2001

Using Java powered iButton to replace student ID card

Yu-Hui Cho

Using Java Powered iButton to Replace Student ID Card

By
Yu-Hui Cho

May 25, 2001

This project is for the degree of
Master of Science
in Computer Science

Approved by:

Committee Chairman
Hans-Peter Bischof, Computer Science Department, R.I.T.

Committee Reader
Paul T. Tymann, Computer Science Department, R.I.T.

Committee Observer
James R. Vallino, Computer Science Department, R.I.T.

Department of Computer Science
College of Applied Science and Technology
Rochester Institute of Technology
Rochester, New York

Permission granted

Title of thesis

Using Java Powered iButton to Replace Student ID card

I, *author's name,* hereby **grant** permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 06/05/01 Signature of Author: _____

Contents

1. Introduction

The Java Powered iButton is a secure Java Virtual Machine encased in a stainless-steel circular container only 16mm wide. It is Java-powered and designed according to Sun's Java Card 2.0 specification. Inside the 16mm steel case is a monolithic chip with a high-speed, 8-bit microprocessor, 32K-bytes of ROM, 6K-bytes of NV SRAM up to 134K-bytes, a True Time clock and a high-speed math accelerator for 1024-bit public key cryptography. And since it relies on battery-backed static RAM to store its applets, it can be programmed over and over again as needs change. The Java iButton also supports full-length 32-bit Java integers, and garbage collection. So we can process and store data on the iButton. See Appendix A. for more detail about Java Powered iButton.

First goal of this project is to design a student Java iButton that can be used to replace conventional student ID card (magnetic stripe card). The student Java iButton will offer more security and keep the same functionalities that student ID card has. These functionalities are roughly described as following:

- Java iButton offers 6K-bytes of NVRAM up to 134K-bytes memory. So we can store the owner's latest information when opening or updating an account (ex, name, address, and etc...).

- Access bank account through the network.
- Use it as an e-wallet (I will use vending machine to demonstrate this).

Second goal of this project is using Remote Method Invocation (RMI) with secure connection to design and implement client/server architecture. I will also use this design to communicate with Java iButton.

In order to make this project work, extra hardware and software are needed:

- **Hardware:** A Blue Dot Receptor and a Parallel Port Adapter are used to make Java iButton communicate with a PC.
- **Software:** The Java iButton Developer's Kit (iB-IDE1.10) from Dallas Semiconductor. This kit contains the JavaCard Platform classes for the Java iButton, a converter for translating the Java byte code into the iButtons machine instructions, and loader for loading the code on to the iButton. It also contains an OpenCard Terminal Driver for the Blue Dot Receptor Reader.

Next, we will see what benefits we will get when using my design.

2. Why Choosing These Topics

This chapter will discuss the reason why I choose these topics as the main skeleton of my project.

2.1 Java Powered iButton

Table 1. gives you a rough idea about the differences between Java Powered iButton and conventional stripe card. And these differences will be discussed later.

Table 1. Java powered iButton and magnetic stripe card comparison

	Java Powered iButton	Magnetic Stripe Card
Data Storage Space	From 6 K-bytes up to 134 K-bytes	About 140 bytes of data
Security	• Careful attention to physical security (rapid zeroization). • Runs Java better (plus portions enhance JavaCard 2.0).	Easy enough today to purchase the tools needed to hack into confidential data.
Portability and Flexibility	With their processing capabilities and increased memory capacity, provide a convenient solution for making data portable and universally accessible.	Require a host system to store and process all data. And card must always be connected to an online system.

- **Data Storage Space**

  Magnetic stripe card: It can hold only about 140 bytes of data. This just enough space for a PIN number and critical data needed to log into a server-based system.

  Java powered iButton: High memory capacity (up to 134K-bytes). Ample memory makes us easily to meet the application's storage needs (ex, name, address, phone number, security number, and etc...).

- **Security**

  Magnetic stripe card: It is easy to bend your card accidentally. And then you lost everything you got on the card. It is also easy to purchase the tools needed to hack into confidential data because there is no extra protection for the card.

  Java powered iButton: In physical security, iButton is armored with stainless steel for the hard knocks of everyday use. If you try to open the iButton, it will generate a tamper response that will cause rapid zeroization[1] of NV SRAM to prevent disclosure of secure data. It is Java Card[2] 2.0-compliant to execute Java applets.

[1] **zeroization:** A tamper response whereby an attempt to access protected memory initiates an instant erasure.

