MODELING AND TESTING OF GUI'S USING FINITE STATE MACHINES

A DISSERTATION

Submitted in partial fulfillment of the
requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

By
SHREY AHUJA -031230
GEETIKA UPADHYAY-031273
NEHA GULATI -031229

JAYPEE UNIVERSITY OF
INFORMATION TECHNOLOGY

Department of Computer Science Engineering and Information Technology,
Jaypee University of Information Technology, Waknaghat, Solan - 173215,
Himachal Pradesh, INDIA.

MAY 2007

CERTIFICATE

This is to certify that the work entitled, "Modeling and Testing of GUI's using Finite State Machines" submitted by "Shrey Ahuja(031230), Geetika Upadhyay(031273) and Neha Gulati(031229)" in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Supervisor:

Mr. Nitin
Senior Lecturer
Department of Computer Science Engineering and Information Technology,
Jaypee University of Information Technology,
Waknaghat, Solan – 173215, Himachal Pradesh,
INDIA.

ACKNOWLEDGEMENTS

We wish to express our earnest gratitude to Mr. Nitin, for providing us invaluable guidance and suggestions, which inspired us to submit this project report on time.

We would also like to thank all the staff members of Computer Science and Engineering Department of Jaypee University of Information Technology, Waknaghat, for providing us all the facilities required for the completion of this project report.

Last but not least we wish to thank all my classmates and friends for their timely suggestions and cooperation during the period of our project report.

Shrey Ahuja (22/05/07)
Shrey Ahuja (031230) Neha Gulati (22/05/07)
Geetika Upadhyay (031273) Geetika (22/05/07)
Neha Gulati (031229)

ABSTRACT

Most of the Human-Computer-Interfaces will be materialized by Graphical User Interfaces (GUI). With the growing complexity of the computer-based system, also their GUIs become more complex, accordingly making the test process more and more costly. The project introduces a holistic view of fault modeling that can be carried out as a complementary step to system modeling, enabling a precise scalability of the test process, revealing many rationalization potential while testing. Appropriate formal notions and tools enable to design and test the software systematically. Based on a basic black box test criteria, test case selection can be carried out efficiently. The elements of the approach will be narrated by a realistic example which will be used also to validate the approach. In our case the example been considered is that of an Intelligent Automated Teller Machine owing to its widespread use in today's world as well as its growing complexity to make it more user friendly. We will showcase through this example how formal tools and notions can be used efficiently to design the GUI. Furthermore, we test the same using the notion of Black Box Testing.

Table of Contents

List of Figures and Tables

CHAPTER 1

INTRODUCTION

There are two distinct types of construction work while developing software:

- Design, implementation, and test of the programs.
- Design, implementation, and test of the user interface (UI).

We assume that UI might be constructed separately, as it requires different skills, and different techniques than construction of common software. The design part of the development job requires a good understanding of user requirements. The implementation part requires familiarity with the technical equipment, i.e. programming platform and language. Testing requires both: a good understanding of user requirements and familiarity with the technical equipment. Our Project is about UI testing that includes testing of the programs that materialize the UI and considering the design aspects. Testing GUIs is a difficult and challenging task for many reasons: First, the input space possesses a great, potentially indefinite number of combinations of inputs and events that occur as system outputs wherein external events may interact with these inputs. Second, even simple GUIs possess an enormous number of states which are also due to interact with the inputs. Last but not least, many complex dependencies may hold between different states of the GUI system, and/or between its states and inputs.

Test Cases generally require the determination of meaningful test inputs and expected system outputs for these inputs. Accordingly, to generate test cases for a GUI, one has to identify the test objects and test objectives. Robust systems also possess a good exception handling mechanism, i.e. they are responsive not in terms of behaving properly in case of correct, legal inputs, but also by behaving good-natured in case of illegal inputs, generating constructive warnings, or tentative correction trials that help to navigate the user to move in the right direction. In order to validate such robust behavior, one needs systematically generated erroneous inputs which would usually entail injection of undesired events into the Software Under Test (SUT). Such events would usually transduce the software under test into an illegal state, e.g. system crash, if the program does not possess an appropriate exception handling mechanism. Test inputs of GUI

represent usually sequences of GUI objects activities and/or selections that will operate interactively with the objects i.e. Interaction Sequences (IS). Such an interactive sequence is complete (CIS), if it eventually invokes the desired system responsibility. Another tough problem while testing is the decision when to stop. Exercising a set of test cases, the test results can be satisfactory, but this is limited to these special test cases. Thus, for the quality judgement of the program under test one needs further, rather quantitative arguments, usually materialized by well-defined coverage criteria. The most well known coverage criteria base either on special, structural issues of the program to be tested (implementation orientation/white-box testing), or its behavioral, functional description (specification orientation/black-box testing). The favored methods for modeling concentrate on finite-state–based techniques, i.e. state transition diagrams and regular events. For the systematically, scalable generating and selection of test sequences, and accordingly, for the test termination, the notion Edge Coverage of the state transition diagram will be introduced.

1.1. Project Specifications

1.1.1. Hardware Specification:

Operating System-Microsoft Windows XP professional 2002, SP2

Primary Memory- 512 MB RAM

Processor-Intel® Pentium® 4 CPU 2.4 GHz

Secondary Memory-80 HDD

1.1.2. Software Specification

Front End- Java (jdk-6-windows-i586), XML, MetaEdit, JCreator

Back End – MS Access (using JDBC driver)

1.2. Java

Java has gained enormous popularity since it first appeared. Its rapid ascension and wide acceptance can be traced to its design and programming features, particularly in its promise that you can write a program once, and run it anywhere. Java was chosen as the

programming language for network computers (NC) and has been perceived as a universal front end for the enterprise database. As stated in Java language white paper by Sun Microsystems: "Java is a simple, object-oriented, interpreted, secure, portable, and dynamic."

(i) Java is simple

Java is considered a much simpler and easy to use programming language when compared to the other object-oriented programming languages as it provides a lot of ease to make interfaces and has no use of pointers.

Java is Object-oriented programming models the real world. Everything in the world can be modeled as an object. Java is centered on creating objects, manipulating objects, and making objects work together.

(ii) Portability: Program once, Run anywhere (Platform Independence)

One of the most compelling reasons to move to Java is its platform independence. Java runs on most major hardware and software platforms, including Windows 95 and NT, the Macintosh, and several varieties of UNIX. Java applets are supported by all Java-compatible browsers. By moving existing software to Java, you are able to make it instantly compatible with these software platforms. JAVA programs become more portable. Any hardware and operating system dependencies are removed.

(iii) Java is interpreted

An interpreter is needed in order to run Java programs. The programs are compiled into Java Virtual Machine code called bytecode. The bytecode is machine independent and is able to run on any machine that has a Java interpreter. Normally, a compiler will translate a high-level language program to machine code and the code is able to only run on the native machine. If the program is run on other machines, the program has to be recompiled on the native machine. With Java, the program need only be compiled once, and the bytecode generated by the Java compiler can run on any platform.

(iv) Security

Java is one of the first programming languages to consider security as part of its design. The Java language, compiler, interpreter, and runtime environment were each developed with security in mind. The compiler, interpreter, and Java-compatible browsers all contain several levels of security measures that are designed to reduce the risk of security compromise, loss of data and program integrity, and damage to system users. Considering the enormous security problems associated with executing potentially untrusted code in a secure manner and across multiple execution environments, Java's security measures are far ahead of even those developed to secure military systems.

(v) Reliability

Security and reliability go hand in hand. Security measures cannot be implemented with any degree of assurance without a reliable framework for program execution. Java provides multiple levels of reliability measures, beginning with the Java language itself. The Java compiler provides several levels of additional checks to identify type mismatches and other inconsistencies. The Java runtime system duplicates many of the checks performed by the compiler and performs additional checks to verify that the executable bytecode form a valid Java program.

(vi) Multimedia: Images, Sounds and Animation

The sizzle of JAVA is MULTIMEDIA - Sounds, Images, Graphics and Video. In this growing age of multimedia, new computers are known as "multimedia ready" with CD-Rom drives, sound cards, 3D accelerator cards and other new special sound or graphic technology capabilities. Multimedia demands incredible computing power and only recently - in the past 5 years at least, affordable computers of this kinds are becoming widespread.

Among the image formats supported by Java is the Graphics Interchange Format .GIF and Joint Photography Experts Group .JPEG. Among the audio formats are AIFF, AU and WAV.

(vii) The Virtual Machine: Java VM

This VM sits, metaphorically, between the Java program and the machine it is running on, offering the program an "abstract computer" that executes the Java code and guarantees certain behaviors regardless of the underlying hardware or software platform. Java compilers thus turn Java programs not into assembly language for a particular machine but into a platform-neutral "byte code" that the machine-specific VM interprets on the fly.

The Java VM also enforces security policies, providing a sandbox that limits what the Java program can do. A Java applet cannot, for example, peek into arbitrary files on the machine it's running on. The most recent version of Java from Sun, known as Java Development Kit (JDK) 1.6, though, provides no consistent method for an applet to request restricted system resources.

(viii) Java is Portable

One advantage of Java is that its programs can run on any platform without having to be recompiled. This is one positive aspect of portability. It goes on even further to ensure that there are no platform-specific features on the Java language specification. In Java, the size of the integer is the same on every platform, as is the behavior of arithmetic. Having a fixed size for numbers makes Java programs portable. The Java environment itself is portable to new hardware and operating systems, and in fact, the Java compiler itself is written in Java.

(ix) Java is Dynamic

The Java programming language was designed to adapt to an evolving environment. New methods and properties can be added freely in a class without affecting their clients. Also, Java is able to load classes as needed at runtime. As an example, you have a class called 'Square'. This class has a property to indicate the color of the square, and a method to calculate the area of the square. You can add a new property to the 'Square' class to indicate the length and width of the square, and a new method to calculate the perimeter of the square, and the original client program that uses the 'Square' class remains the same.

1.2.1. Experimental Setup for working on Java Platform

Step 1: Unzip the folder.

Step 2: Run the jdk-6-windows-i586 setup file.

Step 3: Open the console Window.

Step 4: Reach the specified path where all the java files exist (C: /java/bin).

Step 5: Compile all java files using javac <filename>.java.

Step 6: Run the files using java <filename>.

Step 7: Result is obtained.

1.3. Scripting Languages

There have been two scripting languages used in this project.

- HTML (Hypertext Markup Language)

-XML (Extensible Markup Language)

1.3.1. HTML

This case-insensitive language enables users to present information over the web in a structured and uniform fashion. It is used to markup documents so that a web browser can interpret and display them. It refers to the html code that defines the elements in HTML file like headings, images, lists etc.

(i) The <HTML> tag

It defines the HTML document itself while all the other tags and text are nested in it.

(ii) The <HEAD> tag

It contains information about the HTML file and the title tag is nested in it.

(iii) The <TITLE> tag

It identifies the HTML page being made. It displays the same on the browser's title bar and does not appear as part of the HTML page.

(iv) The <BODY> tag

It is the compliment of the head tag. It contains all the tags and elements that the browser displays as the body of the HTML documents.

***(v) The
 tag***

It is an empty or standalone tag that simply inserts a line break

(vi) The <H2> and < H3> tag

This is used to highlight a given piece of text according to the heading size been specified wherein the text under the <H2> tag is bigger than the one under the <H3> tag and so on.

(vii) The <MARQUEE> tag

This is used to display text in motion which can be set to move in upward or downward direction as well as leftwards or rightwards.

(viii) The <IMG> tag

This is used to display an image of a specified height and width at a certain place in the HTML document. The only factor to be taken into consideration is that the format of the image should be explicitly specified (e.g. .gif, .jpg etc.) and the image should lie in the same folder as the HTML document.

Usage

The HTML page been created in our project basically is used to pictorially display the whereabouts of the project. A brief introduction of the example of this project, that is, the Intelligent Automatic Teller Machine is shown on the document in the heading tag <H2> followed by the MARQUEE tag.

1.3.2. XML

XML is a markup language used to define data much like HTML except that in XML the tags are not predefined but defined by the user according to the usage. XML was designed to describe data and to focus on what data is. Where on one hand, HTML is

about displaying information, on the other, XML is about describing information. XML can be used to store data in files or in databases and also as a format for exchange of information.

(i) XML declaration

Value version- Indicates the XML version to which the document conforms.

(ii) Root element

It is the element that encompasses every other element.

(iii) Container element

It refers to any element that contains other elements.

(iv) Child elements

The elements inside a container element are called child elements.

Usage

In our project XML has been used to dynamically store 2 latest mouse clicks on 2 states of the Finite State Automata for a particular transaction of the ATM. Incase the combination between them is possible a message is generated to notify a VALID transaction else it warns of an INVALID transaction.

The XML Parser function in Java

```
public static String Value(String file) throws Exception {
        IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
        IXMLReader reader = StdXMLReader.fileReader(file);
        parser.setReader(reader);
        XMLElement xml = (XMLElement) parser.parse();
        return xml.getContent();
}
```

This Parser is used to read values from a particular XML document. In our project the values are read from two files MouseClick1.xml and MouseClick2.xml respectively and then the coordinates stored in them after two mouse clicks are compared. Incase they make a valid combination they are termed as Interaction Pairs else they make Faulty Interaction pairs. An object xml is created which parses through a file reading its content and thereby returning the content back through xml.getContent().

Format of the XML document

<? xml version="1.0" encoding="ISO-8859-1"?>

<Click>coordinate1, coordinate2</Click>

Here the tag <Click> is a user specified XML tag and not a predefined one suited for the purpose of storing mouse clicks.

1.4. Formal Languages and Tools

1.4.1. Automata

An Automaton is an abstract model of a digital computer. As such, every automaton every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet written on an input file which the automaton can read but, not change. The input file mechanism can also detect the end of the input string (by sensing end of file condition) and can produce out put of some form. It may have a temporary storage device, capable of holding a single symbol from an alphabet. The automaton also has a control unit which can be in any one of the finite number of internal states and which can change state in some defined manner.

Two types of Automata

(i) Deterministic Automata

(ii) Non Deterministic Automata

(i) Deterministic Automata:

A Deterministic Automata is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input and the contents of temporary storage, we can predict the future behavior of the automata exactly.

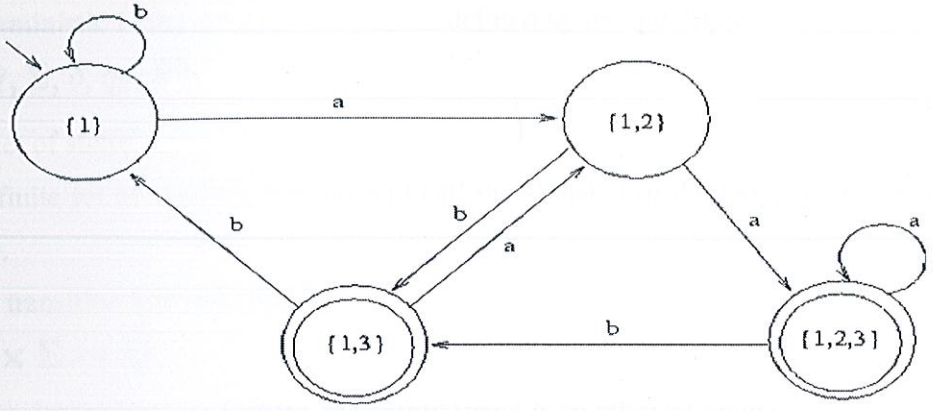


Fig 1.1 – Deterministic Automata

(ii) Non Deterministic Automata:

In a Non Deterministic Automaton at a single point there may be several possible moves so, we can only predict a set of possible actions.

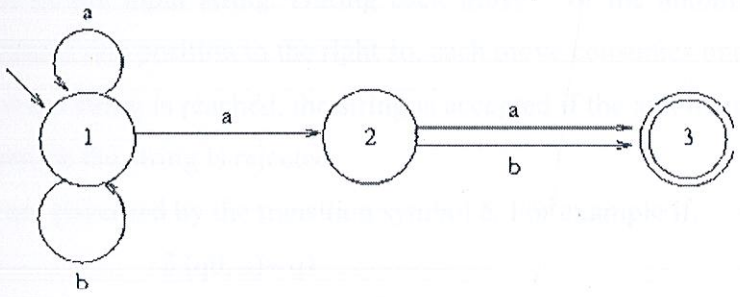


Fig 1.2 –Non-Deterministic Automata

(iii) Finite State Automata

This type of an automaton is characterized by having no temporary storage. Since an input file cannot be re written, a finite automaton is severely limited in its capacity to remember things during the computation. A finite amount of information can be retained in the control unit but, since the number of such states is finite a finite state automaton can only deal with situations in which information to be stored at any time is strictly bounded.

Two types of Finite State Automata

1. Deterministic Finite Acceptors

2. Non Deterministic Finite Acceptors

Deterministic Finite Acceptors

A Deterministic Finite Acceptor of DFA is defined by the quintuple

M= $(Q, \Sigma, \delta, q_0, F)$

Q is a set of states.

Σ is a finite set of symbols that we will call the alphabet of the language the automaton accepts.

δ is the transition function, that is

$\delta : Q \times \Sigma \rightarrow Q$.

(For non-deterministic automata, the empty string is an allowed input).

q0 is the start state, that is, the state in which the automaton is when no input has been processed yet (Obviously, q0□ Q).

F is a set of states of Q (i.e. F□Q), called accept states.

At the initial time, it is assumed to be in the initial state with its input mechanism on the leftmost symbol of the input string. During each move of the automaton, the input mechanism advances one position to the right so, each move consumes one input symbol. When the end of the string is reached, the string is accepted if the automaton is one of the final states otherwise the string is rejected.

The transitions are governed by the transition symbol δ. For example if,

$$\delta (q0, a) = q1$$

then if the DFA is in state q0 and the current input symbol is a the DFA will go into state q1.To visualize and represent the finite state automata, we use transition graphs, in which the vertices represent states and the edges represent transitions. The labels on the vertices are the states while the labels on the edges are the current values of the input string.

For example, if 1 and 2 are the internal states of some DFA M, then the graph associated with M will have one vertex labeled 1 and another 2. An edge (1, 2) labeled a represents transition

$$\delta (1, a) = 2$$

The initial state will be identified by an incoming unlabeled arrow not originating at any vertex. Final States are drawn with a double circle.

Non Deterministic Finite Acceptors

Nondeterminism means a choice of moves for an automaton. Rathe than prescribing a unique move in each situation, we allow a set of possible moves. Formally, we achieve this by defining the transition function so that it ranges over a set of possible states.

A non deterministic finite acceptor if defined by the quintuple

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

Where Q, $\sum$, q0, F are defined in a similar way as for deterministic automaton

Here, the transition function if defined as

$$\delta: Q \times (\sum \cup \{\lambda\}) \rightarrow 2Q$$

There are three major differences between the definition of a deterministic finite state automaton and of a non deterministic one. The range of δ is in the power set 2Q. Also, we allow λ as the second argument of δ. This means that an NFA can make a transition without consuming an input symbol. Although we still assume that the transition is towards right, it is possible that it is stationary on some moves. Finally, in an NFA, the set δ (qi, a) may be empty meaning that there is no transition defined for this specific situation.

Like DFA's, non deterministic acceptors can be represented by transition graphs. The vertices are determined by Q, while the edge (qi, qj) with the label a is in the graph if and only if δ (qi, a) contains qj. Note that since a may be the empty string, there may be some edges labeled λ.

A string is accepted by an NFA if there is some sequence of possible moves that will put the machine to its final state at the end of the string. A string is rejected if only if there is no possible sequence of moves by which the final state can be reached.

