undesirable features in met system contexts. First, as the number of different architecture and operating system platforms grows, the issue of mechanism portability becomes important in addition to efficiency concerns. Furthermore, in metasystem approaches, it is typically infeasible to mandate replacement of the operating system on all participating nodes. These issues, along with the requirement of support for heterogeneity, strongly suggest the use of user-level state capture mechanisms.

## 10.2 Heterogeneous Systems

User Level Mechanism described below recommends an ideal solution of code mobility for heterogeneous systems.

### 10.2.1 User Level Mechanism

A number of systems to date have provided some form of homogeneous process state capture implemented at the user level (i.e. without direct, special kernel support). For example, Condor performs process state capture and recovery in homogeneous environments by using a slightly modified core dump of the process to capture and recover memory and processor state. Needed operating system specific information associated with the process is maintained at the user level by tracking the parameters and return values of all system calls via wrapper routines. An alternative approach described by Plank et. al. links programs with a special library that contains code to capture a process's internal state. In this design, processor state is captured using the Unix `setjmp` system call.

Although these approaches are typically somewhat less efficient than kernel level implementations, user-space designs are generally more portable (e.g. Condor and

Libchckpt are highly portable among Unix-based platform). A common argument against user-level state capture schemes is the difficulty involved in capturing and recovering a process's external and kernel-level state. For example, some user-level approaches such as Condor and Mandelberg/Sunderam restrict certain forms of intra-process communication mechanisms. However, this argument against user-level state capture mechanisms is largely unfounded, as demonstrated systems such as MIST/MPVM, Fail-safe PVM, and Hector. These systems provide a location independent communication layer that leave process migrations transparent to message passing operations performed at the application level. Similar user-level wrappers are possible for other services that involve external state such as file systems.

## **Appendix-C**



## Implementing Strong Code Mobility

Muhammad Kamran Naseem, Sohail Iqbal and Khalid Rashid  
Department of Computer Science, International Islamic University, Islamabad, Pakistan

---

**Abstract:** This study presents the idea of implementing strong code mobility in terms of platform independence, microprocessor architecture reliance and resource management. Proposed system establishes a shared connection with the resources and its surrounding environment based on distributed structured XML-based knowledge. The resources managed by the process are shared between the nodes, so that the developer can program in a centralized setting. The goal is to present a solution for strong code mobility for commonly used platforms.

**Key words:** Strong code mobility, code mobility, code mobility implementation

---

### INTRODUCTION

Code mobility is not a new concept. The research work on distributed operating systems has followed a more structured approach. In this research area, main problem is to support the migration of active processes and objects, along with their codes and data, at the operating system level over a network. In particular, process migration concerns the transfer of a process being run by an operating system, from machine where it is running to a different one<sup>[1]</sup>. Though Java Applets have gained great popularity under the definition of code mobility but they also provide weak code mobility, where only code is transferred, which yet has to start execution. There are no such common technologies which provide strong code mobility. In this study an implementation of strong code mobility which is platform independent but microprocessor architecture specific is presented. A technique for user level process migration between computers, for collecting the memory contents of a process on one computer in an information stream and for restoring the data content from the information stream to the memory space of a new process on a different computer are discussed. Unlike Java feature of object serialization<sup>[2]</sup>, the data fetching and re-establishment method enables complicated data structures such as indirect memory references that is pointers, to be migrated properly between two computer nodes. Study is based on Intel 386 and onward microprocessor architecture.

**Intel memory addressing technique:** The microprocessor has a set of rules that apply to segments whenever memory is addressed. The rules, which apply in the real

and protected mode, define the segment register and offset register combination<sup>[3]</sup>. For example, the code segment register is always used with the instruction pointer to address the next instruction in the program. This combination is CS: IP or CS: EIP, depending upon the microprocessor's mode of operation. The code segment register defines the start of the code segment and the instruction pointer locates the next instruction within the code segment.

**Segment and offset addressing scheme allows relocation:** The complicated scheme of segment plus offset which allows programs to be relocated in the memory system was used<sup>[3]</sup>. Even it also allows programs written to function in the real mode to operate in a protected mode system. A relocate-able program and data are program and data that can be placed in any area of memory and used without any change in their contents. The segment and offset addressing scheme allows both programs and data to be relocated without changing the instructions or contents of program or data, respectively. This is ideal for use in a general-purpose computer system in which not all machines contain the same memory areas. Because memory is addressed within a segment by an offset address, the memory segment can be moved to any place in the memory system without changing the offset addresses<sup>[3]</sup>. This is accomplished by moving the entire program, as a block, to a new area and then changing the contents of the segment registers.