[2] **Java Card:** *JavaCard* was introduced by Schlumberger and submitted as a standard by JavaSoft recently.

Standard cryptographic classes include SHA-1 for secret key digital signatures, RSA[3] public key, DES[4] algorithm. Physically secure computer chip keeps keys safe and executes 1024-bit public key cryptography for the safe exchange of information.

- **Portability and Flexibility**

  Magnetic stripe card: It is not portable because all data must be stored and processed on a host system. And it must always connect to server, then we can get the information we need. It is not programmable, so this limits the card functionalities.

  Java powered iButton: Compared with stripe card, Java iButton is much more flexible. Because Java powered iButton uses high-speed nonvolatile static RAM (NV SRAM), it can be erased and rewritten as often as necessary without wearing out. Java iButton can perform multiple functions in a wide range of industries. It is designed so that its memory can be partitioned into blocks of random size that operate independently from one another. This means multiple service providers can place their services into the Java iButton that one service provider can't access the services of another. About portability, you do not require access to remote databases at the time of the transaction. Java iButton, with their processing capabilities and increased memory capacity, provide a perfect solution for making data portable and universally accessible.

[3]**RSA:** A public-key encryption technology. [4]**DES:** A symmetric-key encryption method.

2.2 Remote Method Invocation (RMI)

Table 2. Compare RMI to Remote Procedure Call (RPC)

	RMI	RPC
Interfaces	Implemented with Java	Implemented with C or Pascal
Object-Oriented	Yes	No
Object Handling	Good	Not Good

- **Interfaces**

  RMI: It is implemented with Java language.

  RPC: It is typically implemented with C or Pascal.

- **Object-Oriented**

  RMI: Based on an object-oriented language, Java. Objects are invoked by calling methods.

  RPC: It is not an object but rather just a set of functions that can be invoked remotely.

- **Object Handling**

  RMI: It uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism. We don't have to bother with breaking down the object into its raw elements when we use RMI.

  RPC: A library of function calls that handles the actual communication between two applications. It does not translate well

into distributed object systems, where communication between program-level objects residing in different address spaces is needed. We still have to handle most of the data type issues.

2.3 Java Database Connectivity (JDBC)

Because my program is written in Java language, it is the first choice to use Java programming language commands rather than SQL. It is more flexible to use because it has many new features [1]. Such as:

- **Batch updates:** Significant improve data update speed.
- **Result set enhancements:** Scrollable result set: Make it possible to move Result set's cursor to the specific row.
- **New data type support:** Binary Large Object (BLOB) and Character Large Object (CLOB) are introduced to manipulate large objects.

2.4 Java 2 Security

There are two major security issues when we use RMI:

- There is no authentication.
- Objects are serialized and transmitted over the network. They are not encrypted, so it is possible for anyone to read the data.

The Java Secure Socket Extension (JSSE) can establish a secure Internet communications between client and server. It includes functionality for data

encryption, server authentication, message integrity, and optional client authentication. This will take care of no authentication problem.

The Java Cryptography Extension (JCE) is a package that provides a framework and implementations for data encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. We can use this extension to en/decrypt objects that will be transmitted over the network. This package will solve the second problem.

Next, we will take an in-depth look at architecture used in my project design.

3. Architecture

I will discuss the architecture used in my project design. And you will get some idea how my program works.

3.1 Communication Between Pc End and Java iButton

3.1.1 JavaCard Framework

Java iButton is designed to be fully compatible with the Java Card 2.0 standard. So its architecture is just like java card architecture (see Figure 1). We have to notice that Java card platform is a scaled-down version of the normal Java platform. But Java Card 2.0 specification used in Java iButton is with additional capability.

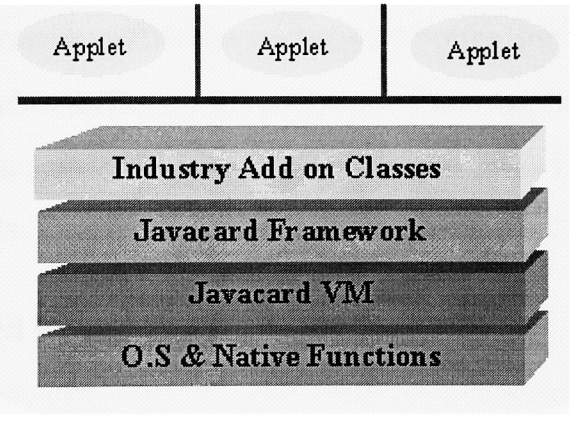


Figure 1. JavaCrad Framework [8]

Enhancements [2] to the Java Card 2.0 specification include:

- True 32-bit Java integers for straightforward computation.
- Automatic garbage collection for efficient reuse of memory space.