Nondeterminism can therefore be viewed as involving "intuitive" insight by which the best move can be chosen for every state (assuming that the NFA wants to accept the string).

1.4.2. Languages and DFA

The language accepted by a DFA M= $(Q, \Sigma, \delta, q_0, F)$ is the set of all strings on Σ accepted by M. In formal notation,

L(M) = { w ϵ Σ^* : δ^* (q0 , w) ϵ F }

A DFA will process every string in Σ^* and either accept it or not accept it.

Non acceptance means that the DFA stops at a non final state, such that

L(M) = { w ϵ Σ^* : δ^* (q0 , w) F }

1.4.3. Regular languages and Regular expressions

Regular language can be obtained from the basic languages using the union, concatenation and kleene (*) operations. A regular language can be represented by a simple form called a regular expression.

CHAPTER 2

MODELING AND DESIGN OF GUI'S

2.1 Underlying Idea

Modeling of a system requires the ability of abstraction, extracting the relevant issues and information from the irrelevant ones, taking the present stage of the system development into account. While modeling a GUI, the focus is usually addressed rather to the correct behavior of the system as desired situations, triggered by legal inputs. Describing the system behavior in undesired, exceptional situations which will be triggered by illegal inputs and other undesired events are likely to be neglected, due to time and cost pressure of the project. The precise description of such undesired situations is, however, of decisive importance for a user-oriented fault handling, because the user has not only a clear understanding how his or her system functions properly, but also which situations are not in compliance with his or her expectations. In other words, we need a specification to describe the system behavior both in legal and illegal situations, in accordance with the expectations of the user. Once we have such a complete description, we can then also precisely specify our hypotheses to detect undesired situations, and determine the due steps to localize and correct the faults that cause these situations.

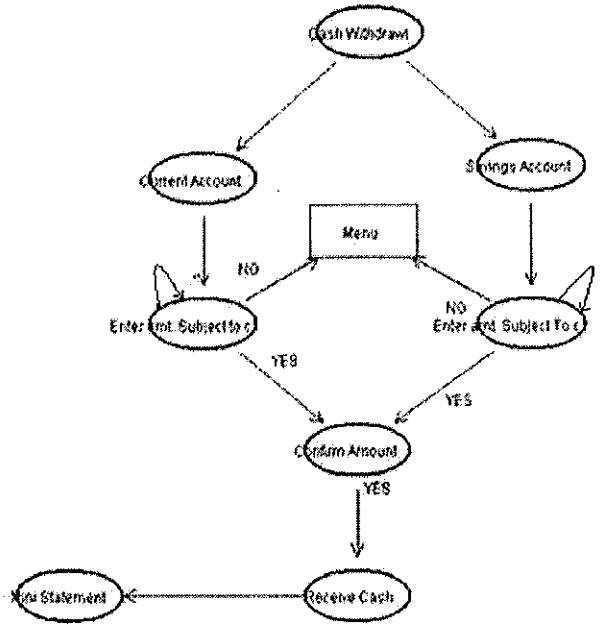
Finite State Automata are broadly accepted for the design and specification of sequential systems for good reason. First, they have excellent recognition capabilities to effectively distinguish between correct and faulty events/situations. Moreover, efficient algorithms exist for converting FSA into equivalent regular expressions (RegEx). RegEx, on the other hand, are traditional means to generate legal and illegal situations and events systematically.

A FSM can be represented by

- A set of inputs.
- A set of outputs.
- A set of states.
- An output function that maps pairs of inputs and states to outputs.
- A next-state function that maps pairs of inputs and states to next states.

Any chain of edges from one vertex to another one, materialized by sequences of user inputs states- triggered outputs defines an interaction sequence (IS) traversing the FSA from one vertex to another.

Once the FSA has been constructed, more information can be gained by means of its state transition graph. First, we can identify now all legal sequences of user-system interactions which may be complete or incomplete, depending on the fact whether they do or do not lead to a well-defined system response that the user expects the system to carry out (Please note that the incomplete interaction sequences are sub-sequences of the complete interaction sequences). Second, we can identify the entire set of the compatible, i.e. legal interaction pairs (IP) of inputs as the edges of the FSA. This is key issue of the present approach, as it will enable us to define the edge coverage notion as a test termination criterion. We start the designing process with the example of an Automatic Teller Machine automaton including all states. Its sub automatons are shown one by one as under.



Legend
C: *Cash Withdrawl*
C1: *Current Acc.*
S: *Savings Acc.*
M: *Menu*
E: *Enter amt subject to c1*
E1: *Enter amt subject to c2*
C2: *Confirm Amt.*
R: *Receive cash*
M1: *Mini Statement*

Fig 2.1: Sub Automata showing Cash Withdrawal

Table 2.1- IPs and FIPs of Cash Withdrawal

Sub Graph	Interaction Pairs	Faulty Interaction Pairs
Cash Withdrawal.	CC1, CS, C1E, EM, EC2, C2R, RM1, SE1, E1M, E1C2.	C1C, SC, EC1, ME, C2E, RC2, M1R, E1S, ME1, C2E1, CC2, CR, CM, ME, ME1, MM, RR, E1E1, EE, C2C2, CC, SS, C1C1.

Table 2.2- Regular Expression for Cash Withdrawal

Sub Graph	Regular Expression
Cash Withdrawal.	$(CC1E^{+} + CSE1^{+})M+(CC1E^{+} + CSE1^{+})C2RM1$

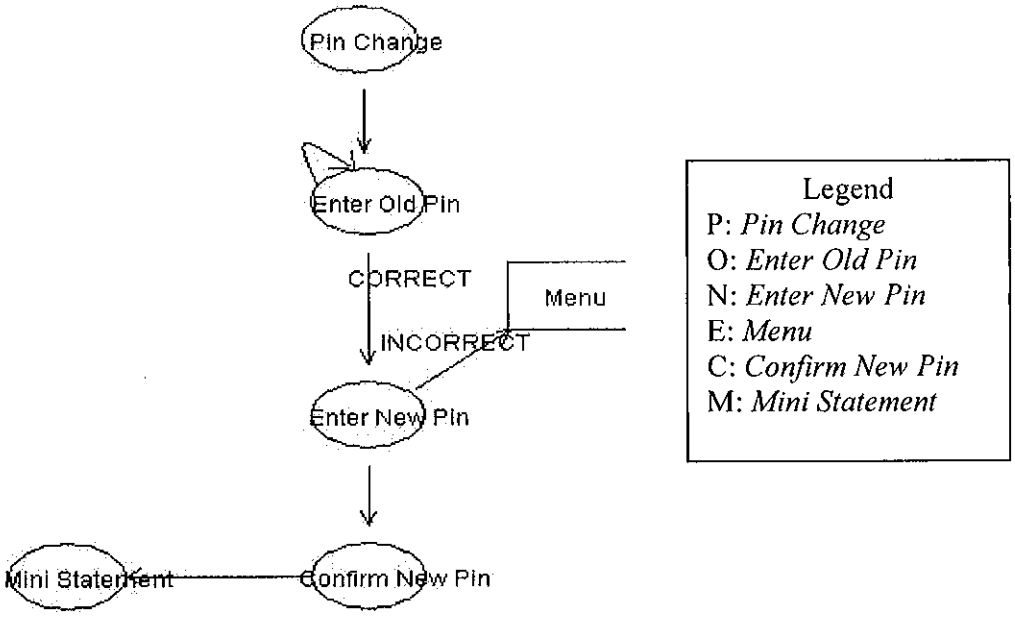


Fig 2.2: Sub Automata showing PIN Change

Table 2.3- IPs and FIPs of PIN Change

Sub Graph	Interaction Pairs	Faulty Interaction Pairs
PIN Change	PO, ON, NC, CM, NE.	PP, OO, NN, EE, CC, MM, OP, NO, CN, MC, EN, MP, MO, MN, ME, CP, CO, NP, PM, PN, PC, PE, OC, OM, EP, EO, EC, EM.

Table 2.4- Regular Expression for PIN change

Sub Graph	Regular Expression
PIN Change	$(PO^{+}NCM + PO^{+}NE)$

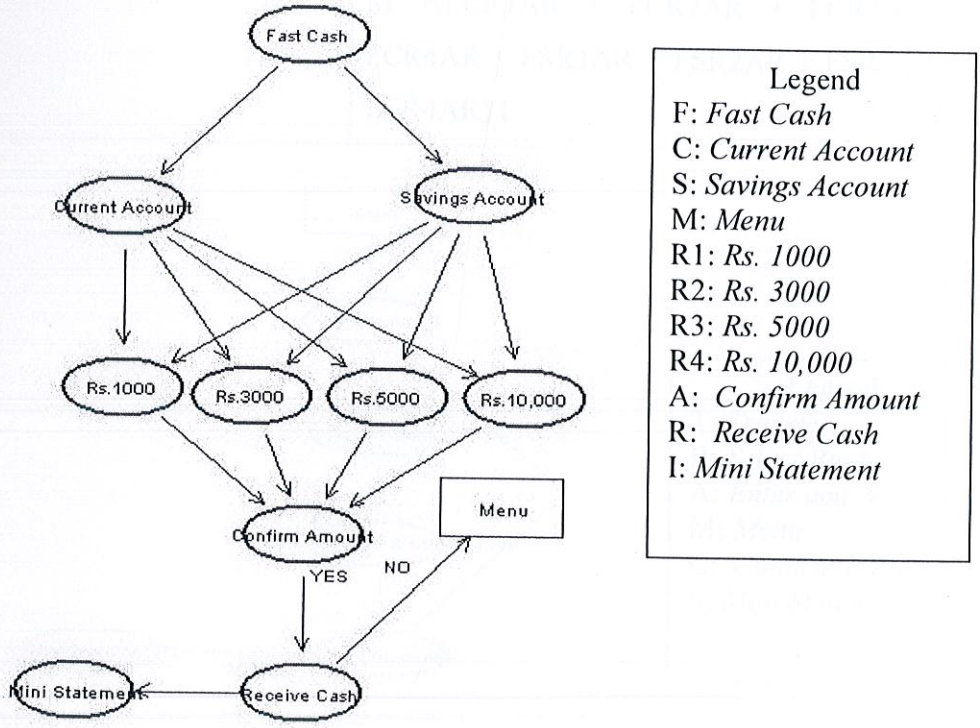


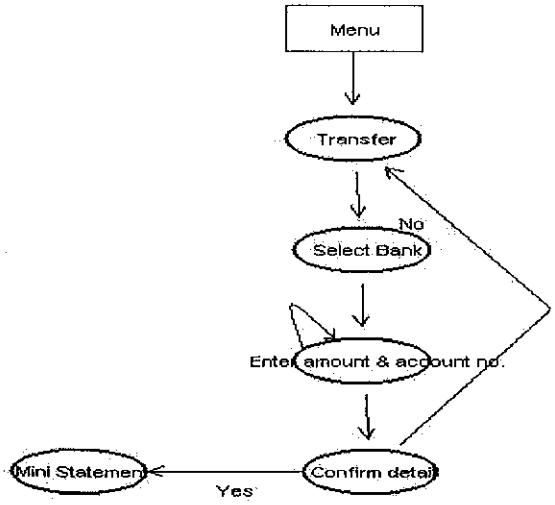
Fig 2.3: Sub Automata showing Fast Cash

Table 2.5-IPs and FIPs of Fast Cash

Sub Graph	Interaction Pairs	Faulty Interaction Pairs
Fast Cash	FC, FS, CR1, CR2, AR CR3, CR4, SR1, RM, SR2, RI, SR3, SR4, R1A, R2A, R3A, R4A .	FF, CC, SS, R1R1, R2R2, R3R3, R4R4, MM, AA, RR, II, CF, SF, R1C, R2C, RA, R3C, R4C, R1S, MR, R2S, IR, R3S, R4S, AR1, AR2, AR3, AR4, MF, MC, MS, MR1, MR2, MR3, MR4, RF, RC, RS, RR1, RR2, RR3, RR4, IR1, IR2, IR3, IR4, IM, IF.

Table 2.6- Regular Expression for Fast Cash

Sub Graph	Regular Expression
Fast Cash	(FCR1AR + FCR2AR + FCR3AR + FCR4AR + FSR1AR + FSR2AR + FSR3AR + FCR4AR) M +(FCR1AR + FCR2AR + FCR3AR + FCR4AR + FSR1AR + FSR2AR + FSR3AR + FCR4AR) I



Legend
T: *Transfer*
B: *Select Bank*
A: *Enter amt & acc*
M: *Menu*
C: *Confirm Detail*
S: *Mini Statement*

Fig 2.4: Sub Automata showing Transfer

Table 2.7- IPs and FIPs of Cash Transfer

Sub Graph	Interaction Pairs	Faulty Interaction Pairs
Transfer	MT, TB, BA, AC, CS.	MM, TT, BB, AA, CC, SS, TM, BT, AB, CA, SC, SM, ST, SB, SA, CM, CT, CB, AM, AT, AS, BM, BC, BS, TA, TC, TS, MS, MC, MA, MB.

Table 2.8- Regular Expression for Cash Transfer

Sub Graph	Regular Expression
Transfer	$(MTBA^{+}C)S + M\ (TBA^{+}C)^{+}$

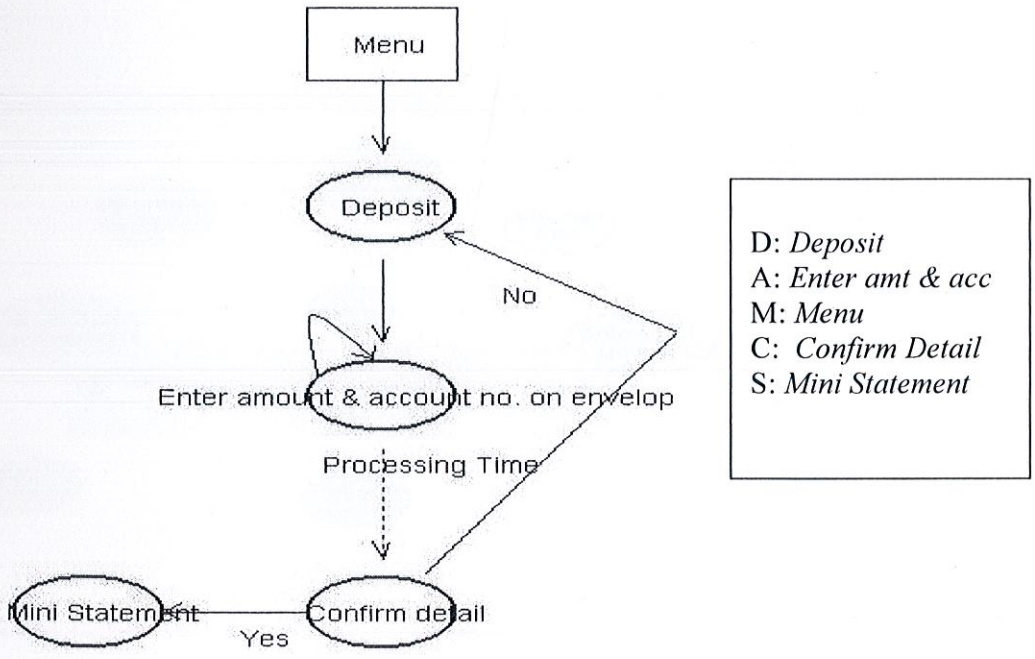


Fig 2.5: Sub Automata showing Deposit

Table 2.9-IPs and FIPs of Cash Deposit

Sub Graph	Interaction Pairs	Faulty Interaction Pairs
Deposit	MD, DA, AC, CS.	MM, DD, AA, CC, SS, DM, AD, CA, SC, SM, SD, SA, CM, CD, AS, AM, DS, DC, MS, MC, MA.

Table 2.10-Regular Expression for Cash Deposit

Sub Graph	Regular Expression
Deposit	$(MDA^{+}C)S + M\,(DA^{+}C)^{+}$

2.2 Demonstration of Interaction Pairs (IPs) and Faulty Interaction Pairs (FIPs)

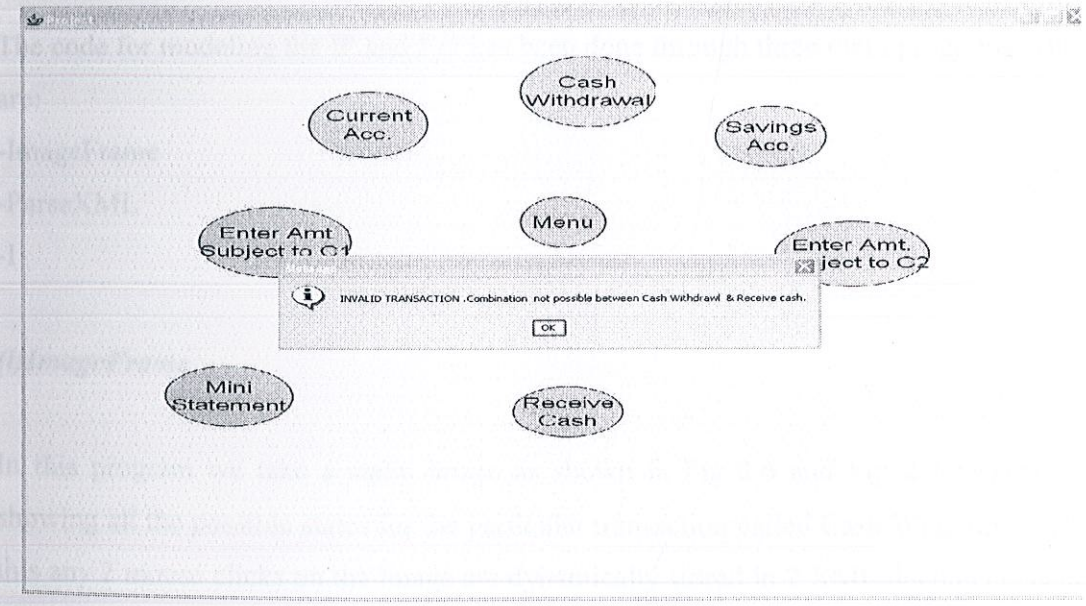


Fig 2.6- Snapshot of an FIP in Cash Withdrawal

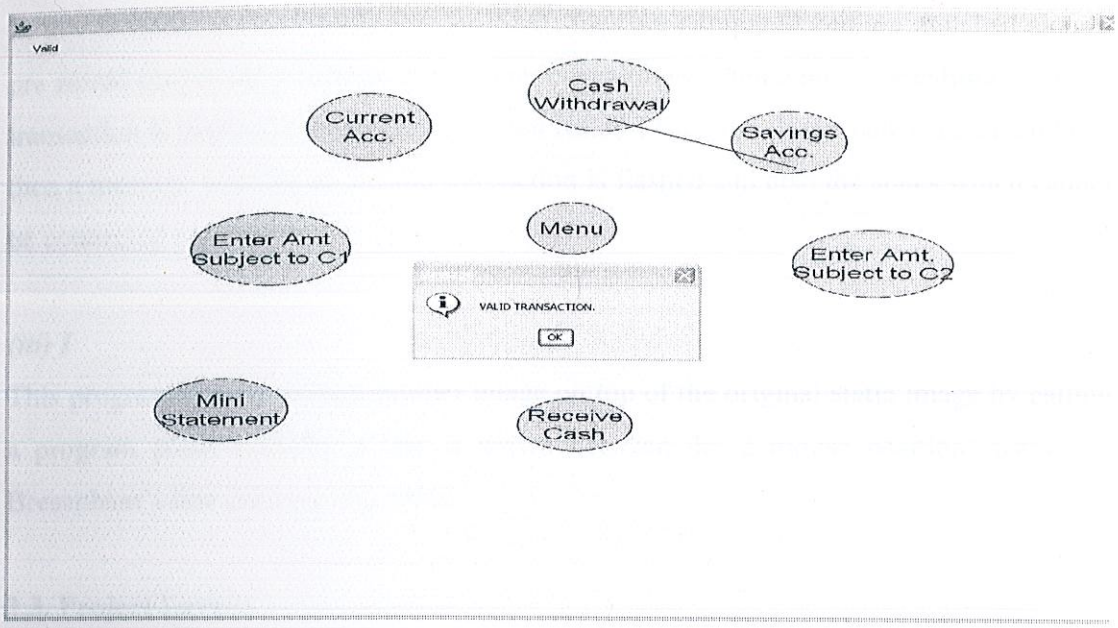


Fig 2.7- Snapshot of an IP in Cash Withdrawal

2.2.1. Programs being used:

The code for modeling the IP and FIP has been done through three main programs. These are:

-ImageFrame

-ParseXML

-I

(i)ImageFrame

In this program we take a static image as shown in Fig 2.6 and Fig 2.7 respectively showing all the possible states for the particular transaction called Cash Withdrawl. After this any 2 mouse clicks on the image are dynamically stored in 2 XML documents called MouseClick1.xml and MouseClick2.xml. Thereby a program called ParseXML is called.