Process migration occurs when a process is transferred between two machines which differ in software environments such as compiler, operating system or software tools. Presented study defines that, if memory

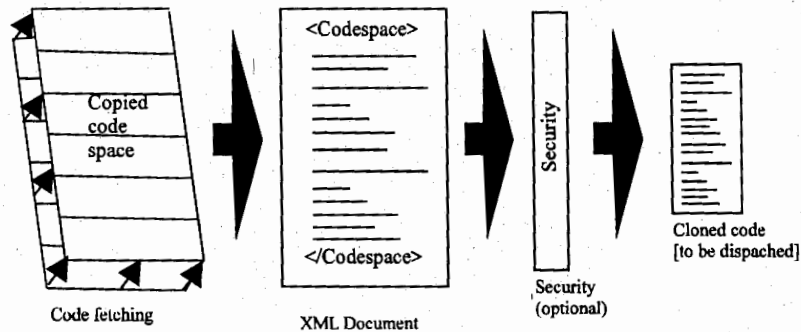


Fig. 1: Mechanism from code fetching to code dispatching

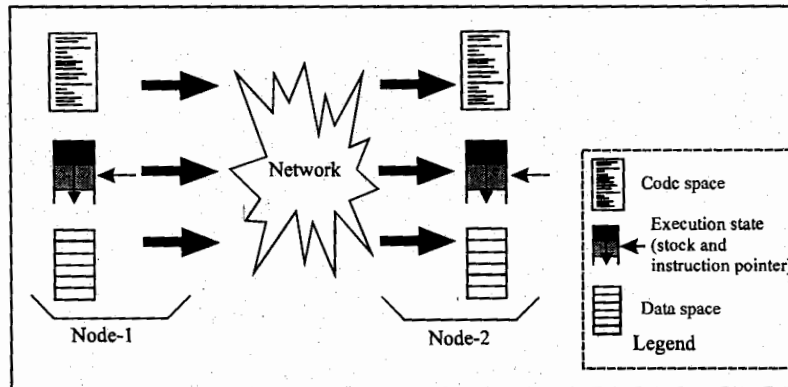


Fig. 2: Abstract model of code migration

contents can be cloned or relocated on the same system then they can also be cloned on another system's memory over network. This network can be a set-up from p2p to internet. In this way the programs picked from real mode can also be instantiated in protected mode.

**Fetching data:** In the approach to process migration, there is a need for efficient methods to identify, assemble and restore data contents of a process. From now onwards we refer data, code and state as process. For migrating a process, all data necessary for future execution of the process was collected (Fig. 1) and then restored in the segments of the new process on another machine. We copy the binary contents of process from the system memory into an information stream and store them into a buffer. Since there are two basic types of data objects that can be contained in the memory space of a process: the storage object and the memory reference object<sup>[4]</sup>.

Due to different operating system memory management and loading operations, a memory address used in a process on one computer may be worthless to a process on a different computer. When the executable

file is loaded into the computer's main memory, its contents are placed at particular memory locations, depending on the memory management scheme used by the operating system of that node. Therefore, while a memory address in one process refers to particular data, most probably the same memory address might be undefined or may refer to any unrelated data when used by a process on another machine. For this purpose a Value-to-Pointer table was maintained to store the data addressed by any pointer. These pointers can be identified while scanning the memory for shipment of data. Each pointer entry in the table was assigned with pointer's value and offset, which makes easy to assign data values to pointers on the destination machine.

**Migrating the fetched data:** The fetched data, code, execution state along with required register values was transferred to the destination machine (Fig. 2). We use to dispatch the copied data (Code and data etc. from the computer memory) in extensible Markup Language (XML) tags from one machine to the other (Fig. 1).

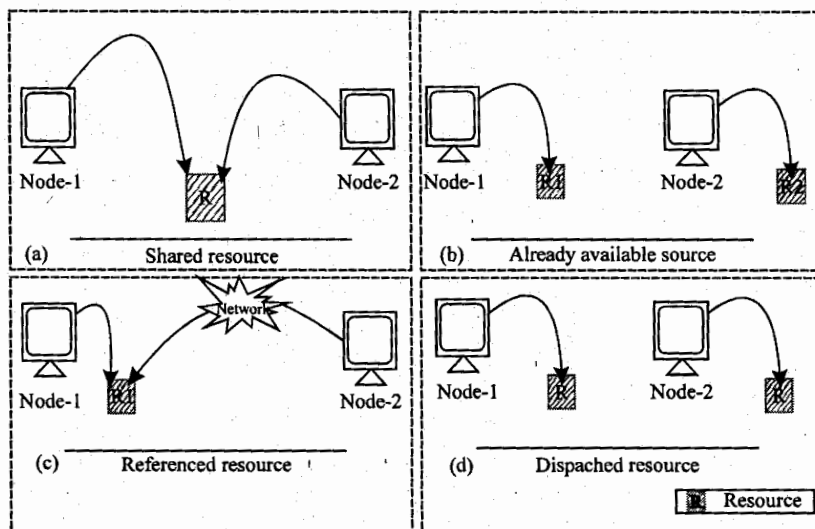


Fig. 3: Resource management



Fig. 4: Four steps of process migration

The reason to use XML for shipment and maintenance is that it provide several potential advantages:

1. A general, open and self-describing data format
2. An open and interoperable environment for distributed applications which rely on the concept of code mobility. This opens up the possibility to separate the maintenance from the design and implementation aspects of code mobility.
3. Hierarchical and complex relationship between or within dispatched data segments.
4. Creating structured documents in flexible ways.
5. Designed for use on the internet and for exchanging data.

All the above mentioned points are directly related to the requirements and it was observed that XML was most suitable for transferring data. As XML based data transfer was not secure over a network, we apply optional security measures before transferring the process (Fig. 1).

**Protocol encapsulation:** Though TCP/IP for communication was used but we did not bother about the

protocol to be used while transferring the data.

The technique or protocol to be adopted for dispatching or receiving the information stream was not concerned but may be decided by the programmer on the basis of operating system and software environment to be used. The process can be transferred via direct network-to-network communication (network migration) or by any other means of communication.

**Code retrieval and reactivation:** To restore the contents to the memory space on a different machine, the restoration mechanism must be able to extract the collected data from the received information stream and reconstruct the data structure into the memory space. When we receive information stream of the migrating process on destination machine, we restore the data from the information stream and reactivate it until the end of its execution. The reverse of (Fig. 1) was practiced on destination machine.

**Resource management:** When a process was migrated from one machine to another there might be a case that the migrated process was using a resource at the previous node. We believe either that resource is already available at new node (Fig. 3b) or resource on previous node is

shared (Fig. 3a). If the resource is not available at new node then that resource (Fig. 3d) or a reference to that resource (Fig. 3c) is dispatched along with the code, if possible. Here by resource we mean device (s) or file (s). The process rebinds with the resource already available at new node or previous node, if possible.

**Steps of strong code mobility implementation:** Each operation involving code mobility is divided into following steps:

- (I) Determine the requested code in the system memory
- (ii) Fetch the required code along with state into a stream or buffer
- (iii) Transfer code
- (iv) Integrate code into the target system: I-e
  - (a) Activate the instance of the code
  - (b) Connect it to the existing data or code or resource
- © Continue its transfer over the network to yet another node if required.

These steps of implementing strong code mobility have been elaborated in Fig. 4.

## DISCUSSION

### Applications of process migration include

**Load distribution:** migrating processes from overloaded machines to under loaded machines to exploit unused computing cycles.

**Fault resilience:** Migrating processes from those machines that may experience partial failure.

**Resource sharing:** Migrating processes to machines with special hardware or other unique resources such as databases or peripherals required for computations.

**Data access locality:** Migrating processes toward the source of data.

**Mobile computing:** Migrating processes from a host to a mobile computer.

A technique for collecting memory contents of a process on one computer into a platform-independent information stream and for restoring the data content from that information stream to the memory space of a new process on a different computer was presented. The mechanisms of data collection and restoration enable complicated data structures such as pointers to be migrated properly. This mechanism examines the current program state for migration of process and can be used in process migration, as well as in sequential and parallel distributed computing. These procedures may be used in any general solution of process migration over a network to carry out the following tasks automatically and effectively.

- Recognize the complex data structures like pointers of a migrating process for process migration
- Replicate the data into a machine-independent format
- Transmit the buffered information stream for a new process on the destination node
- Decode the transmitted information stream and retrieve the data in the memory space of the new process and reactivate it on the destination machine

## REFERENCES

1. Fuggeta, A., G.P. Picco and G. Vigna, 1998. Understanding Code Mobility: Transactions on Software Engineering. IEEE., 24.
2. Riggs, R., J. Waldo, A. Wollrath and K. Bharat, 1996. Pickling State in the Java System, Computer Sys., 9: 313-329.
3. Barray, B.B., 2000 The Intel Microprocessors: 5th Edn., pp: 55-57.
4. Sun, Xian-He and K. Chanchio, 1998. Data collection and restoration for homogeneous or heterogeneous process migration. Software Practice and Experience, 32: 1-27.

## Notes