- Resizable commit buffer optimizes memory usage and allows for large atomic transactions.
- Add or delete applets in a secure manner to update applications after issuance.
- Large Java stack supports complex computation.
- Java-accessible True Time Clock time stamps transactions.
- Java-accessible unique 64-bit registration number supplements IP addresses to make the intermittent network of roaming iButtons globally addressable.
- Standard cryptographic classes include SHA-1 for secret key digital signatures (RSA and DES for R1.1).
- Java-accessible random number generator seeds generation of cryptographic keys.

The Java Card framework defines a set of Application Programming Interface (API) classes for developing Java Card applications and for providing system services to those applications. Java Card applications are called applets. One Java iButton can hold multiple applets.

3.1.2 OpenCard Framework

The OpenCard Framework provides an Application Programming Interface (API), which allows programmers to develop applications in Java and make Java iButton use on different target platforms possible. The architecture is shown in Figure 2.

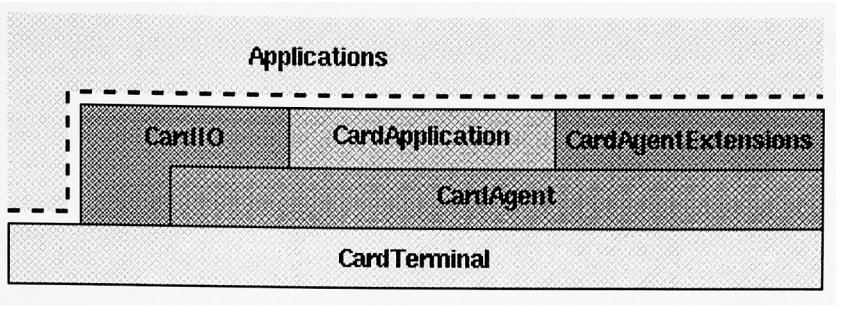


Figure 2. OpenCard Framework [7]

OpenCard consists of four Java packages with the prefix opencard:

- **Application and IO:** Provide the high-level API used by the application developer.
- **Terminal and Agent:** Provide the services needed by the high-level API.

3.1.3 Application Protocol Data Units (APDUs)

The basic unit of exchange with Java iButton is the APDU packet. The command message sent from Pc end, and the response message returned by Java iButton. APDU can be two kinds of format: Command APDU Format (Figure 3) and Response APDU Format (Figure 4).

Mandatory Header				Conditional Body		
CLA	INS	P1	P2	Lc	Data field	Le

Figure 3. Command APDU Format

CLA: Class byte. **INS:** Instruction byte. **P1-P2:** Parameter bytes.

Lc: The number of bytes in the data field of the command APDU.

Le: The maximum number of bytes expected in the data field of the following response APDU.

Conditional Body	Mandatory Trailer	
Data field	SW1	SW2

Figure 4. Response APDU Format

SW1-SW2: the processing status of the command APDU in a card.

3.2 Remote Method Invocation (RMI)

RMI is a specification for how Java objects can be accessed remotely. The specification contains the definitions of naming (how objects can be located and called remotely), marshalling (how an object should be converted into a byte stream and then converted it back to an object) and proxies (how method parameters and computation results are passed between Java Virtual Machines). These objects may reside on the same physical machine as the hosting JVM, or they may be located on other machines that are connected to the server through the network. The relationship between the client and server is pictured in Figure 5. We have to notice that Skeleton files are not

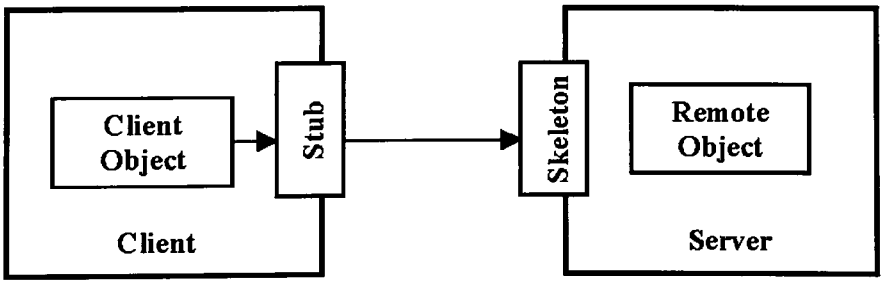


Figure 5. RMI Architecture [16]

required by RMI in Java 2. In java 2, un-packing will be managed by RMI libraries. We can break RMI into three major elements [10], and details are described as below:

- **Ability to locate remote objects:** A naming or directory service is used for locating remote object. RMI simply uses *rmiregistry* to obtain references to remote objects. Figure 6 shows the organization of a generic naming service.