(ii) ParseXML

In this program the coordinates in the XML files are compared and are matched against pre stored coordinate positions. If they made a valid pair then a message calling it a valid transaction is displayed and a new program called I is called. If the pair is an invalid one then a message warning an invalid transaction is flashed and also the states which cannot be connected are notified.

(iii) I

This program is used to load another image on top of the original static image by calling a program point whereby a line is drawn between the 2 mouse positions using the Bresenham's line drawing algorithm.

2.3. Project Layout

2.3.1. Help Menu

Layout

Contents → *Cash Withdrawal*
Fast Cash
Cash Transfer
Cash Deposit
PIN Change

About

Image

(i) Contents

A transition state diagram view of the automaton for the transactions possible in the ATM is explained in this part of the software. An explanation for each part is given in this part where each state is traversed to show the valid transaction possible. Each transaction begins with the start state as "Ready" and ends up with the state "Mini Statement". Each help menu item displays the detail for that transaction and provides a button on clicking of which the full page view of that transaction if given.

(a) Cash Withdrawal:

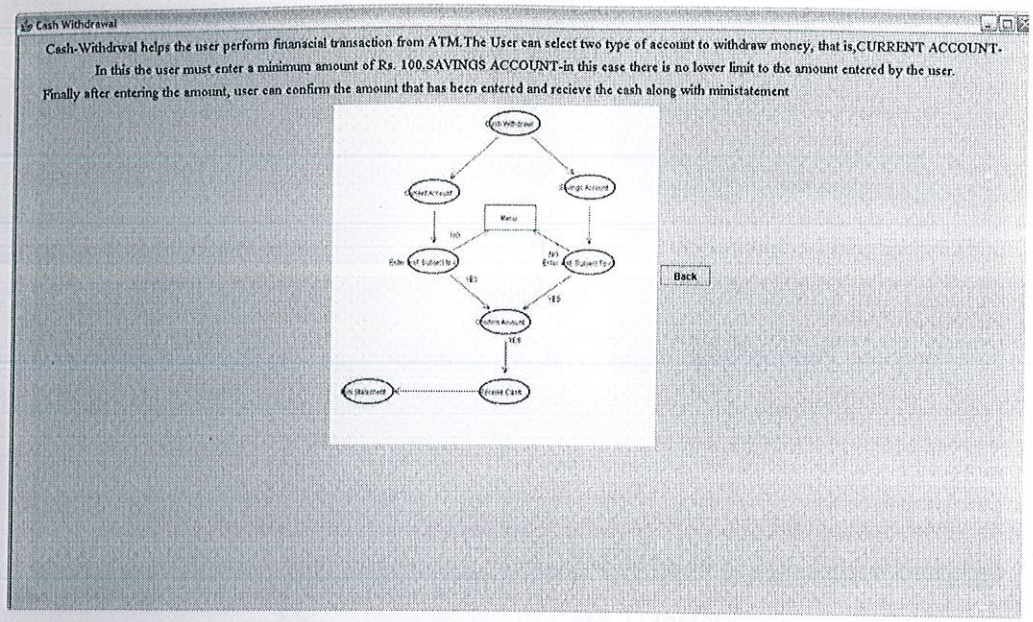


Fig 2.8- Snapshot of Cash WithDrawl

(b) Fast Cash

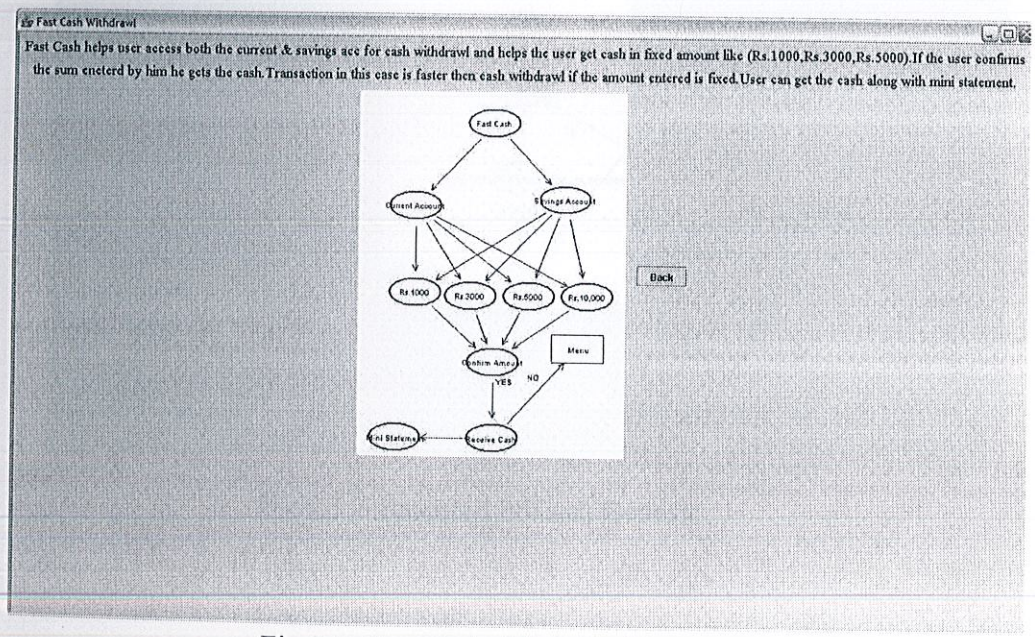


Fig 2.9- Snapshot of Fast Cash

(c) Cash Transfer

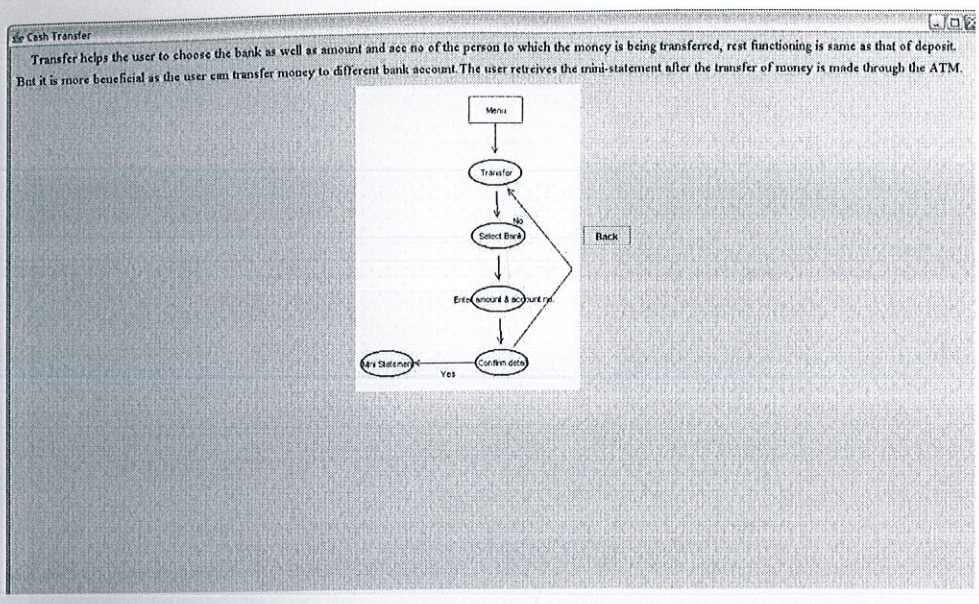


Fig 2.10- Snapshot of Cash Transfer

(d) Cash Deposit

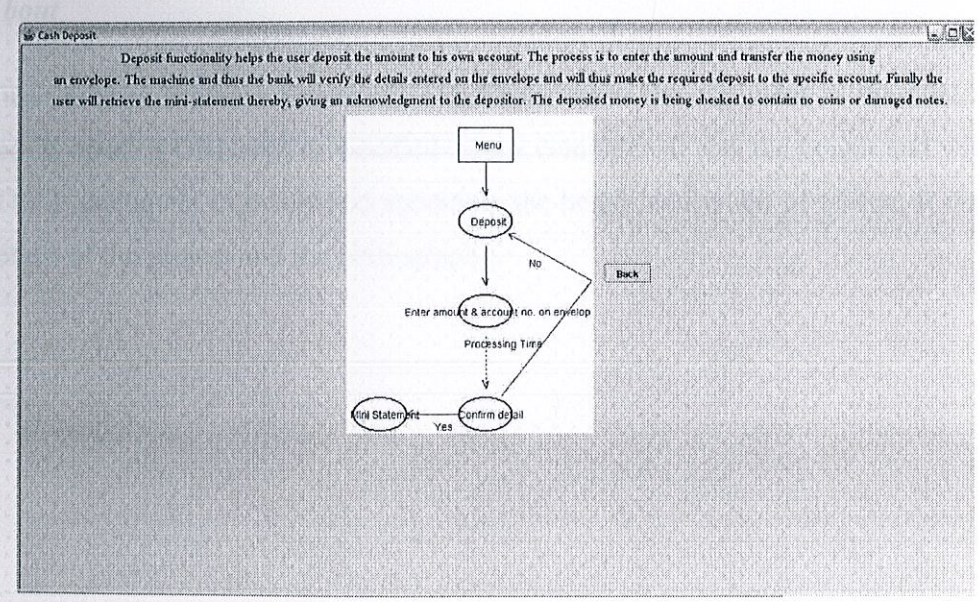


Fig 2.11- Snapshot of Cash Deposit

(e) PIN Change

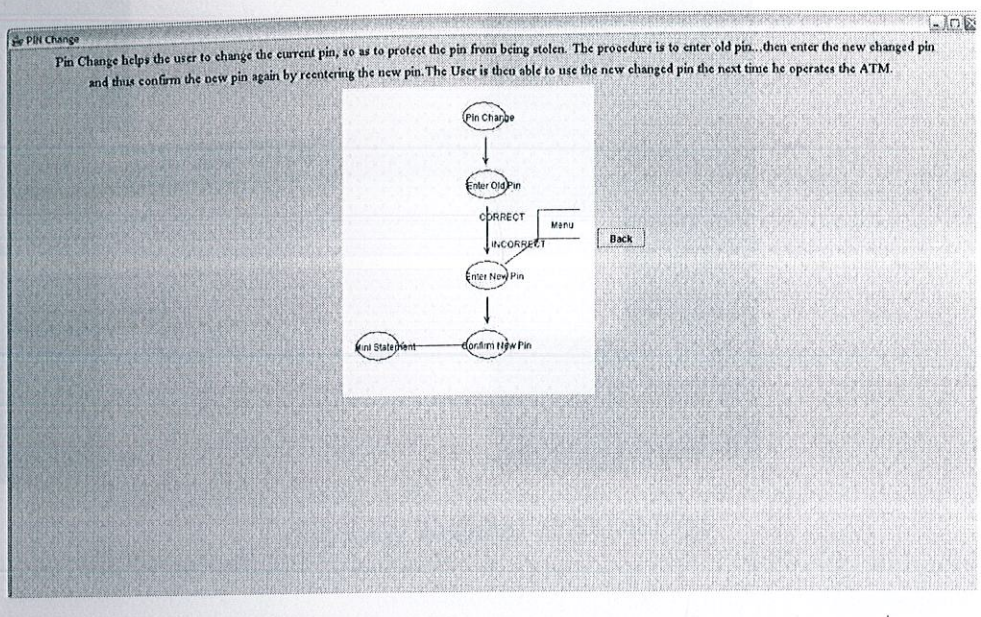


Fig 2.12- Snapshot of PIN Change

(ii) About

This uses a class Browser. Browser's constructer takes the argument as the URL to be opened. It opens a URL that is specified on the computer. It sets the height and width of the HTML document to be opened according the height and width of screen. It displays the details of the project and the bibliography.

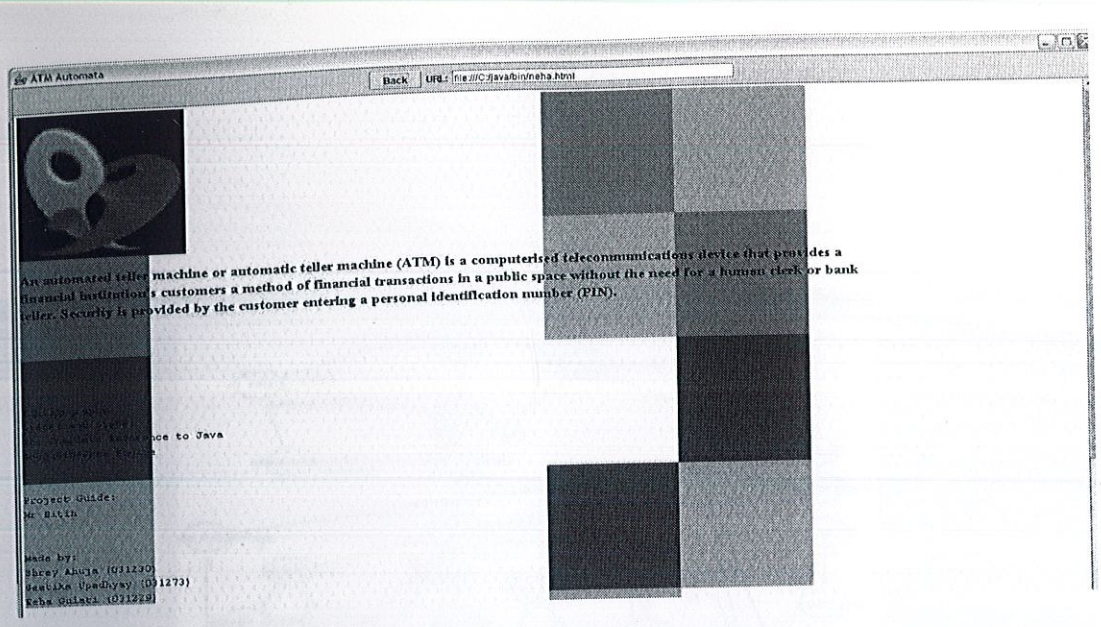


Fig 2.13- Snapshot of About Page

(iv) Image

It uses the class MyImage. It shows the complete transition diagram for the ATM Automaton. This is based on functionality that on click of button, the complete automaton is displayed for viewing purpose only.

The automaton shows the transitions from one state to another and links only those states which can be reached in sequential order. A path from one initial state till the whole path is covered and the final state is reached forms the complete interaction sequence while only two connecting states form a pair. The states that cannot be linked are termed as faulty Interaction Pairs.

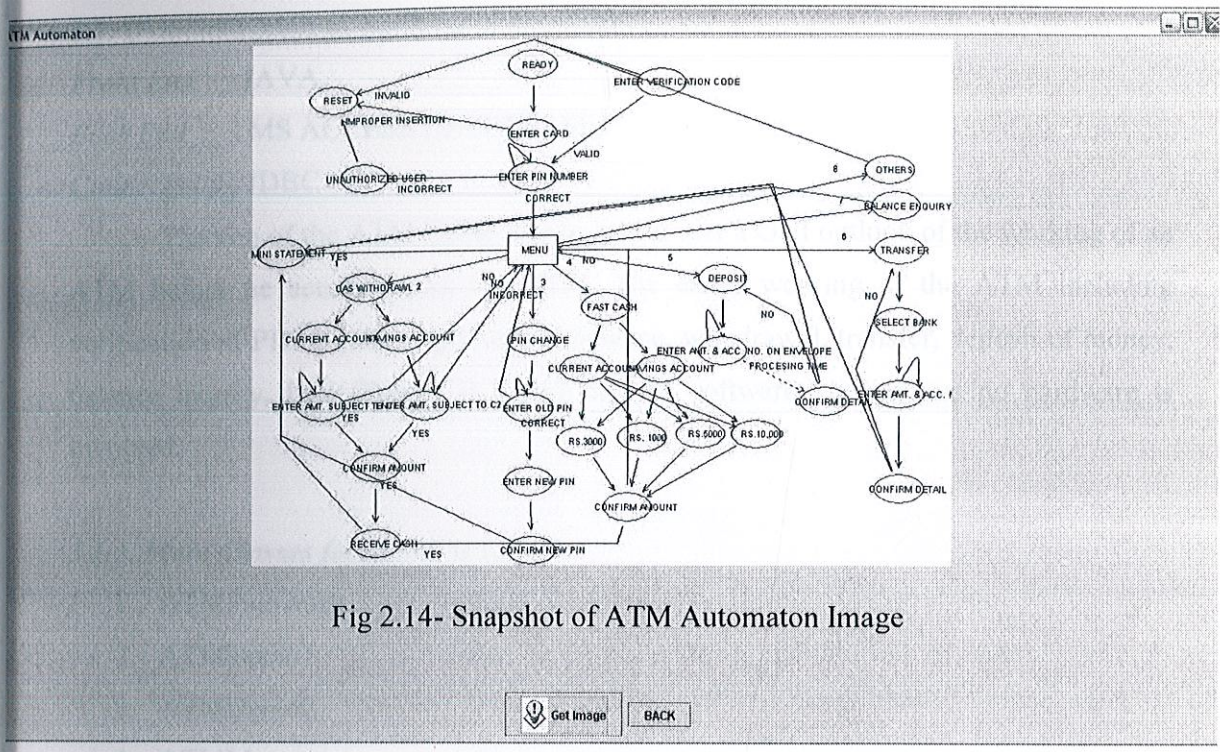


Fig 2.14- Snapshot of ATM Automaton Image

CHAPTER 3

TESTING OF GUI

3.1. Demonstration of working of an ATM

Front End : JAVA

Back End : MS ACCESS (DATABASE)

Connectivity : JDBC

The aim of the ATM Demo was to give a user a GUI outlook of the working of an ATM before he actually uses the ATM. The entire working of the ATM including verification of PIN number, updating of balance, withdrawal, transfer, deposit of money, balance inquiry, PIN Change is shown from a software view where no hardware is involved.

3.1.1. Main Classes for an ATM Demo

1. ATMPassword
2. ATMDemo1
3. ATMDemo2
4. ATMMenu
5. ATMCash
6. ATMSave
7. ATMCurrent
8. ATMMini
9. ATMDeposit
10. ATMTransfer
11. TransferLogin
12. ATMAmt
13. CardAuthenticate
14. ATMPIN
15. ATMFast

(i) ATMPassword

This class authenticates the administrator into usage of the ATM Demo.

(a) Name of Database Accessed:

Password

(b)Attributes of Password:

Login, Password

(c) Input:

Login, Password

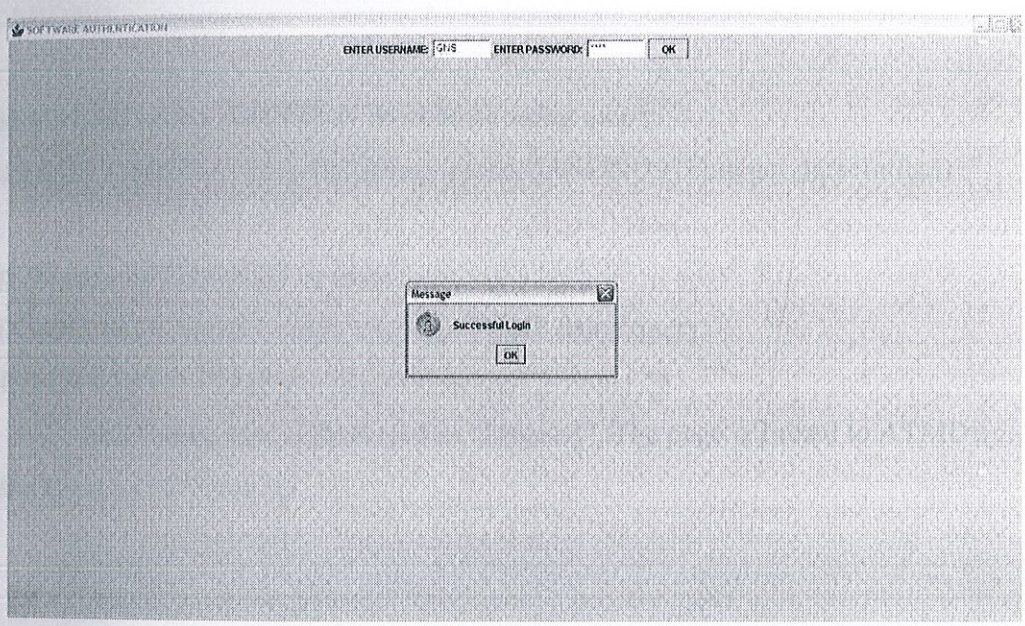


Fig 3.1-Snapshot of the login/password window

(ii) ATMDemo1

This class takes the input as the CARD NUMBER which is verified as a VALID CARD NUMBER from thedatabase and if in case the card is valid, the user is sent to another page to test the PIN number and so on.

***(a) Name of Database Accessed*:**

BankData, TEMP

(b) Attributes of BankData:

CARDNO(primary key), PIN, ACCOUNT NO, NAME, BALANCE IN CURRENT ACCOUNT, BALANCE IN SAVINGS ACCOUNT

(c) Attributes of TEMP:

CARDNO (primary key)

(d) Input:

CARDNO

(e) Verified with CARDNO in BankData using query:

SELECT CARDNO from BankData where CARDNO="+Integer.parseInt(res)+"

(f) Storage of CARDNO in num:

If CARDNO is correct, value stored in TEMP using query:

"INSERT INTO TEMP (CARDNO) values('"+num+"')

And Dialog box is displayed as "Proceed". The page is linked to ATMDemo2 to take input as PIN number.

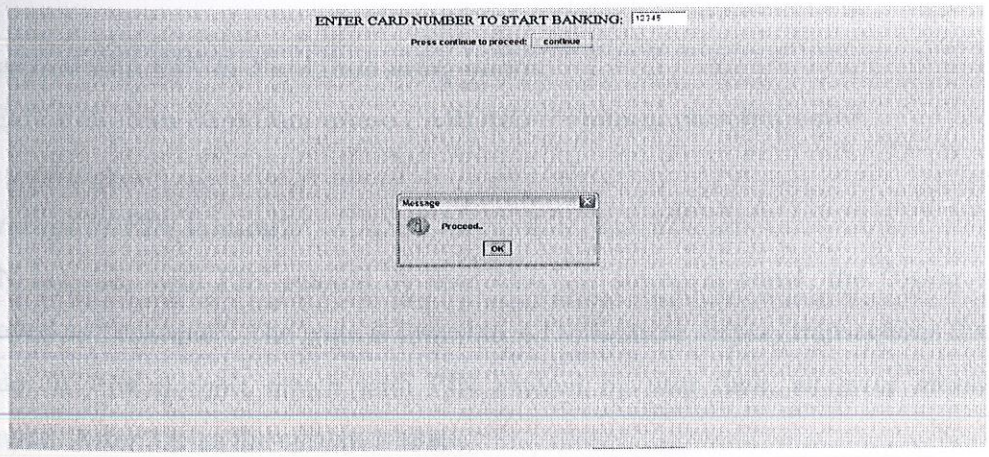


Fig 3.2- Snapshot for entering card no.

(iii) ATMDemo2

This class takes the input as the PIN which is verified as a VALID PIN from the database and if in case the PIN is valid, the user is sent to another page to access the services provided by the bank such as withdrawal, transfer, deposit of money, balance inquiry, PIN Change.

(a) Name of Database Accessed:

BankData, TEMP

(b) Attributes of BankData:

CARDNO(primary key), PIN, ACCOUNT NO, NAME, BALANCE IN CURRENT ACCOUNT, BALANCE IN SAVINGS ACCOUNT

(c) Attributes of TEMP:

CARDNO (primary key)

(d) Input:

PIN

(e) Input taken (using buttons, no user keyboard input is accepted):

Value Stored in res = text.getText();

(f) Verified with PIN in BankData using query:

SELECT PIN from BankData where CARDNO="+Integer.parseInt(res)+"

(g) Storage of PIN in num2:

If PIN matches with PIN entered by user which is stored in num1, the Dialog box is displayed as "Proceed". The page is linked to ATMMenu to display transactions that user can do. If PIN doesn't match with PIN entered by user then, an error message is displayed. After 3 tries the system is reset.

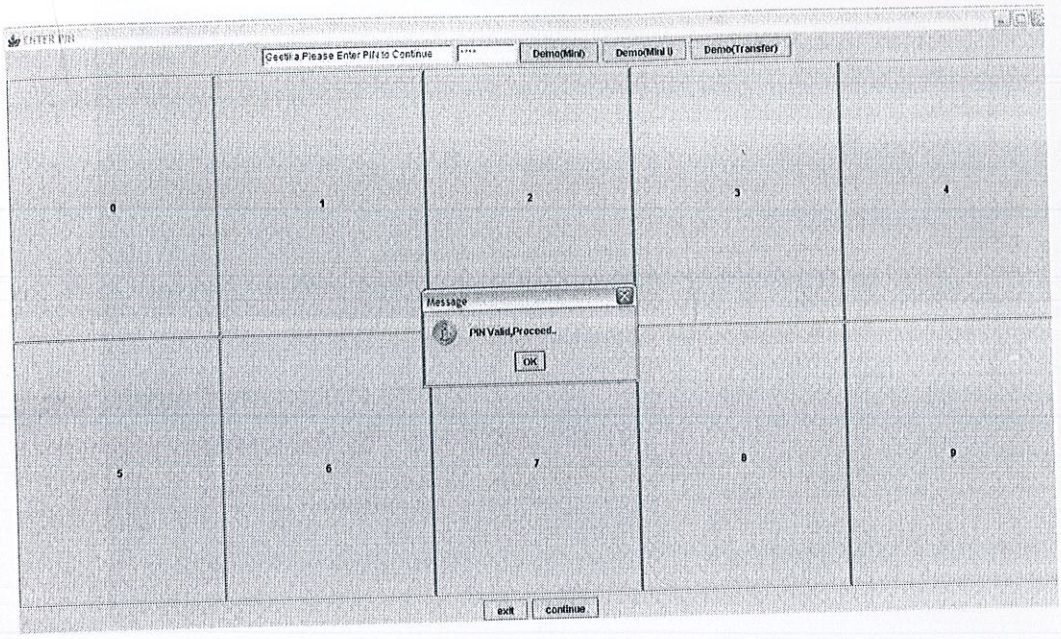


Fig 3.3- Snapshot for entering the PIN number

(iv) ATMMenu

This class gives the user the various options that the user can access, that of:

Cash Transfer (to accounts in the two banks accessible by database)

Cash Deposit

Cash Withdrawal

-Savings Account

-Current Account

Fast Cash Transfer

PIN Change

Balance Enquiry

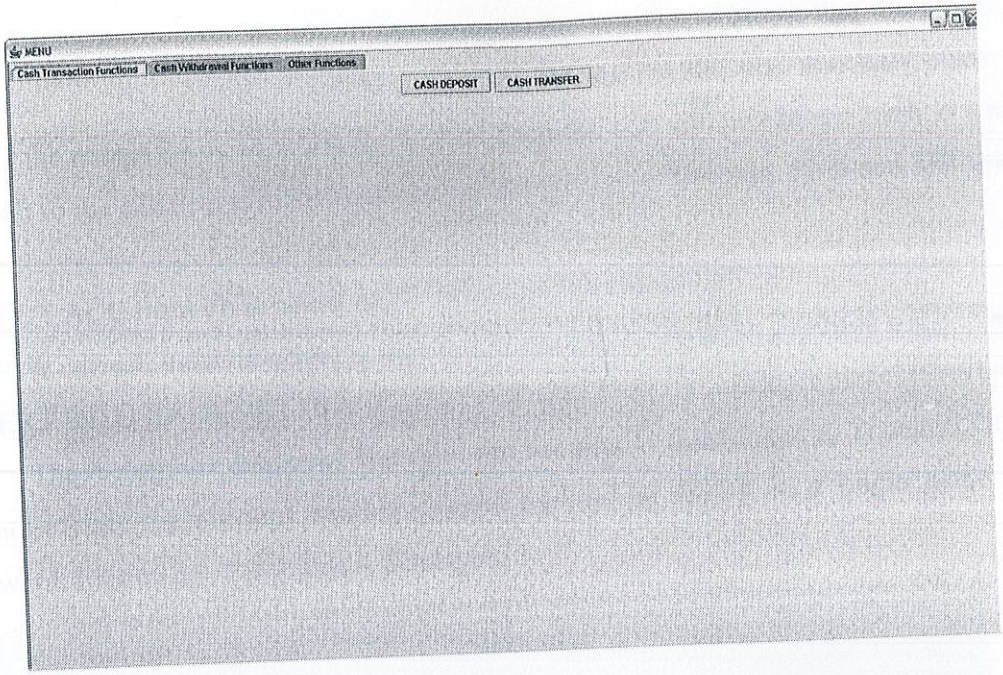


Fig 3.4- Snapshot of displaying the Menu for an ATM

(v) ATMTransfer

This class takes the input as the Bank and the account number to which the amount is to be transferred which is then verified to the existing database of banks with account numbers.

(a) Name of Database Accessed:

BankDet, PNB, SBI

(b) Attributes of BankDet:

ACC, Bank

(c) Attributes of PNB, SBI:

ACC, LOGIN, Password, CARDNUM, NAME, Balance

(d) Input:

Bank, Account no

ATMTransfer has no. of conditions according to the number of banks that the database has access to. The system proceeds depending on the BANK that that user plans to transfer the money to. It takes the account no from the user stores it in res, bank in s and checks it to the existing account numbers for that particular database, PNB and SBI using query:

"SELECT ACC from PNB where ACC="+Integer.parseInt(res)+"

If ACC is correct, value is stored in res1 and is inserted in BankDet database using query:

"INSERT INTO BANKDET (BANK, ACC) values ('"+s+" ', '"+Integer.parseInt(res1)+"')

The dialog box displays "Proceed" and the page is then linked to TransferLOGIN to authenticate the account number entered by user by taking in a Login name and password. Else an error message is displayed.

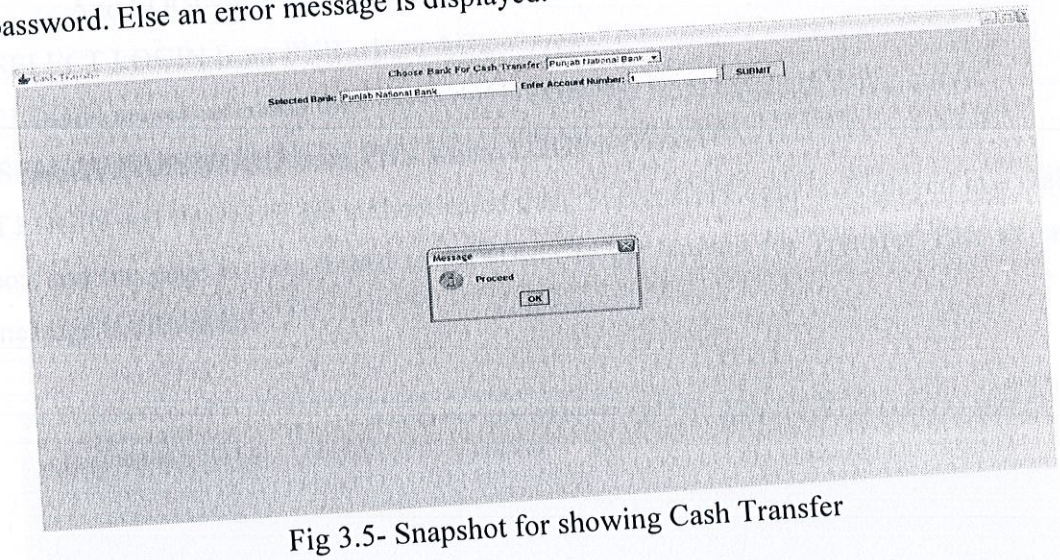


Fig 3.5- Snapshot for showing Cash Transfer

(vi) TransferLOGIN

This class takes the input as the LOGIN and PASSWORD which is PIN for the account holder to whom the amount is to be transferred. User is validated.

(a) Name of Database Accessed:

PNB, SBI

(b) Attributes of PNB, SBI:

ACC, LOGIN, Password, CARDNUM, NAME, Balance

(c) Input:

LOGIN, Password

TransferLOGIN takes the LOGIN from the user stores it in res and checks it to the existing LOGIN for that particular ACC No., PNB and SBI.It gets Account no from BANKDET using query:

"SELECT ACC from BANKDET"

This ACC value is stored in acc

It gets LOGIN name corresponding to this account no using query:

"SELECT LOGIN from PNB where ACC="+acc+" "

And LOGIN name corresponding to LOGIN entered by user using query:

"SELECT LOGIN from PNB where ACC="+res+" "

If LOGIN names are same then password is extracted from database using query:

"SELECT PASSWORD from PNB where LOGIN='"+res+"'"

If LOGIN and Password are authenticated then, Successful Login is displayed in a dialog box and the page is then linked to ATMAmt to get amount for Transfer. Else an error message is displayed.

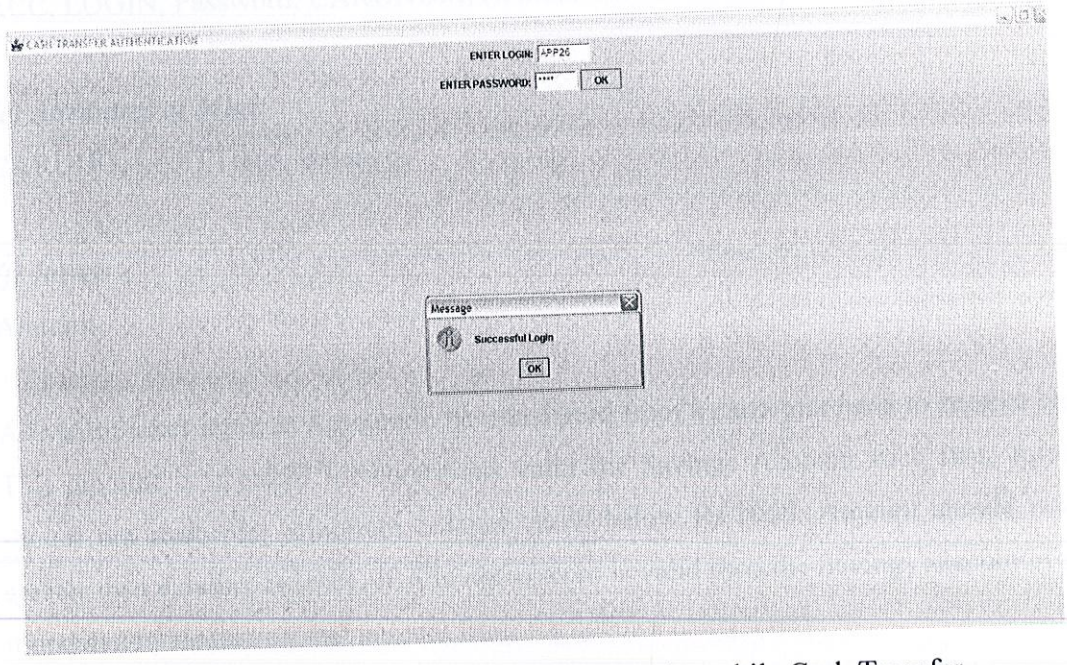


Fig 3.6- Snapshot for showing authentication while Cash Transfer

(vii) ATMAmt

This class takes the input as the Amount to be transferred from Bank's Account holder to another account in another Bank.

Details such as Name, Balance and Account number for Account holder are displayed along with Account number, Name, Balance and Bank Details for the account holder of another bank.

(a) Name of Database Accessed:

BankData , TEMP, PNB, SBI, Mini

(b) Attributes of BankData:

CARDNO(primary key), PIN, ACCOUNT NO, NAME, BALANCE IN CURRENT ACCOUNT, BALANCE IN SAVINGS ACCOUNT

(c) Attributes of TEMP:

CARDNO

(d) Attributes of PNB, SBI:

ACC, LOGIN, Password, CARDNUM, NAME, Balance

(e) Attributes of Mini:

CARDNO, LASTTrans, Balance

(f) Input:

Amount

ATMAmt takes input as Amount to be transferred from a particular bank to another bank. This amount is checked for conditions valid for Savings Account such that, Amount should not make the minimum balance reach below Rs.1000. Amount should not be greater than existing balance. If Amount entered is valid then the Savings account of user if updated by deducting that amount from the balance using query:

"UPDATE PNB SET BALANCE="+ (num4-num1) +" where

ACC="+Integer.parseInt(res5)+""

Where existing balance is in num4 and amount entered is in num1.

And the table for mini statement is populated.