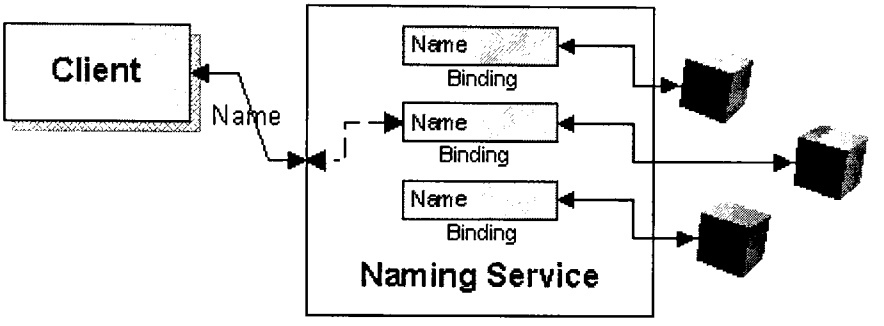
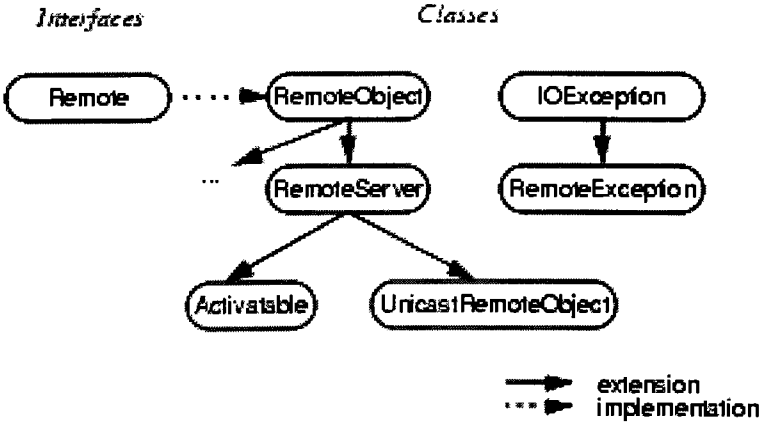


Figure 6. A generic naming service [11]

- **Ability to Communicate with remote objects:** Details of communication between remote objects are handled by RMI.

Figure 7. RMI handled remote objects communication [12]



- **Ability to Load class bytecodes for objects that are passed around:**

  In RMI's use of the Proxy pattern, the stub class plays the role of the proxy, and the remote service implementation class plays the role of the RealSubject. The proxy knows how to forward the method calls between the objects. See Figure 8.

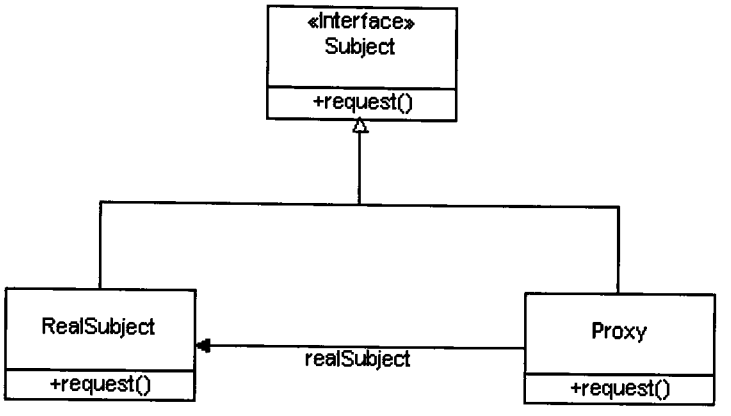


Figure 8. Proxy pattern [9]

3.3 Database Management

3.3.1 Java Database Connectivity (JDBC)

The JDBC API contains two major interfaces: the first is the JDBC API for application writers, and the second is the lower-level JDBC driver API for driver writers. JDBC drivers are divided into four types:

- **Type 1 – Connection through an ODBC Data Source:** This type of driver is useful because it allows you to use any database that's

accessible through ODBC, but it requires that ODBC be installed in each client machine. See Figure 9.