INSERT INTO MINI (CARDNO, LASTTRANS, BALANCE) values ('"+num3+"', '

"+num1+"','"+ (num2-num1) +" ')

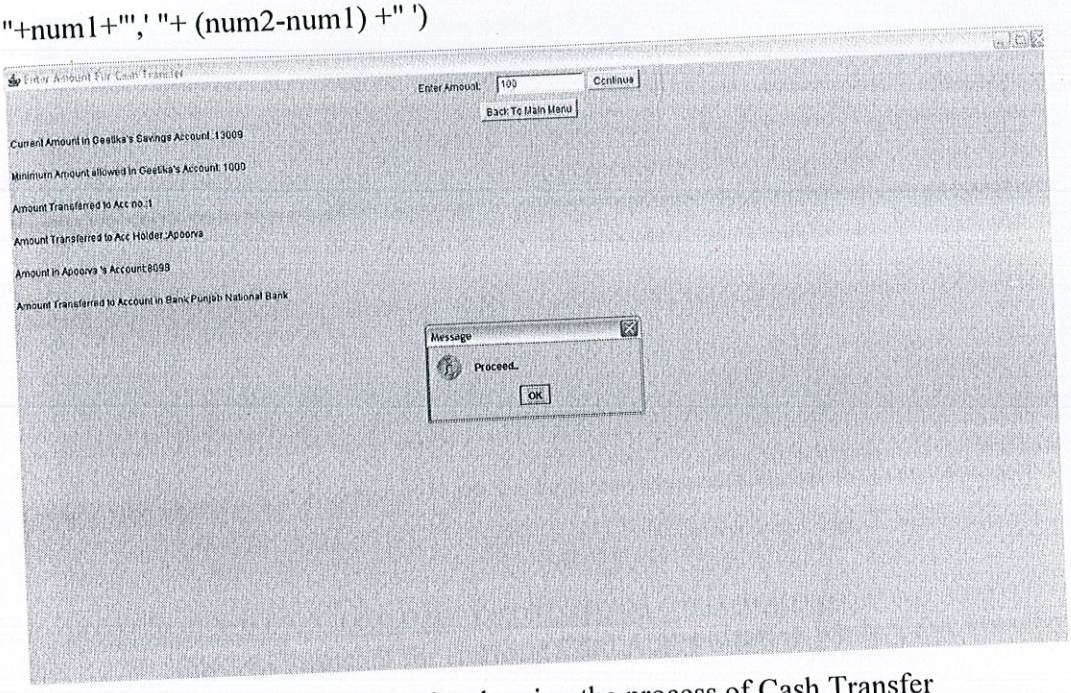


Fig 3.7- Snapshot for showing the process of Cash Transfer

(viii) CardAuthenticate

This class takes the input as any three digits of the CARD Number of the user to whom the data is to be transferred to. In this case the 4th, 7th and 9th digits of the CARD Number are taken by the user. The CARD number is selected using the Account number of the user using query:

"SELECT CARDNUM from PNB where ACC="+Integer.parseInt(res)+"

Where account number is stored in res.

CARDNNUM is stored in num.

The 4th, 7th and 9th digits are extracted as

4th = (num%100)/10;

7th = (num%10000)/1000;

9th = (num%10000000)/1000000;

The Digits are entered as text boxes and authenticated using n1, n2, n3. If the Digits are authentic, the system goes on to display the Mini Statement. Else an error message is displayed.

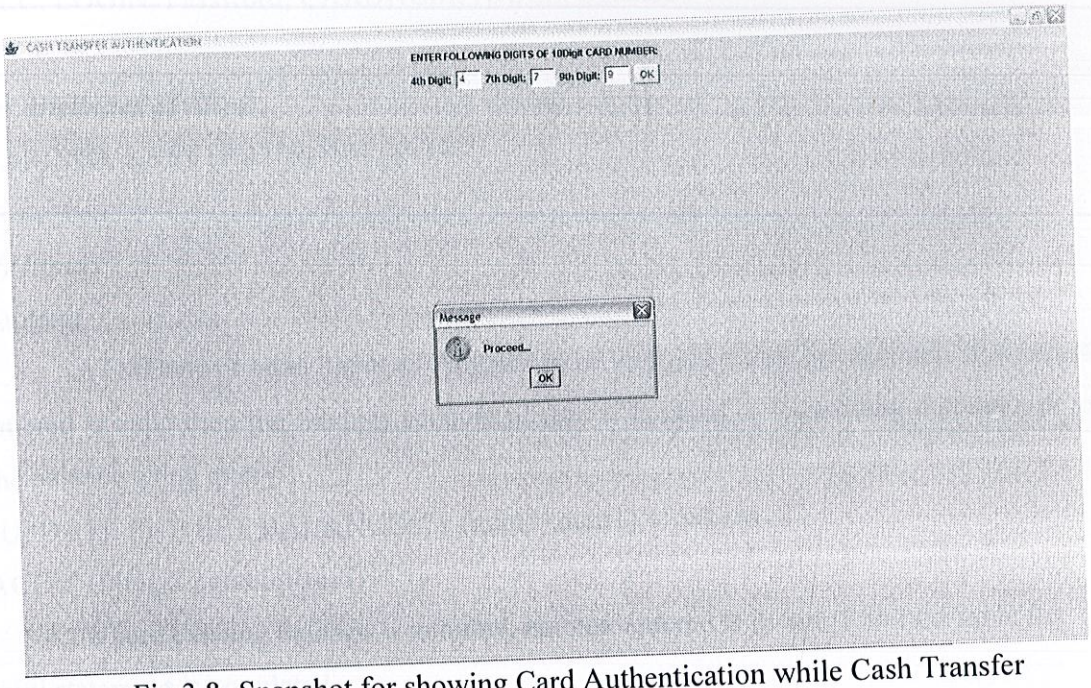


Fig 3.8- Snapshot for showing Card Authentication while Cash Transfer

(ix) ATMDeposit

This class takes the input as the Amount to be deposited into Bank's savings account. Details such as Name, Balance and Account number for Account holder are displayed along minimum balance allowed in that account.

(a) Name of Database Accessed:

BankData, TEMP, PNB, SBI, Mini

(b) Attributes of BankData:

CARDNO(primary key), PIN, ACCOUNT NO, NAME, BALANCE IN CURRENT ACCOUNT, BALANCE IN SAVINGS ACCOUNT

(c) Attributes of TEMP:

CARDNO

(d) Attributes of PNB, SBI:

ACC, LOGIN, Password, CARDNUM, NAME, Balance

(e) Attributes of Mini:

CARDNO, LASTTRANS, BALANCE

(f) Input:

Amount

ATMDeposit takes input as Amount to be deposited into the account. If Amount entered is valid then the Savings account of user if updated by increasing that amount to the balance using query:

"UPDATE PNB SET BALANCE="+ (num4+num1) +" where ACC="+Integer.parseInt(res5)+""

Where existing balance is in num4, amount entered is in num1 and the table for mini statement is populated.

INSERT INTO MINI (CARDNO, LASTTRANS, BALANCE) values ('"+num3+"', ' "+num1+"',' "+ (num2-num1) +" ')

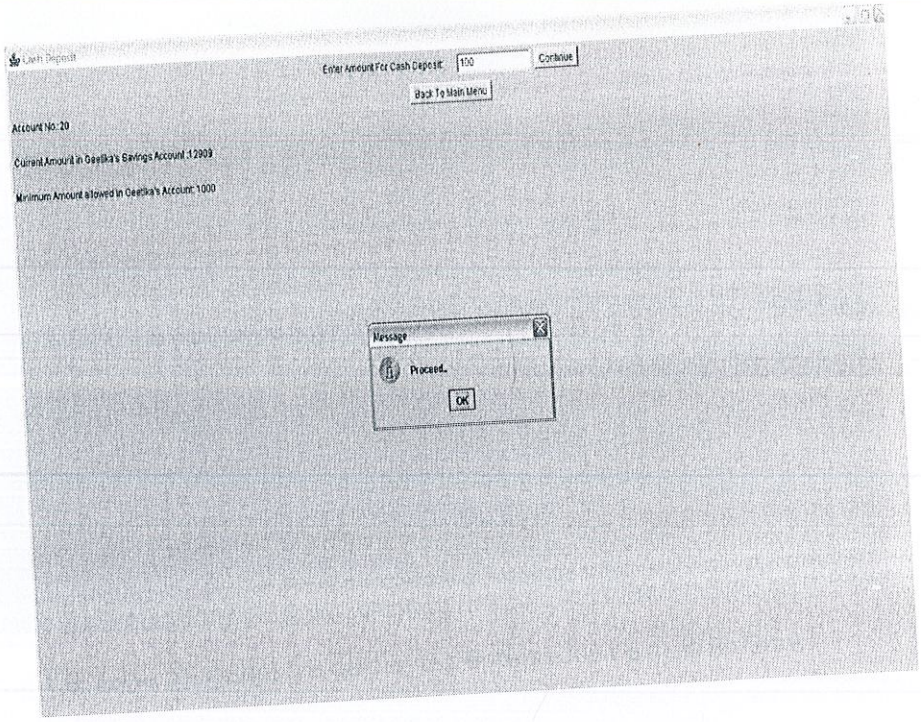


Fig 3.9- Snapshot for showing Cash Deposit

(x) ATMCash

This class gives the user the option of the two types of accounts that the user can access, such that:

-Savings Account

-Current Account

The user clicks on either of the two accounts and can access the accounts and can take out money from the account by entering the amount.

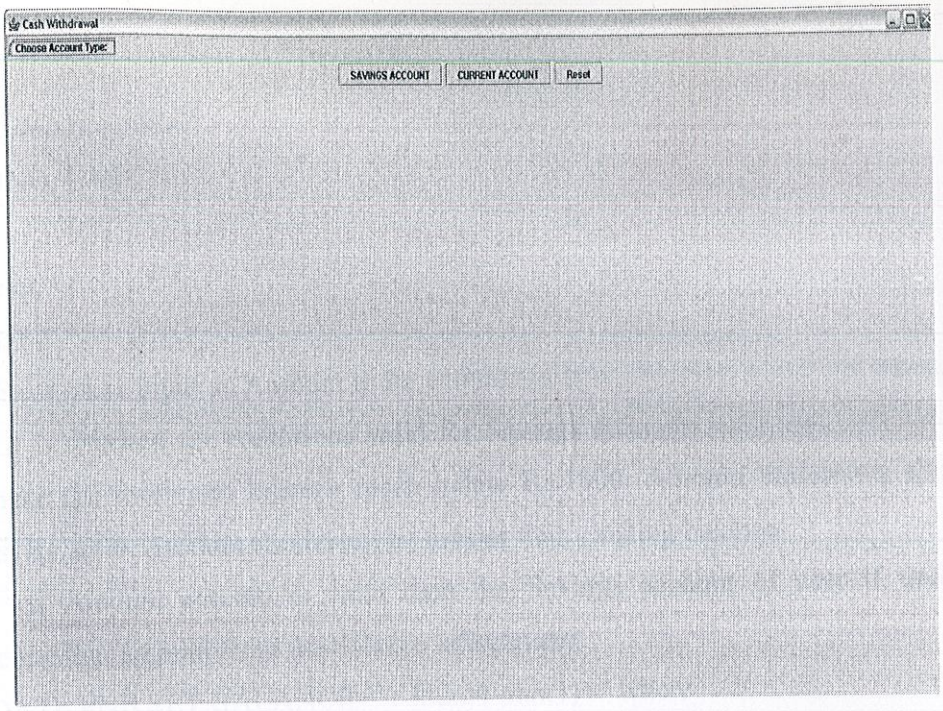


Fig 3.10- Snapshot for displaying options for Cash WithDrawl

(xi) ATMSave

This class takes the input as the Amount to be withdrawn from Savings account. Details such as Name, Balance and Account number for Account holder are displayed along with Account number, Name, Balance and Bank Details for the account holder of another bank.

(a) Name of Database Accessed:

BankData , TEMP, Mini

(b) Attributes of BankData:

CARDNO(primary key), PIN, ACCOUNT NO, NAME, BALANCE IN CURRENT ACCOUNT, BALANCE IN SAVINGS ACCOUNT

(c) Attributes of TEMP:

CARDNO

(d) Attributes of Mini:

CARDNO, LASTTrans, Balance

(e) Input:

Amount

ATMSave takes input as Amount to be withdrawn from that user's savings account. This amount is checked for conditions valid for Savings Account such that, Amount should not make the minimum balance reach below Rs.1000. Amount withdrawn should not exceed Rs.3000. Amount should not be greater than existing balance.

If Amount entered is valid then the Savings account of user if updated by deducting that amount from the balance using query:

"UPDATE PNB SET BALANCE="+ (num4-num1)+" where

ACC="+Integer.parseInt(res5)+""

Where existing balance is in num4 and amount entered is in num1. And the table for mini statement is populated.

INSERT INTO MINI (CARDNO, LASTTRANS, BALANCE) values ('"+num3+"', '

"+num1+"',' "+ (num2-num1) +" '). Or else an error condition is generated.

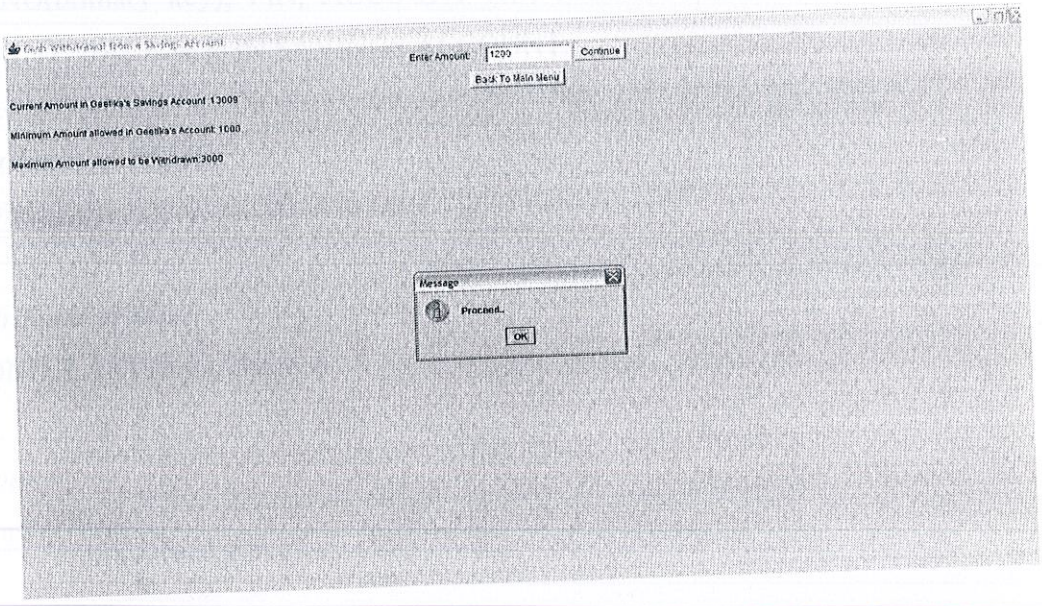


Fig 3.11- Snapshot for showing Cash WithDrawl from a Savings Account

(xii) ATMCurrent

This class has a similar functionality to that of Savings account except it accesses the user's current account and the conditions for cash withdrawal are such as:

Amount should not make the minimum balance reach below Rs.0. Amount withdrawn should not exceed Rs.2000. Amount should not be greater than existing balance.

(xiii) ATMFast

This class takes the input as the Amount to be withdrawn from Savings account. Details such as Name, Balance and Account number for Account holder are displayed along with Account number, Name, Balance and Bank Details for the account holder of another bank.

(a) Name of Database Accessed:

BankData , TEMP, Mini

(b) Attributes of BankData:

CARDNO(primary key), PIN, ACCOUNT NO, NAME, BALANCE IN CURRENT ACCOUNT, BALANCE IN SAVINGS ACCOUNT

(c) Attributes of TEMP:

CARDNO

(d) Attributes of Mini:

CARDNO, LASTTrans, Balance

(e) Input:

Amount

ATMFast takes input as Amount to be withdrawn from that user's savings account. This amount is strictly taken in brackets of Rs.1000, Rs.2000, Rs.3000, and Rs.4000. This

amount is checked for conditions such that, Amount should not make the minimum balance reach below Rs.100 and amount should not be greater than existing balance. If Amount entered is valid then the Savings account of user if updated by deducting that amount from the balance using query:

"UPDATE PNB SET BALANCE="+ (num4-num1)+" where

ACC="+Integer.parseInt(res5)+""

Where existing balance is in num4 and amount entered is in num1. And the table for mini statement is populated.

INSERT INTO MINI (CARDNO, LASTTRANS, BALANCE) values ('"+num3+"', '

"+num1+"',' "+ (num2-num1) +" '). Else it generates an error.

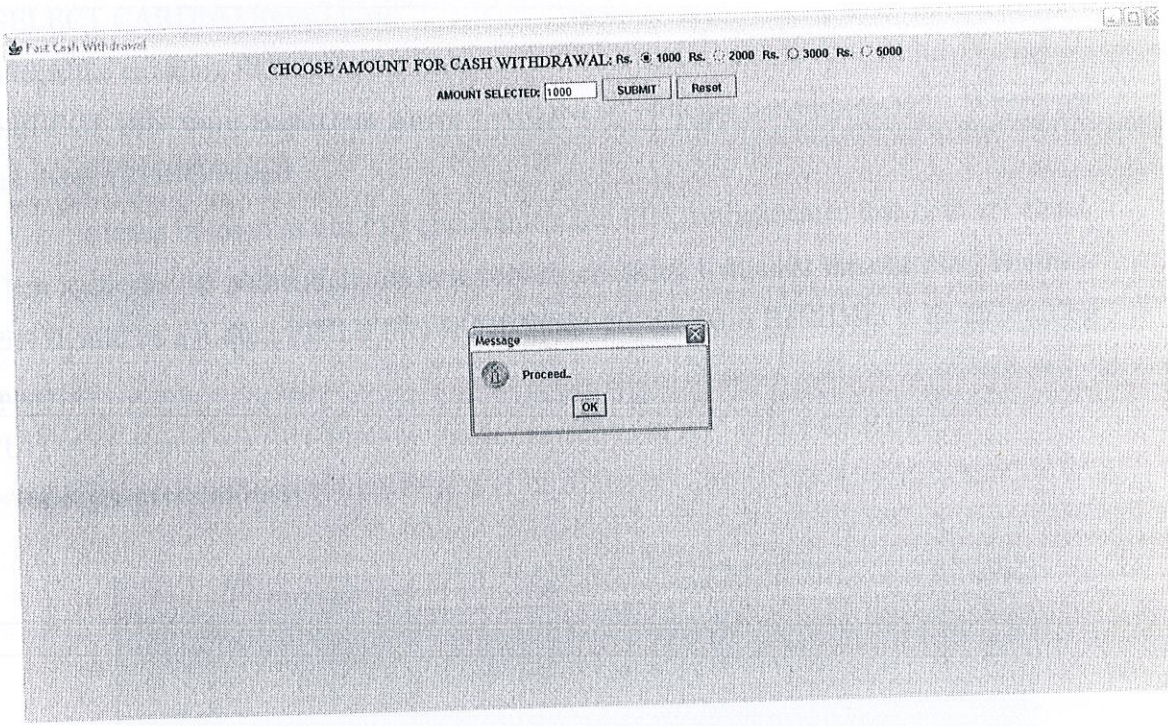


Fig 3.12- Snapshot for Fast Cash WithDrawl

(xiv) PINChange

This class takes the input as the old PIN and new PIN authenticates them and changes the PIN for user accessing the Demo at that time.

(a) Name of Database Accessed:

BankData , TEMP

(b) Attributes of BankData:

CARDNO(primary key), PIN, ACCOUNT NO, NAME, BALANCE IN CURRENT ACCOUNT, BALANCE IN SAVINGS ACCOUNT.

(c) Attributes of TEMP:

CARDNO

It selects the CARDNO for user accessing the System at that time using query:

"SELECT CARDNO from TEMP"

Selects the exisiting PIN using the query:

"SELECT PIN from BankData where CARDNO="+Integer.parseInt(res)+" . Where res has the CARDNO stored.

It takes in input as old PIN and reenters old PIN, authenticates that both are equal. Then it checks the authenticity of new PIN by checking it against the old PIN. The new PIN should be a 4 digit PIN. If PIN is valid then the database BankData is updated using query:

"UPDATE BankData SET PIN="+ Integer.parseInt (str3)+" where CARDNO ="+Integer.parseInt(res)+"

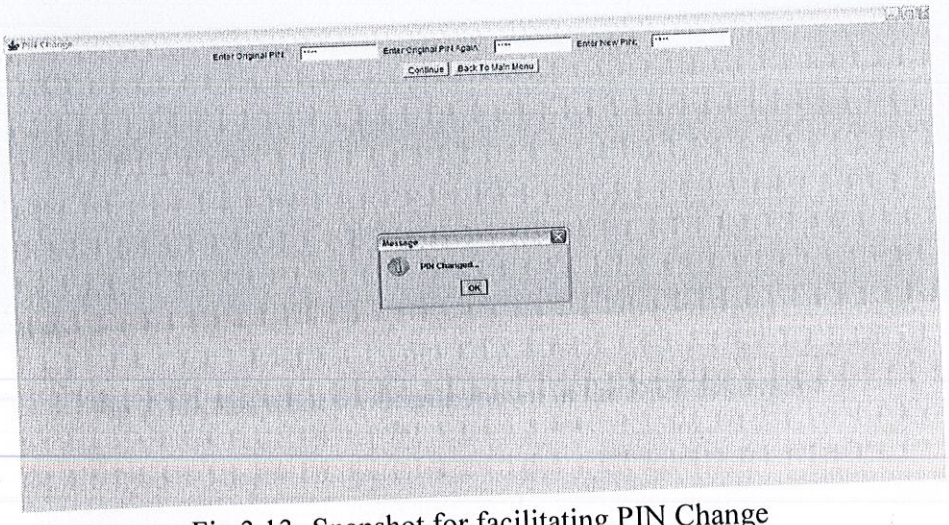


Fig 3.13- Snapshot for facilitating PIN Change

(xv) ATMMini

This class displays the details of the last transaction after the transaction is over.

(a) Name of Database Accessed:

BankData ,TEMP,Mini

(b) Attributes of BankData:

CARDNO(primary key), PIN, ACCOUNT NO, NAME, BALANCE IN CURRENT ACCOUNT, BALANCE IN SAVINGS ACCOUNT

(c) Attributes of TEMP:

CARDNO

(d) Attributes of Mini:

CARDNO, LASTTrans, Balance

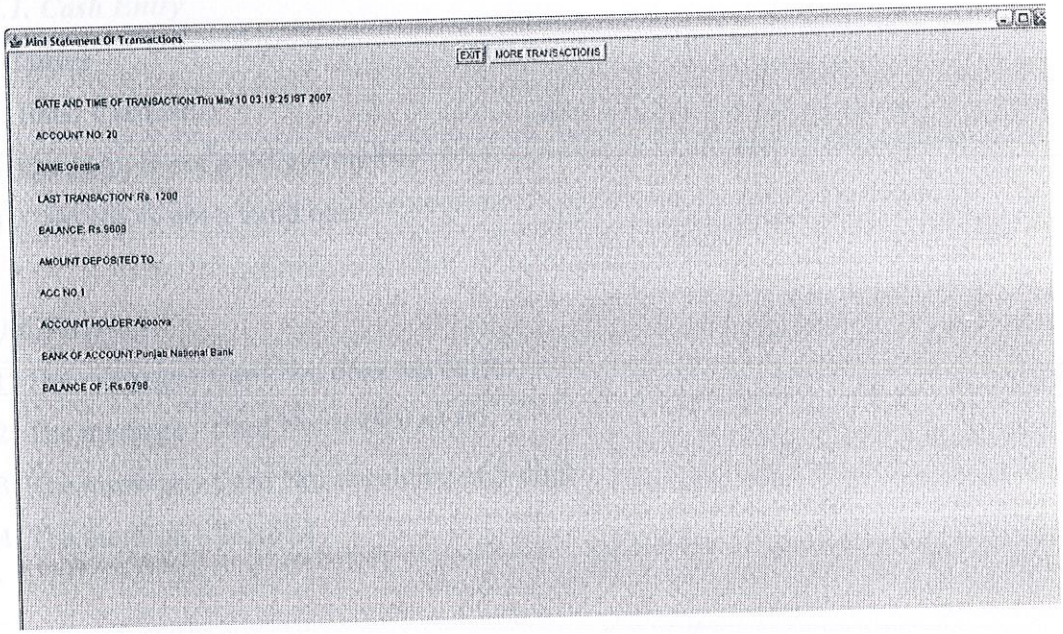


Fig 3.14- Snapshot for details shown in the Mini Statement

3.2. Test Cases

Black Box Testing is testing without knowledge of the internal workings of the item being tested. When black box testing is applied to software engineering, the tester would only know the "legal" inputs and what the expected outputs should be, but not how the program actually arrives at those outputs.

3.2.1. Advantages of Black Box Testing

-Tester needs no knowledge of implementation.

- Tester and programmer are independent of each other.
- Tests are done from a user's point of view.
- Will help to expose any ambiguities or inconsistencies in the specifications.
- Test cases can be designed as soon as the specifications are complete.

3.3. ATM Software Testing

3.3.1. Cash Entry

(i) Causes

C1: Enter Card No.

C2: Card No. is not a 5 digit number

C3: Card No. is not a valid one

(ii) Effects

E1: The message - Card No. does not exist.

E2: The message - Card No. is INVALID.

E3: The message - Card No. should be of 5-digit

E4: The message - Proceed

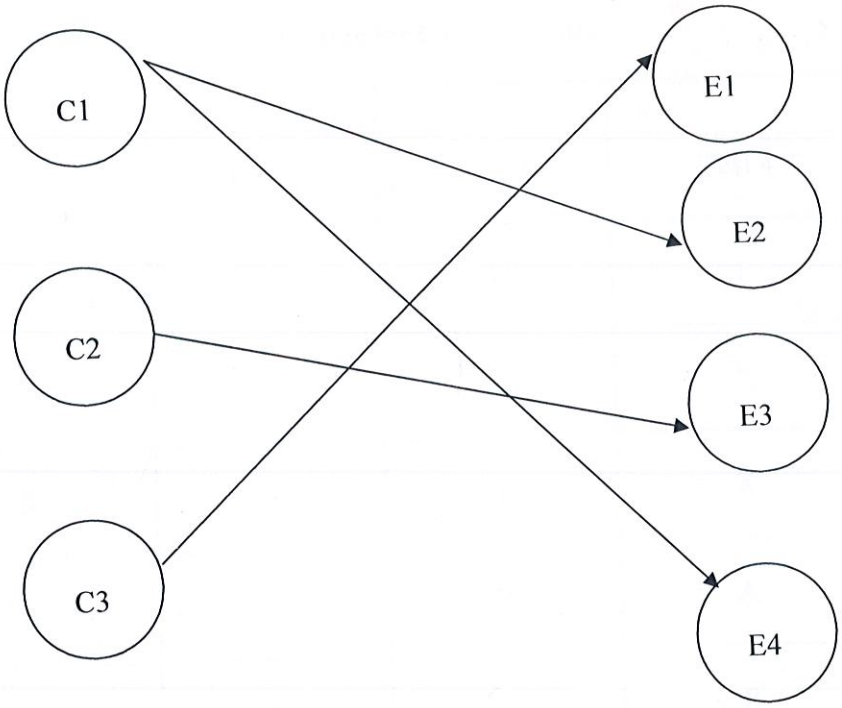


Fig 3.15 - Cause Effect Graph for Card Number entry.

The following are the test cases for the above Cause Effect Graph. The symbols followed in the design of the test cases are as under.

I: input present

S: input absent

X: don't care

P: output present

A: output absent

Table 3.1- Test Cases for Card No. entry

	Test 1	Test 2	Test 3	Test 4
C1	I	I	I	I
C2	S	I	S	X
C3	S	S	I	X
E1	A	A	P	A
E2	A	A	A	P
E3	A	P	A	A
E4	P	A	A	A

3.3.2. PIN Authentication

(i) Causes

C1: Enter PIN

C2: PIN is not a 4 digit number

C3: PIN is not a valid one

(ii) Effects

E1: The message – PIN Invalid

E2: The message – PIN should be of 4-digits

E3: The message – PIN Invalid, Reset System

E4: The message - Proceed

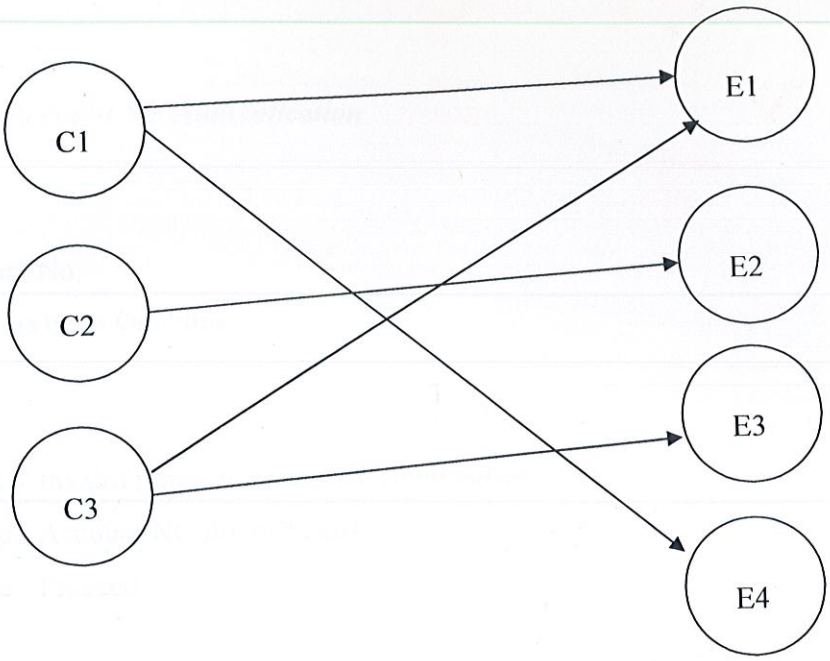


Fig 3.16 - Cause Effect Graph for PIN entry.

Table 3.2- Test Cases for PIN entry

	Test 1	Test 2	Test 3	Test 4
C1	I	I	I	I
C2	S	I	S	X
C3	S	S	I	X
E1	A	A	P	A
E2	A	P	A	A
E3	A	A	P	P

3.3.3. Bank and Account No. Authentication

(i) Causes

C1: Select Bank

C2: Enter Account No.

C3: Account No. is not a valid one

(ii) Effects

E1: The message - Invalid Bank, Account No. combination

E2: The message - Account No. doesn't exist

E3: The message - Proceed

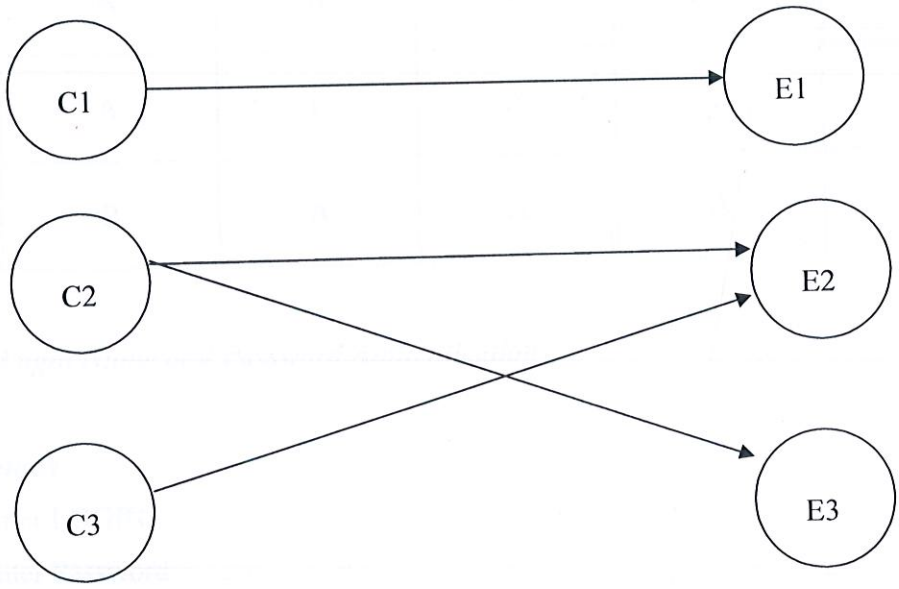


Fig 3.17- Cause Effect Graph for Bank and Account No. Authentication.

Table 3.3- Test Cases for Bank and Account No. Authentication

	Test 1	Test 2	Test 3	Test 4	Test 5
C1	I	I	S	I	I
C2	I	S	I	I	X
C3	S	S	S	I	X
E1	A	A	P	A	P
E2	A	P	A	P	A
E3	P	A	A	A	A

3.3.4. Login Name and Password Authentication

(i) Causes

C1: Enter LOGIN

C2: Enter Password

C3: LOGIN is not a valid one

C4: Password is not a valid one

(ii) Effects

E1: The message- LOGIN is invalid

E2: The message- Password is incorrect

E3: The message - Proceed

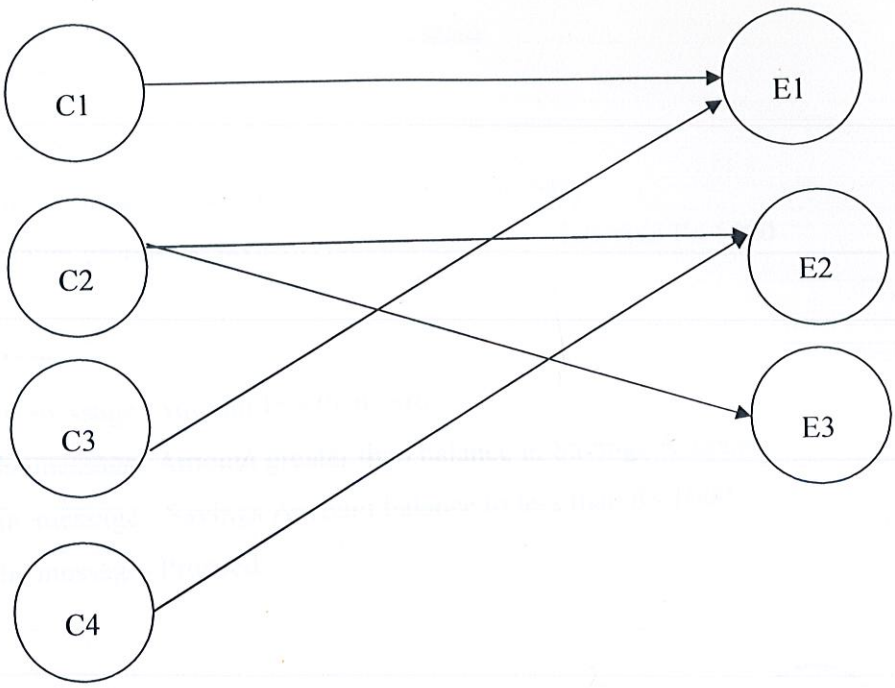


Fig 3.18-Cause Effect Graph for Bank and Account No. Authentication.

Table 3.4- Test Cases for Bank and Account No. Authentication

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
C1	I	I	I	I	I	X
C2	I	I	I	I	X	I
C3	S	I	S	I	X	X
C4	S	S	I	I	X	X
E1	A	P	A	A	A	P
E2	A	A	P	P	P	A
E3	P	A	A	A	A	A

3.3.5. Cash Transfer Amount Authentication

(i) Causes

C1: Enter Amount

C2: Amount greater than balance in Savings Account

C3: Amount decreases Savings Account balance to less than Rs.1000

(ii) Effects

E1: The message: Amount less than zero

E2: The message- Amount greater than balance in Savings Account

E3: The message - Savings Account balance to less than Rs.1000

E4: The message- Proceed

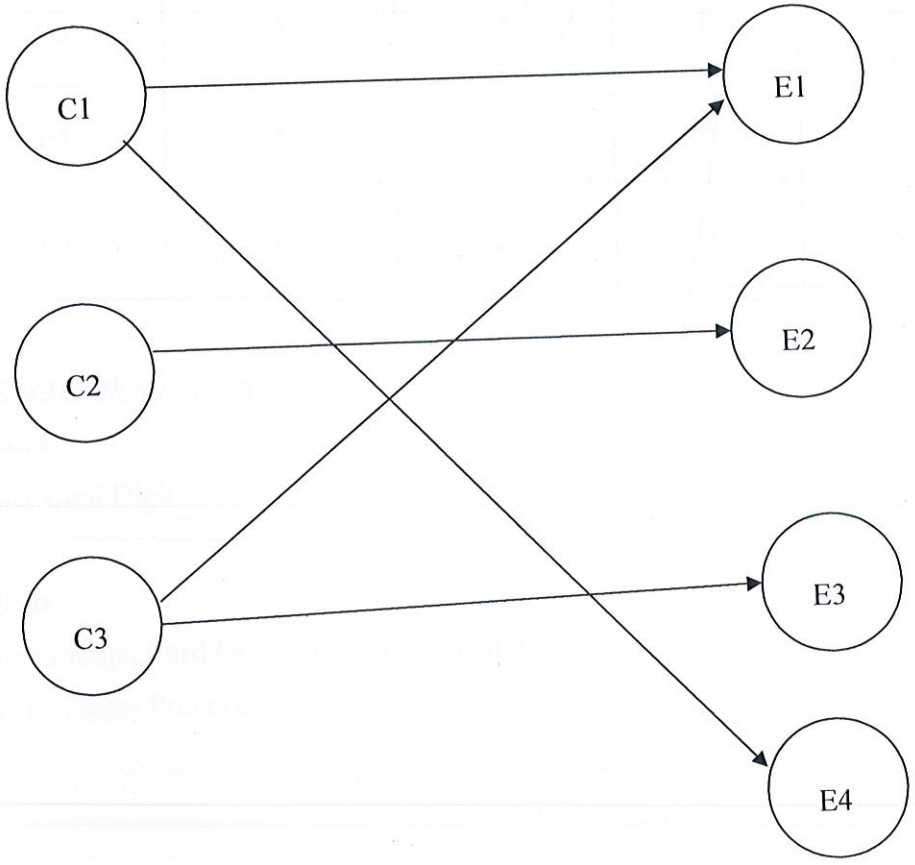


Fig 3.19- Cause Effect Graph for Cash Transfer Amount Authentication

Table 3.5- Test Cases for Cash Transfer Amount Authentication

	Test 1	Test 2	Test 3	Test 4
C1	I	I	I	I
C2	S	X	I	S
C3	S	X	S	I
E1	A	P	A	A
E2	A	A	P	A
E3	A	A	A	P
E4	P	A	A	A