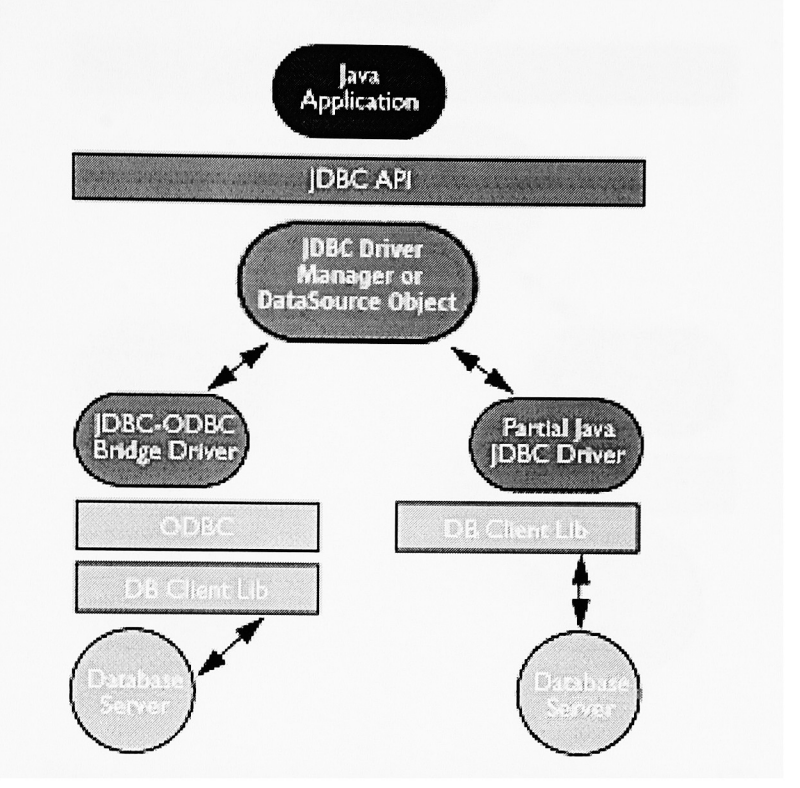


Figure 9. Type 1 and Type 2 JDBC Drivers [3]

- **Type 2 – Connection through Native Client Networking Code:** This type of driver includes both Java and native code, and it communicates with the client-side network software of a particular DBMS. See Figure 9.

- **Type 3 – Connection through Middleware:** This type of driver is written in pure Java. It requires a server-side component. See Figure 10.