3.3.6. Card Authentication

(i) Causes

C1: Enter Card Digits

(ii) Effects

E1: The message: Card Digits entered are invalid

E2: The message- Proceed

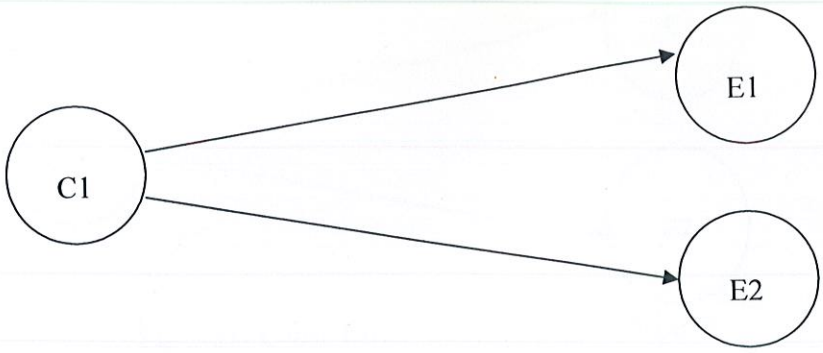


Fig 3.20- Cause Effect Graph for Card Authentication

Table 3.6- Test Cases for Cash Transfer Amount Authentication

	Test 1	Test 2
C1	I	I
E1	A	P
E2	P	A

3.3.6. Card Deposit

(i) Causes

C1: Enter Amount

(ii) Effects

E1: The message: Amount less than zero

E2: The message- Proceed

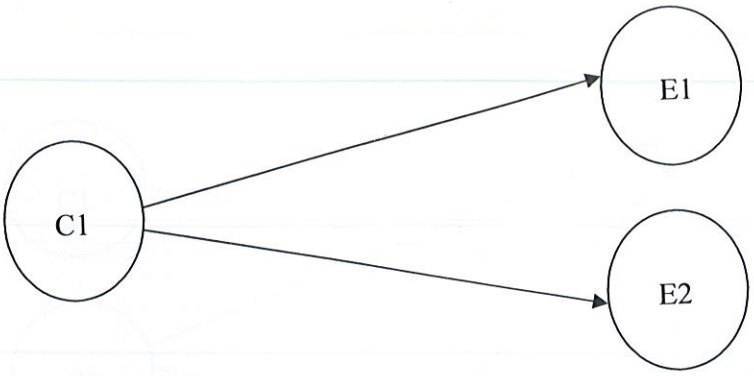


Fig 3.21- Cause Effect Graph for Card Deposit

Table 3.7- Test Cases for Cash Deposit

	Test 1	Test 2
C1	I	I
E1	A	P
E2	P	A

3.3.7. PIN Change

(i) Causes

C1: Enter Old PIN

C2: Reenter Old PIN

C3: Enter New PIN

C4: Confirm New PIN

(ii) Effects

E1: Message- Proceed

E2: Message- Old PIN is invalid

E3: Message- Reentered Old PIN does not match with the previous one

E4: Message- Reentered New PIN does not match with the previous one

E5: Message- New PIN is invalid

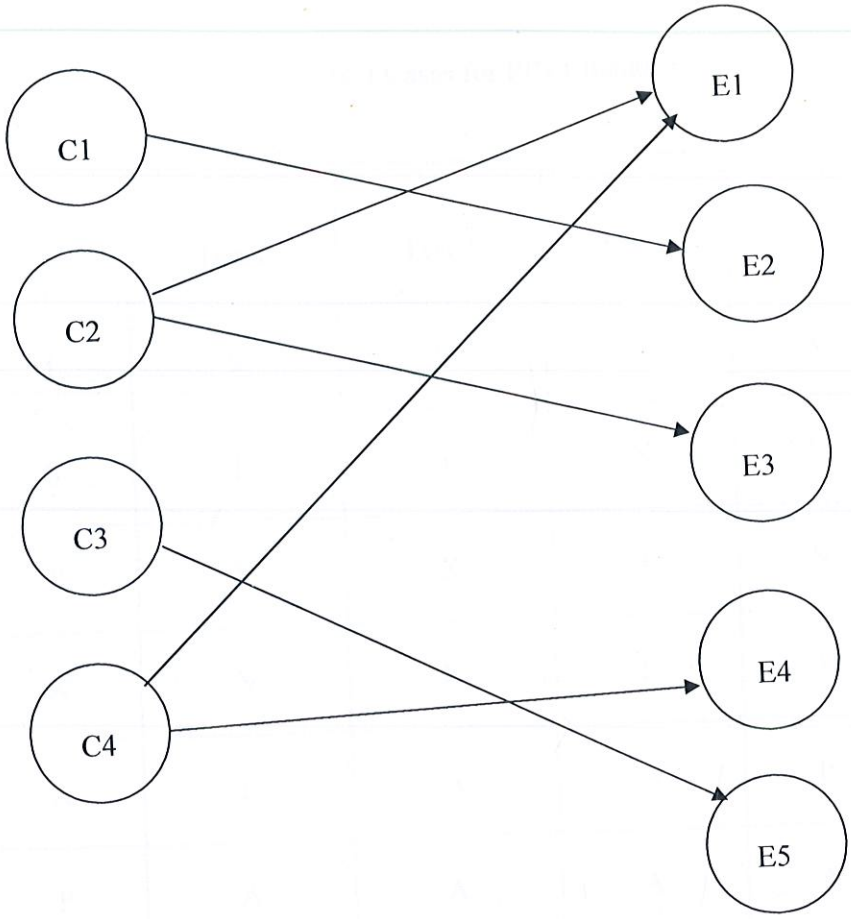


Fig 3.22- Cause Effect Graph for PIN Change

Table 3.8- Test Cases for PIN Change

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
C1	I	S	S	X	X	X
C2	S	I	I	X	X	X
C3	X	X	X	I	S	S
C4	X	X	X	S	I	I
E1	A	P	A	A	P	A
E2	P	A	A	A	A	A
E3	A	A	P	A	A	A
E4	A	A	A	A	A	P
E5	A	A	A	P	A	A

CHAPTER 4
CONCLUSION AND FUTURE SCOPE

4.1. Conclusion

Formal notions and tools have been efficiently used to design the Graphical User Interface of an Intelligent Automated Teller Machine. The complementary view of the system has been taken into account. The software not only showcases the good system behavior but also helps the user learn all the faults in the system behavior so that no matter how complex the Human-Computer Interface gets the user is aware of the system behavior.

Black Box Testing has been efficiently used to test the software developed as the software is concerned to educate the user about the good and faulty system behavior. Essentially, Black Box testing deals with just the input and output which is what the user is interested in. Furthermore, the design process is accomplished using the notion of a Finite State Automata which at a basic level deals with input to and output from a particular state and this is exactly what the black box testing tests.

4.2. Future Scope

Our major contribution has been to model the desired and undesired events for software. However, the approach can be improvised by using the notion of Regression testing which helps provide a better solution and divides the task of testing in two stages thus reducing the number of test cases available at one go. Moreover, minimization of FSA helps reduce the complexity of large interfaces, thereby, decreasing the cost of testing considerably enabling the cumulating costs of testing to be in compliance with test budget.

Bibliography

References:

1. F. Belli, "Finite-State Testing and Analysis of Graphical User Interfaces", Proc. of 12th ISSRE, IEEE Computer Society Press, 2001, pp. 34-43.
2. F. Belli, B. Güldali, "A Holistic Approach to Test-Driven Model Checking", Proc. of the 18th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2005), ACM Press, 2005 (to appear).
3. F. Belli, Ch. J. Budnik, N. Nissanke, "Finite-State Modeling, Analysis and Testing of System Vulnerabilities", ARCS Workshops 2004, pp. 19-33.

Books:

1. Deitel and Deitel, "JAVA How to Program", Pearson Education (Fifth Edition).
2. Herbert Schildt, "The Complete Reference to Java 2", Tata McGrawHill (Fifth Edition).
3. Peter Linz, "Formal Languages and Automata", Narosa (Third Edition).

Websites:

1. www.wikipedia.org.
2. www.google.com.
3. www.w3schools.com.

APPENDIX A

A.1 HTML

```
<html>
<head>
 <title>
 Automated Teller Machine
 </title>
 </head>
 <body background="3.gif">
 <img src="2.gif" align="left" height="200" width="200">
 <H2> An automated teller machine or automatic teller machine (ATM) is a computerized telecommunications device that provides a financial institution's customers a method of financial transactions in a public space without the need for a human clerk or bank teller. Security is provided by the customer entering a personal identification number (PIN).
 </H2>
 <h3>
 <marquee direction="up" width=100% height=50%>
 <pre>
 Bibiliography:
 Dietel and Dietel
 The Complete Reference to Java
 Google:Search Engine
 Project Guide:
 Mr Nitin
 Made by:
 Shrey Ahuja (031230)
 Geetika Upadhyay (031273)
 Neha Gulati (031229)
 </h3>
 </marquee> </body> </html>
```

A.2. Storage and Comparison of coordinate values using XML

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import java.io.*;
import java.util.*;
class ParseXML3
{
int[] cor = new int[2];
int[] cor2 = new int[2];
I3 i = new I3("deposit2.jpg");
public ParseXML3()
{
try
{
String value =DumpXML.Value("MouseClick1.xml");
System.out.println("Value = "+ value);
StringTokenizer st = new StringTokenizer(value,",");
String[] coord = new String[2];
for(int i=0; i<2; i++)
{
if(st.hasMoreTokens())
{
String s = st.nextToken();
coord[i] =s;
cor[i] = Integer.parseInt(coord[i]);
}
System.out.println(cor[i]+ "");
}
String value1 =DumpXML.Value("MouseClick2.xml");
```

```
System.out.println("Value = "+ value1);
StringTokenizer st1 = new StringTokenizer(value1,",");
String[] coord1 = new String[2];
int[] cor1 = new int[2];
for(int i=0; i<2; i++)
{
if(st1.hasMoreTokens())
{
String s = st1.nextToken();
coord1[i] =s;
cor2[i] = Integer.parseInt(coord1[i]);
}
System.out.println(cor2[i]+ "");
}
String first = Check(cor[0],cor[1]);
String sec = Check(cor2[0],cor2[1]);
if(first.equals("c") && sec.equals("c2"))
{
System.out.println("combination possible");
ImagePanel4.X1 =462;
ImagePanel4.Y1 = 112;
ImagePanel4.X2 = 468;
ImagePanel4.Y2 = 261;
i.setSize(1200,1200);
i.setVisible(true);
}
if(first.equals("c2") && sec.equals("c3"))
{
System.out.println("combination  possible");
i.setVisible (false);
ImagePanel4.X1 = 468;
```

```
ImagePanel4.Y1 = 261;
ImagePanel4.X2 = 722;
ImagePanel4.Y2 = 406;
i.setSize(1200,1200);
i.setVisible(true);
}
if(first.equals("c3") && sec.equals("c4"))
{
System.out.println("combination  possible");
i.setVisible(false);
ImagePanel4.X1 = 722;
ImagePanel4.Y1 = 406;
ImagePanel4.X2 = 294;
ImagePanel4.Y2 = 397;
i.setSize(1200,1200);
i.setVisible(true);
}
if(first.equals("c4") && sec.equals("c5"))
{
System.out.println("combination  possible");
i.setVisible(false);
ImagePanel4.X1 = 294;
ImagePanel4.Y1 = 397;
ImagePanel4.X2 = 501;
ImagePanel4.Y2 = 551;
i.setSize(1200,1200);
i.setVisible(true);
}
if(first.equals("c4") && sec.equals("c2"))
{
System.out.println("combination  possible");
```

```
i.setVisible(false);
ImagePanel4.X1 = 294;
ImagePanel4.Y1 = 397;
ImagePanel4.X2 = 468;
ImagePanel4.Y2 = 261;
i.setSize(1200,1200);
i.setVisible(true);
}
if(!((first.equals("c") && sec.equals("c2"))|| (first.equals("c2") && sec.equals("c3")) ||
(first.equals("c3") && sec.equals("c4")) || (first.equals("c4") && sec.equals("c5")) ||
(first.equals("c4") && sec.equals("c2"))))
{
System.out.println("combination not  possible");
JOptionPane.showMessageDialog(null, "combination  not possible" );
}}
catch(Exception e) {}
}
public String Check(int x, int y)
{
String st ="c3";
Point center = new Point(462,112);
Circle c = new Circle(center,50.0);
Point center2 = new Point(468,261);
Circle c2 = new Circle(center2,50.0);
Point center3 = new Point(722,406);
Circle c3 = new Circle(center3,50.0);
Point center4 = new Point(294,397);
Circle c4 = new Circle(center4,50.0);
Point center5 = new Point(501,551);
Circle c5 = new Circle(center5,50.0);
Point px = new Point(x,y);
```

```
if (c2.contains(px))
st = "c2";
if (c.contains(px))
st = "c";
if (c3.contains(px))
st = "c3";
if (c4.contains(px))
st = "c4";
if (c5.contains(px))
st = "c5";
return st;
}
public static void main(String args[])
{
ParseXML3 px = new ParseXML3();
}
}
```

APPENDIX B

B.1 Project Layout

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.String;
import java.awt.* ;
class MyImage extends JFrame
{
private JPanel imagePanel,buttonPanel;
private JLabel imageLabel;
private JButton button;
private Container contentPane;
public MyImage()
{
super("My Image");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
contentPane=getContentPane();
contentPane.setLayout(new BorderLayout());
buildImagePanel();
buildButtonPanel();
contentPane.add(imagePanel,BorderLayout.NORTH);
contentPane.add(buttonPanel,BorderLayout.SOUTH);
pack();
setVisible(true);
}
private void buildImagePanel()
{
imagePanel=new JPanel();
imageLabel=new JLabel("CLick Button To Check Image");
imagePanel.add(imageLabel);
```

```
}
private void buildButtonPanel()
{
ImageIcon Img;
buttonPanel=new JPanel();
Img=new ImageIcon("TIPS.gif");
button =new JButton("Get Image");
button.setIcon(Img);
button.addActionListener(new ButtonListener());
buttonPanel.add(button);
button= new JButton("EXIT");
button.addActionListener(new ButtonListener1());
buttonPanel.add(button);
}
private class ButtonListener implements ActionListener
{
public void actionPerformed(ActionEvent e)
{
ImageIcon mainImage=new ImageIcon("ATM.jpg");
imageLabel.setIcon(mainImage);
imageLabel.setText(null);
pack();
}}
private class ButtonListener1 implements ActionListener
{
public void actionPerformed(ActionEvent e)
{
System.exit(0);
}}
public static void main(String args[])
{
```

```
MyImage image=new MyImage();
}}
public class MenuTest extends JFrame {
private final Color colorValues[] = { Color.black, Color.blue, Color.red, Color.green };
private JRadioButtonMenuItem colorItems[], fonts[];
private JCheckBoxMenuItem styleItems[];
private JLabel displayLabel;
private ButtonGroup fontGroup, colorGroup;
private int style;
public MenuTest()
{
super( "Using JMenus" );
JMenu fileMenu = new JMenu( "File" );
fileMenu.setMnemonic( 'F' );
JMenuItem userMenu = new JMenu( "User IPs & FIPs" );
userMenu.setMnemonic( 'U' );
fileMenu.add( userMenu );
userMenu.addActionListener(
new ActionListener() {
public void actionPerformed( ActionEvent event )
{
JOptionPane.showMessageDialog( MenuTest.this,
“This option helps the user to find out the IPs and FIPs for every transaction",
"User IPs & FIPs", JOptionPane.PLAIN_MESSAGE );
}}
);
JMenuItem cash1SubItem=new JMenuItem("CashWithdrawl");
cash1SubItem.setMnemonic('d');
userMenu.add(cash1SubItem);
cash1SubItem.addActionListener(
new ActionListener(){
```

```
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click on Ok to transact on states of cash withdrawl",
"CashWithdrawl", JOptionPane.PLAIN_MESSAGE );
setVisible(false);
c.add(new ImageFrame(), "Center");
pack();
ImageFrame ifr  = new ImageFrame("cash2.jpg");
setSize(1000,1000);
}}
);
JMenuItem pin1SubItem= new JMenuItem("PinChange");
pin1SubItem.setMnemonic('g');
userMenu.add(pin1SubItem);
pin1SubItem.addActionListener(
new ActionListener(){
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click Ok to transact on states of pin change",
"PinChange", JOptionPane.PLAIN_MESSAGE );
ImageFrame1 ifr  = new ImageFrame1("PIN2.jpg");
setSize(1000,1000);
}}
);
JMenuItem fast1SubItem= new JMenuItem("FastCash");
fast1SubItem.setMnemonic('h');
userMenu.add(fast1SubItem);
fast1SubItem.addActionListener(
new ActionListener(){
```

```
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click Ok to transact on states of fast cash",
"FastCash", JOptionPane.PLAIN_MESSAGE );
ImageFrame2 ifr  = new ImageFrame2("Fast2.jpg");
setSize(1000,1000);
}}
);
JMenuItem deposit1SubItem= new JMenuItem("Deposit");
deposit1SubItem.setMnemonic('t');
userMenu.add(deposit1SubItem);
deposit1SubItem.addActionListener(
new ActionListener(){
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click Ok to transact on states of deposit money",
"Deposit", JOptionPane.PLAIN_MESSAGE );
ImageFrame3 ifr = new ImageFrame3("deposit2.jpg");
setSize(1000,1000);
}}
);
JMenuItem transfer1SubItem= new JMenuItem("Transfer");
transfer1SubItem.setMnemonic('r');
userMenu.add(transfer1SubItem); transfer1SubItem.addActionListener(
new ActionListener(){
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click Ok to transact on states of transfer money",
```

```
"Transfer", JOptionPane.PLAIN_MESSAGE );
ImageFrame4 ifr = new ImageFrame4("Transfer2.jpg");
setSize(1000,1000);
}}
);
fileMenu.add( userMenu );
fileMenu.addSeparator();
JMenuItem exitItem = new JMenuItem( "Exit" );
exitItem.setMnemonic( 'x' );
fileMenu.add( exitItem );
exitItem.addActionListener(
new ActionListener() {  // anonymous inner class
public void actionPerformed( ActionEvent event )
{
System.exit( 0 );
}}
); // end call to addActionListener
JMenu helpMenu = new JMenu( "Help" );
helpMenu.setMnemonic( 'H' );
JMenuItem contentMenu = new JMenu( "Contents..." );
contentMenu.setMnemonic( 'C' );
helpMenu.add( contentMenu );
contentMenu.addActionListener(
new ActionListener() {
public void actionPerformed( ActionEvent event )
{
JOptionPane.showMessageDialog( MenuTest.this,
"This Software helps in Making a Finite Automata and Shows The Faulty Interaction
Pairs", "Contents", JOptionPane.PLAIN_MESSAGE );
}}
);
```

```
JMenuItem cashSubItem=new JMenuItem("CashWithdrawl");
cashSubItem.setMnemonic('W');
contentMenu.add(cashSubItem);
cashSubItem.addActionListener(
new ActionListener()
{
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click on Ok to see details about cash withdrawl",
"CashWithdrawl", JOptionPane.PLAIN_MESSAGE );
Container c= getContentPane();
setVisible(false);
c.add(new Cash(), "Center");
pack();
setSize(1000,1000);
}}
);
JMenuItem pinSubItem= new JMenuItem("PinChange");
pinSubItem.setMnemonic('h');
contentMenu.add(pinSubItem);
pinSubItem.addActionListener(
new ActionListener(){
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click Ok to check details for changing pin",
"PinChange", JOptionPane.PLAIN_MESSAGE );
Container c=getContentPane();
setVisible(false);
c.add(new Pin(), "Center");
```

```
pack();
setSize(1000,1000);
}}
);
JMenuItem fastSubItem= new JMenuItem("FastCash");
fastSubItem.setMnemonic('F');
contentMenu.add(fastSubItem);
fastSubItem.addActionListener(
new ActionListener(){
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click Ok to check details for fast cash",
"FastCash", JOptionPane.PLAIN_MESSAGE );
Container c= getContentPane();
setVisible(false);
c.add(new Fast(), "Center");
pack();
setSize(1000,1000);
}}
);
JMenuItem depositSubItem= new JMenuItem("Deposit");
depositSubItem.setMnemonic('e');
contentMenu.add(depositSubItem);
depositSubItem.addActionListener(
new ActionListener(){
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click Ok to check details for depositing money",
"Deposit", JOptionPane.PLAIN_MESSAGE );
```

```
Container c=getContentPane();
setVisible(false);
c.add(new Deposit(), "Center");
pack();
setSize(1000,1000);
}}
);
JMenuItem transferSubItem= new JMenuItem("Transfer");
transferSubItem.setMnemonic('T');
contentMenu.add(transferSubItem);
transferSubItem.addActionListener(
new ActionListener(){
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(MenuTest.this,
"Click Ok to check details for transferring money",
"Transfer", JOptionPane.PLAIN_MESSAGE );
Container c=getContentPane();
setVisible(false);
c.add(new Transfer(), "Center");
pack();
setSize(1000,1000);
}}
);
helpMenu.add( contentMenu );
helpMenu.addSeparator();
JMenuItem aboutItem = new JMenuItem( "About..." );
aboutItem.setMnemonic( 'A' );
helpMenu.add( aboutItem );
aboutItem.addActionListener(
new ActionListener() {  // anonymous inner class
```

```
public void actionPerformed( ActionEvent event )
{
Sample s = new Sample();
s.getContentPane().add(new Browser("file:///C:/java/bin/neha.html"),"Center");
}}
); // end call to addActionListener
JMenuItem imageItem=new JMenuItem("Image");
imageItem.setMnemonic('I');
helpMenu.add(imageItem);
imageItem.addActionListener(
new ActionListener() {
public void actionPerformed(ActionEvent event)
{
MyImage i=new MyImage();
}}
);
JMenuBar bar = new JMenuBar();
setJMenuBar( bar );
bar.add( fileMenu );
JMenu viewMenu = new JMenu( "View" );
viewMenu.setMnemonic( 'v' );
JMenuItem demoMenu = new JMenu( "Demos..." );
demoMenu.setMnemonic( 'D' );
helpMenu.add( demoMenu );
setVisible(false);
demoMenu.addActionListener(
new ActionListener() {
public void actionPerformed( ActionEvent event )
{
JOptionPane.showMessageDialog( MenuTest.this,
```

```
"This Software helps in Making a Finite Automata and Shows The Faulty Interaction
Pairs","Contents", JOptionPane.PLAIN_MESSAGE );
}}
);
JMenuItem mini1SubItem=new JMenuItem("Mini Statement");
mini1SubItem.setMnemonic('M');
demoMenu.add(mini1SubItem);
mini1SubItem.addActionListener(
new ActionListener()
{
public void actionPerformed(ActionEvent event)
{
Container c= getContentPane();
setVisible(false);
c.add(new MainFrame3(), "Center");
pack();
setSize(1000,1000);
setVisible(true);
}}
);
viewMenu.add( demoMenu );
viewMenu.addSeparator();
String colors[] = { "Black", "Blue", "Red", "Green" };
JMenu colorMenu = new JMenu( "Color" );
colorItems = new JRadioButtonMenuItem[ colors.length ];
colorGroup = new ButtonGroup();
ItemHandler itemHandler = new ItemHandler();
for ( int count = 0; count < colors.length; count++ ) {
colorItems[ count ] =
new JRadioButtonMenuItem( colors[ count ] );
colorMenu.add( colorItems[ count ] );
```

```
colorGroup.add( colorItems[ count ] );
colorItems[ count ].addActionListener( itemHandler );
}
colorItems[ 0 ].setSelected( true );
viewMenu.add( colorMenu );
viewMenu.addSeparator();
String fontNames[] = { "Serif", "Monospaced", "SansSerif" };
JMenu fontMenu = new JMenu( "Font" );
fontMenu.setMnemonic( 'n' );
fonts = new JRadioButtonMenuItem[ fontNames.length ];
fontGroup = new ButtonGroup();
for ( int count = 0; count < fonts.length; count++ ) {
fonts[ count ] = new JRadioButtonMenuItem( fontNames[ count ] );
fontMenu.add( fonts[ count ] );
fontGroup.add( fonts[ count ] );
fonts[ count ].addActionListener( itemHandler );
}
fonts[ 0 ].setSelected( true );
fontMenu.addSeparator();
String styleNames[] = { "Bold", "Italic" };
styleItems = new JCheckBoxMenuItem[ styleNames.length ];
StyleHandler styleHandler = new StyleHandler();
for ( int count = 0; count < styleNames.length; count++ ) {
styleItems[ count ] =
new JCheckBoxMenuItem( styleNames[ count ] );
fontMenu.add( styleItems[ count ] );
styleItems[ count ].addItemListener( styleHandler );
 }
viewMenu.add( fontMenu );
bar.add(viewMenu);
bar.add(helpMenu);
```

```
displayLabel = new JLabel("Automated Teller Machine", SwingConstants.CENTER);
displayLabel.setForeground( colorValues[ 0 ] );
displayLabel.setFont( new Font( "Serif", Font.PLAIN, 72 ) );
getContentPane().setBackground( Color.PINK);
getContentPane().add( displayLabel, BorderLayout.CENTER );
setSize( 1000, 800 );
setVisible( true );
} // end constructor
public static void main( String args[] )
{
MenuTest application = new MenuTest();
application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}
private class ItemHandler implements ActionListener {
public void actionPerformed( ActionEvent event )
{
for ( int count = 0; count < colorItems.length; count++ )
if ( colorItems[ count ].isSelected() ) {
displayLabel.setForeground( colorValues[ count ] );
break;
}
for ( int count = 0; count < fonts.length; count++ )
if ( event.getSource() == fonts[ count ] ) {
displayLabel.setFont(
new Font( fonts[ count ].getText(), style, 72 ) );
break;
}
repaint ();
}} // end class ItemHandler
private class StyleHandler implements ItemListener {
public void itemStateChanged( ItemEvent e ) {
```

```
style = 0;
if ( styleItems[ 0 ].isSelected() )
style += Font.BOLD;
if ( styleItems[ 1 ].isSelected() )
style += Font.ITALIC;
displayLabel.setFont(
new Font( displayLabel.getFont().getName(), style, 72 ) );
repaint();
}}
} // end class MenuTest
```