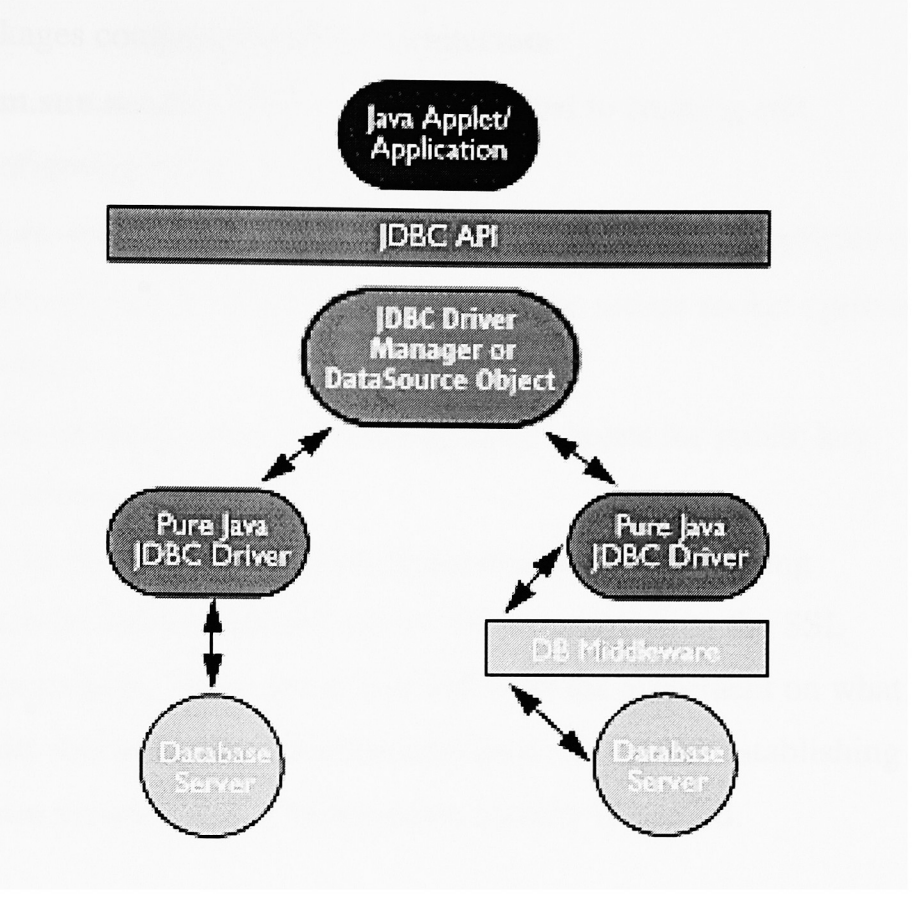


Figure 10. Type 3 and Type 4 JDBC Drivers [3]

- **Type 4 – Direct Connection to DBMS:** I use this type of driver in my project. It is easy to use because the only component needed is the driver itself. See Figure 10.

3.3.2 Database Management System (DBMS)

I use Cloudscape DBMS that is included in Java 2 Enterprise Edition. Cloudscape is a Java- and SQL-based object-relational database management system (ORDBMS).

3.4 Java 2 Security

3.4.1 Java Secure Socket Extension (JSSE)

Four packages compose the JSSE architecture:

- **com.sun.net.ssl:** Provides classes related to creating and configuring secure socket factories.
- **javax.net:** Provides optional classes for networking applications.
- **javax.net.ssl:** Provides the classes for the secure socket optional package.
- **javax.security.cert:** Provides optional classes for public key certificates.

I use SSL in my project. SSL handshake is used for exchanging information between client and server. By accomplishing the SSL handshake process, client and server will have the agreement on what cipher suite and encryption mechanisms they will use for establishing information security, and if authenticate identity is needed.

3.4.2 Java Cryptography Extension (JCE)

The design principles behind JCE use the same implementation as Java Cryptography Architecture (JCA) and algorithm independence by the use of the provider architecture. JCE also introduces some new interfaces to make it possible to use newer algorithm for generating key. I use SealedObject class to encrypt and decrypt all data that will be transferred across the network. By using an appropriately initialized Cipher object, we can encrypt and seal any serializable object.

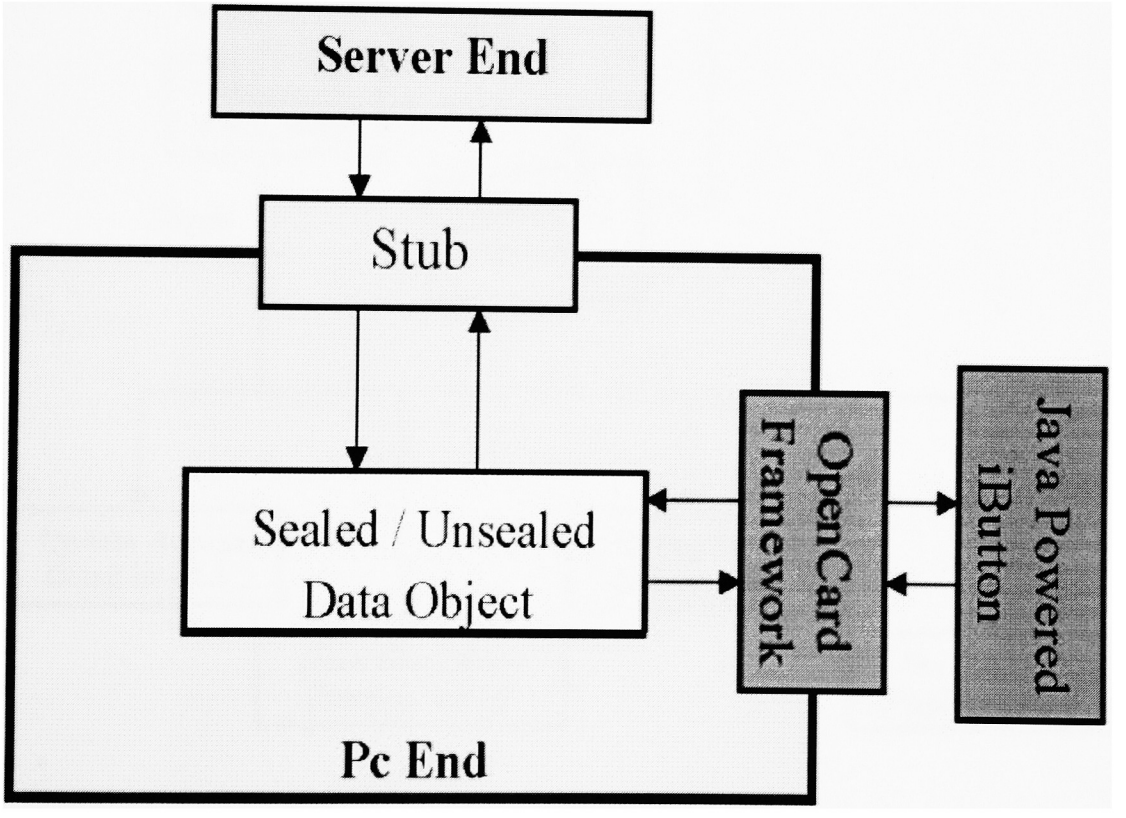
After having the basic idea of architecture used in my project, we will go to the design and implementation phase next. All designs are presented by flow chart.