APPENDIX C

C.1 ATM Menu

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.lang.*;
import javax.swing.*;
import javax.swing.JOptionPane;
import java.io.*;
import java.lang.String;
import java.sql.*;
import java.util.*;
public class ATMMenu extends JFrame implements ActionListener {
JPanel horizontal2 ;
JPanel horizontal3 ;
JPanel horizontal4 ;
JButton blist2[];
Connection con;
Statement stat1;
Statement stat2;
ResultSet result;
ResultSet result1;
public ATMMenu()
{
setTitle("MENU");
horizontal2 = new JPanel();
horizontal3 = new JPanel();
horizontal4 = new JPanel();
final int SIZE = 6; // number of buttons on each Box
blist2=new JButton[10];
blist2[1]= new JButton("CASH WITHDRAWAL");
```

```
blist2[3]=new JButton("FAST CASH");
blist2[4]=new JButton("CASH DEPOSIT");
blist2[5]=new JButton("CASH TRANSFER");
blist2[6]= new JButton("PIN CHANGE");
blist2[7]=  new JButton("BALANCE ENQUIRY");
blist2[0]=  new JButton("RESET SYSTEM");
blist2[8]=  new JButton("RESET SYSTEM");
blist2[9]=  new JButton("RESET SYSTEM");
horizontal2.add( blist2[1]);
horizontal2.add(blist2[3]);
horizontal2.add(blist2[0]);
horizontal3.add(blist2[4] );
horizontal3.add(blist2[5] );
horizontal3.add(blist2[8]);
horizontal4.add(blist2[6] );
horizontal4.add(blist2[7] );
horizontal4.add(blist2[9]);
JTabbedPane tabs = new JTabbedPane(
JTabbedPane.TOP, JTabbedPane.SCROLL_TAB_LAYOUT );
tabs.addTab( "Cash Transaction Functions",horizontal3 );
tabs.addTab( "Cash Withdrawal Functions",horizontal2);
tabs.addTab( "Other Functions",horizontal4 );
getContentPane().add( tabs );  // place tabbed pane on content pane
setSize( 1400, 800 );
setVisible( true );
blist2[0].addActionListener(this);
blist2[1].addActionListener(this);
blist2[3].addActionListener(this);
blist2[4].addActionListener(this);
blist2[5].addActionListener(this);
blist2[6].addActionListener(this);
```

```
blist2[7].addActionListener(this);
blist2[8].addActionListener(this);
blist2[9].addActionListener(this);
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection("jdbc:odbc:db1");
stat1= con.createStatement();
result=stat1.executeQuery("SELECT * from MINI");
if(result.next())
{
stat2 = con.createStatement();
stat2.executeUpdate( "DELETE FROM MINI");
}
stat2= con.createStatement();
result1=stat2.executeQuery("SELECT * from BANKDET");
if(result1.next())
{
stat1 = con.createStatement();
stat1.executeUpdate( "DELETE FROM BANKDET");
}}
catch(ClassNotFoundException cnfex)
{
System.err.println("Failed to load jdbc odbc driver.");
cnfex.printStackTrace();
System.exit(1);
}
catch (SQLException sqlex)
{
JOptionPane.showMessageDialog(null,"sql exception","database
error",JOptionPane.ERROR_MESSAGE);
```

```
System.err.println("Unable to load.");
sqlex.printStackTrace();
System.exit(1);
} }
public void actionPerformed(ActionEvent ae)
{
String str = ae.getActionCommand();
if(str.equals("exit"))
{
System.exit(0);
}
else if(str.equals("RESET SYSTEM"))
{
JOptionPane.showMessageDialog(null,"RESETING SYSTEM....");
JFrameBg b=new JFrameBg();
setVisible(false);
}
else if(str.equals("CASH WITHDRAWAL"))
{
ATMCash a=new ATMCash();
setVisible(false);
a.setVisible(true);
}
else if(str.equals("CASH DEPOSIT"))
{
ATMDeposit a=new ATMDeposit();
setVisible(false);
a.setSize(1400,800);
a.setVisible(true);
}
else if(str.equals("FAST CASH"))
```

```
{
setVisible(false);
ATMFast a=new ATMFast();
a.setVisible(true);
a.setSize(1400,800);
}
else if(str.equals("PIN CHANGE"))
{
ATMPIN a=new ATMPIN();
setVisible(false);
a.setVisible(true);
a.setSize(1400,800);
}
else if(str.equals("CASH TRANSFER"))
{
ATMTransfer1 a=new ATMTransfer1();
setVisible(false);
a.setVisible(true);
a.setSize(1400,800);
}
else if(str.equals("BALANCE ENQUIRY"))
{
ATMMini6 a=new ATMMini6();
setVisible(false);
a.setVisible(true);
a.setSize(1400,800);
}}
public static void main( String args[] )
{
ATMMenu application = new ATMMenu();
application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

```
application.setSize(1400,800);
}}
```

C.2 ATM Demo1

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.lang.*;
import javax.swing.*;
import javax.swing.JOptionPane;
import java.io.*;
import java.lang.String;
import java.sql.*;
import java.util.*;
public class ATMDemo1 extends JFrame implements ActionListener
{
String msg = "";
String res=" ";
String res1=" ";
JPanel buttonPanel;
JPanel textPanel;
JPanel labelPanel;
JPanel labelPanel1;
TextField text;
JLabel label2,label4,label3,label1,label5;
JButton blist2[];
Connection con;
Statement stat;
Statement stat2;
ResultSet result1;
Statement stat1;
int j=0;
```

```
public ATMDemo1()
{
setTitle("CARD VERIFICATION");
Container container=getContentPane();
buttonPanel=new JPanel();
blist2=new JButton[3];
blist2[0] = new JButton("exit");
blist2[1]=new JButton("continue");
blist2[2]=new JButton("reset");
label4=new JLabel("Press:");
label3=new JLabel(" 'reset' to start demo again & 'exit' to exit the appliction");
buttonPanel.add( label4);
buttonPanel.add( label3 );
buttonPanel.add( blist2[0] );
buttonPanel.add(blist2[2]);
labelPanel=new JPanel();
label2=new JLabel("ENTER CARD NUMBER TO START BANKING: ");
label2.setFont(new Font("Serif",Font.PLAIN,20));
labelPanel.add(label2);
label1=new JLabel(" Press continue to proceed:");
text=new TextField(7);
textPanel=new JPanel();
labelPanel.add(text);
textPanel.add(label1);
textPanel.add(blist2[1]);
container.add(labelPanel,BorderLayout.NORTH);
container.add(textPanel,BorderLayout.CENTER);
container.add(buttonPanel,BorderLayout.SOUTH);
for(j=0;j<3;j++)
blist2[j].addActionListener(this);
text.addActionListener(this);
```

```
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection("jdbc:odbc:db1");
stat2= con.createStatement();
result1=stat2.executeQuery("SELECT * from TEMP");
if(result1.next())
{
stat = con.createStatement();
stat.executeUpdate( "DELETE CARDNO FROM TEMP");
}}
catch(ClassNotFoundException cnfex) {
System.err.println("Failed to load jdbc odbc driver.");
cnfex.printStackTrace();
System.exit(1);
}
catch (SQLException sqlex) {
JOptionPane.showMessageDialog(null,"sql exception","database
error",JOptionPane.ERROR_MESSAGE);
System.err.println("Unable to load.");
sqlex.printStackTrace();
System.exit(1);
}}
public void actionPerformed(ActionEvent ae)
{
String str = ae.getActionCommand();
if(str.equals("exit"))
System.exit(0);
else if(str.equals("reset")){
JOptionPane.showMessageDialog(null,"RESETING SYSTEM....");
JFrameBg b=new JFrameBg();
setVisible(false); }
```

```
else if(str.equals("continue")){
msg=text.getText();
if(msg.length()==0){
JOptionPane.showMessageDialog(null,"Insert Card Properly!!");
JFrameBg b=new JFrameBg();
setVisible(false);
}
if(msg.length()>0)
{
res=text.getText();
try{
stat = con.createStatement();
result1= stat.executeQuery( "SELECT CARDNO from BankData where
CARDNO="+Integer.parseInt(res)+"");
result1.next();
res1 = result1.getString(1);
int num=Integer.parseInt(res1);
System.out.println(num);
stat2 = con.createStatement();
stat2.executeUpdate("INSERT  INTO TEMP(CARDNO) values('"+num+"') ");
if(res1!="") {
JOptionPane.showMessageDialog(null,"Proceed..");
ATMDemo2 a=new ATMDemo2();
a.setVisible(true);
a.setSize(1400,760);
 setVisible(false);
}}
catch (SQLException sqlex)  {
text.setText("");
msg="";
```

```
JOptionPane.showMessageDialog(null,"Card Invalid!!Try Again
Later","ERROR",JOptionPane.ERROR_MESSAGE);
System.err.println("Unable to load.");
JFrameBg b=new JFrameBg();
setVisible(false);
}}} }
public static void main(String args[])
{
ATMDemo1 app = new ATMDemo1();
app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE );
app.setVisible(true);
app.setSize(1400,800);
}}
```

C.3 ATM PIN

```
import java.awt.*;
import java.awt.event.*;
import java.lang.*;
import javax.swing.*;
import javax.swing.JOptionPane;
import java.io.*;
import java.lang.String;
import java.sql.*;
import java.util.*;
public class ATMPIN extends JFrame implements ActionListener
{
JPanel buttonPanel;
JPanel resetPanel;
Label amount;
Label pin1;
Label pin2;
```

```
Label pin3;
Label pin7;
TextField pin;
TextField pin4;
TextField pin5;
TextField pin6;
Button button1;
Button button2;
String msg=" ";
String str1=" ";
String str2=" ";
String str3=" ";
String str4=" ";
String res=" ";
Connection con;
Statement stat1;
Statement stat2;
Statement stat;
ResultSet result1;
ResultSet result;
public ATMPIN()
{
setTitle("PIN Change");
Container container=getContentPane();
buttonPanel=new JPanel();
resetPanel=new JPanel();
Label pin1 = new Label("Enter Original PIN: ");
pin = new TextField(4);
buttonPanel.add(pin1);
buttonPanel.add(pin);
Label pin2 = new Label("Enter Original PIN Again: ");
```

```
pin4 = new TextField(4);
buttonPanel.add(pin2);
buttonPanel.add(pin4);
Label pin3 = new Label("Enter New PIN: ");
pin5= new TextField(4);
buttonPanel.add(pin3);
buttonPanel.add(pin5);
Label pin7 = new Label("Enter New PIN: ");
pin6= new TextField(4);
buttonPanel.add(pin7);
buttonPanel.add(pin6);
button1=new Button("Continue");
button2=new Button("Reset");
resetPanel.add(button1);
resetPanel.add(button2);
container.add(buttonPanel,BorderLayout.NORTH);
container.add(resetPanel,BorderLayout.CENTER);
pin.setEchoChar('*');
pin4.setEchoChar('*');
pin5.setEchoChar('*');
pin6.setEchoChar('*');
button1.addActionListener(this);
button2.addActionListener(this);
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection("jdbc:odbc:db1");
stat = con.createStatement();
result= stat.executeQuery( "SELECT CARDNO from TEMP");
result.next();
res = result.getString(1);
stat1 = con.createStatement();
```

```
result1= stat1.executeQuery( "SELECT PIN from BankData where
CARDNO="+Integer.parseInt(res)+"");
result1.next();
msg= result1.getString(1);
}
catch(ClassNotFoundException cnfex) {
System.err.println("Failed to load jdbc odbc driver.");
cnfex.printStackTrace();
System.exit(1);
}
catch (SQLException sqlex) {
JOptionPane.showMessageDialog(null,"sql exception","database
error",JOptionPane.ERROR_MESSAGE);
System.err.println("Unable to load.");
sqlex.printStackTrace();
System.exit(1);
}}
public void actionPerformed(ActionEvent ae)
{
String str = ae.getActionCommand();
if(str.equals("Continue"))
{
str1=pin.getText();
str2=pin4.getText();
str3=pin5.getText();
str4=pin6.getText();
if(Integer.parseInt(str1)!=Integer.parseInt(msg))
{
JOptionPane.showMessageDialog(null,"PIN Entered
Incorrect","Error",JOptionPane.ERROR_MESSAGE);
pin.setText("");
```

```
pin4.setText("");
pin5.setText("");
pin6.setText("");}
else if(Integer.parseInt(str1)!=Integer.parseInt(str2))
{
JOptionPane.showMessageDialog(null,"PIN Entered Don't
Match","Error",JOptionPane.ERROR_MESSAGE);
pin.setText("");
pin4.setText("");
pin5.setText("");
pin6.setText("");
}
else if(Integer.parseInt(str3)!=Integer.parseInt(str4))
{
JOptionPane.showMessageDialog(null,"New PINs Entered Don't
Match","Error",JOptionPane.ERROR_MESSAGE);
pin.setText("");
pin4.setText("");
pin5.setText("");
pin6.setText("");
}
else if(str3.length()!=4)
{
JOptionPane.showMessageDialog(null,"Please enter a 4digit
PIN","Error",JOptionPane.ERROR_MESSAGE);
pin.setText("");
pin4.setText("");
pin5.setText("");
pin6.setText("");
}
else if(str1.length()>0 && str2.length()>0 && str3.length()==4)
```

```
{
try{
stat=con.createStatement();
stat.executeUpdate( "UPDATE BankData SET PIN="+Integer.parseInt(str3)+" where
CARDNO="+Integer.parseInt(res)+"");
}
catch (SQLException sqlex) {
JOptionPane.showMessageDialog(null,"sql exception","database
error",JOptionPane.ERROR_MESSAGE);
System.err.println("Unable to load.");
sqlex.printStackTrace();
System.exit(1);
}
JOptionPane.showMessageDialog(null,"PIN Changed...");
setVisible(false);
JFrameBg a=new JFrameBg();
}}
if(str.equals("Reset"))
{
JOptionPane.showMessageDialog(null,"Resetting System...");
setVisible(false);
JFrameBg a= new JFrameBg();
a.setVisible(true);
a.setSize(1400,800);
}}
public static void main(String args[])
{
ATMPIN app = new ATMPIN();
app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE );
app.setVisible(true);
app.setSize(1400,800);}}
```