4. Design and Implementation

Three categories (Pc End, Server End and Java iButton) are presented. I will explain them in detail as they ordered.

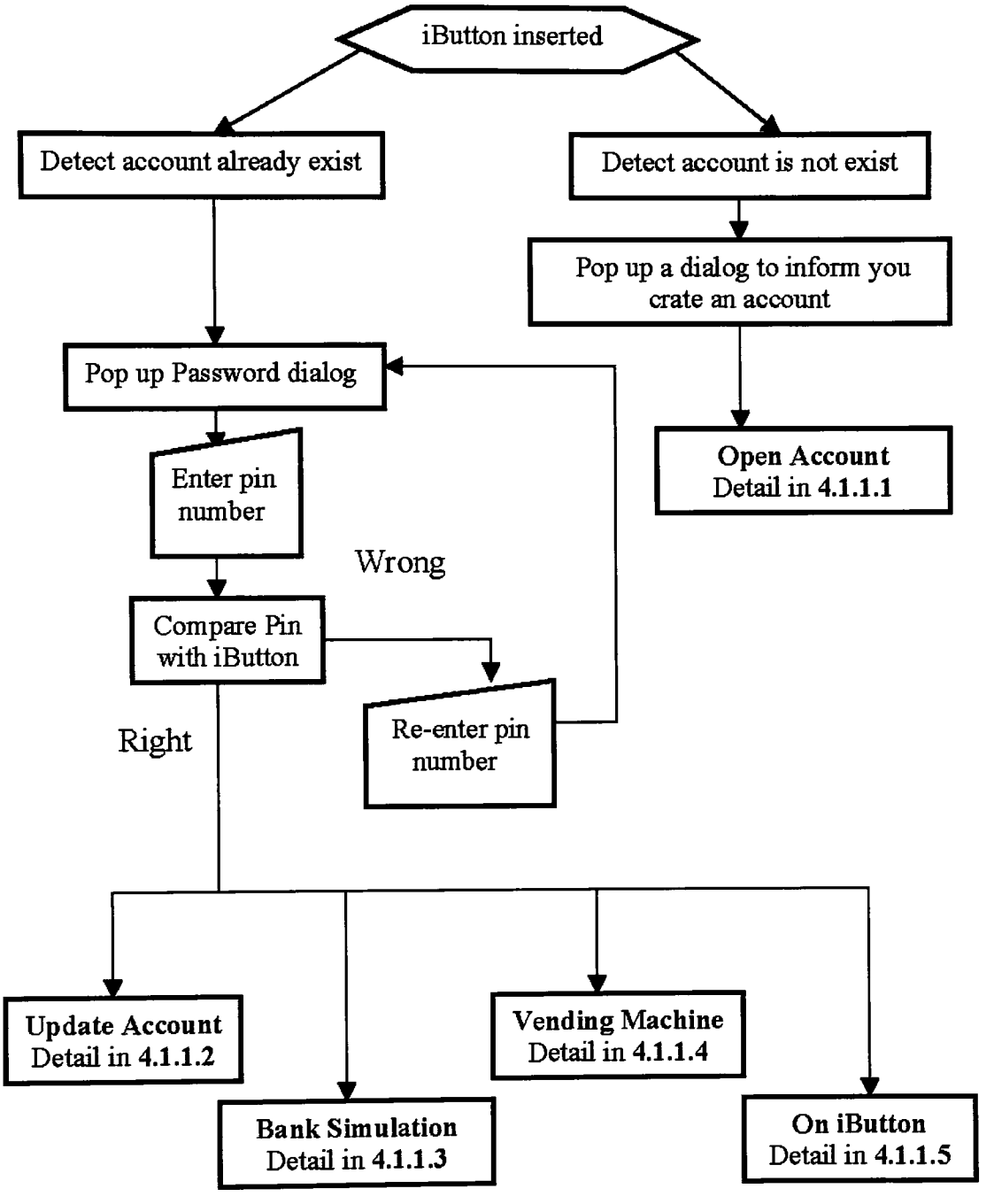
4.1 Pc End

It provides the GUI allowing client to do open/update account, bank simulation, shopping and check information on Java iButton. Pc End design is shown as below.



4.1.1 Function Specification

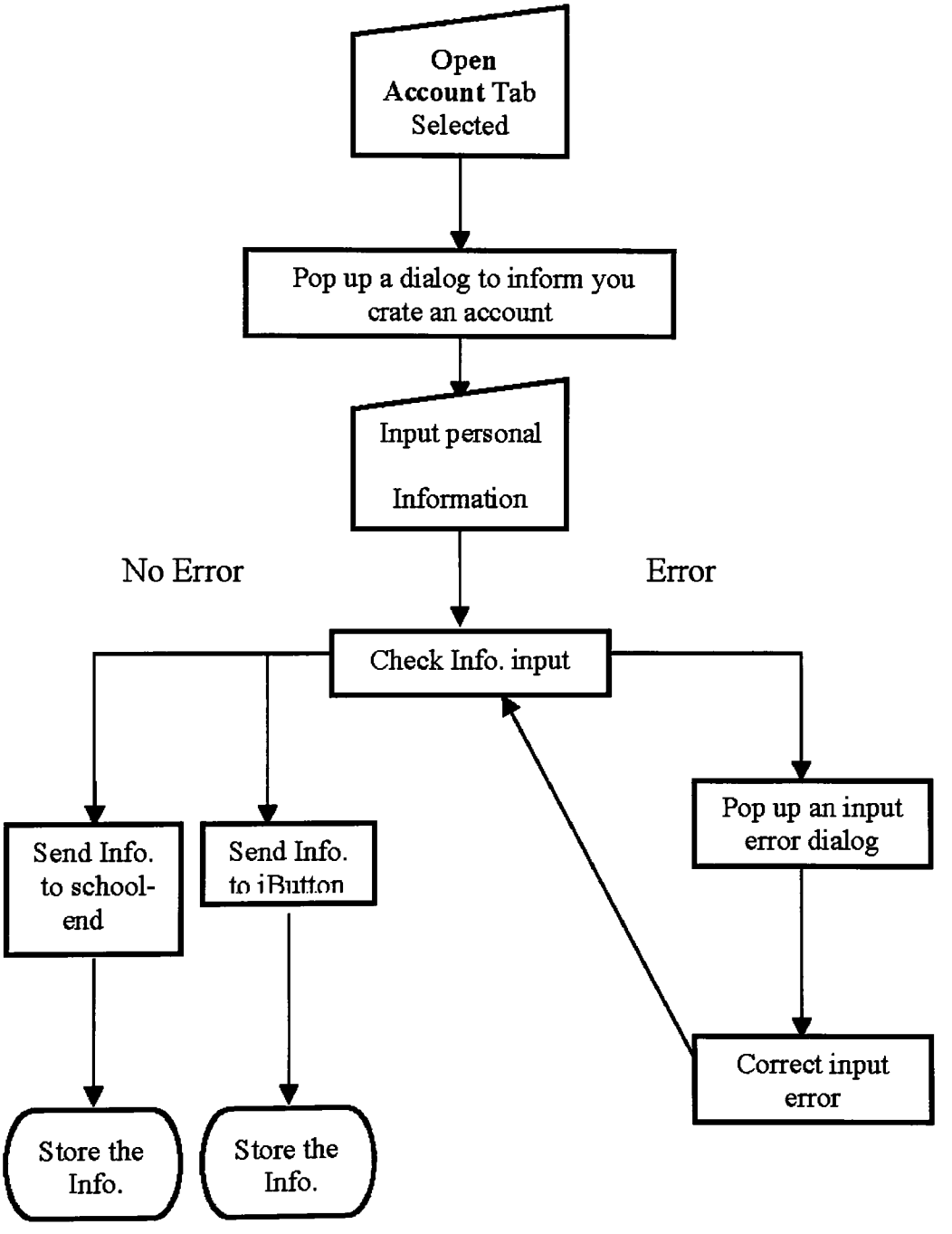
Figure in next page shows all functions provided by the program.



4.1.1.1 Create an Account

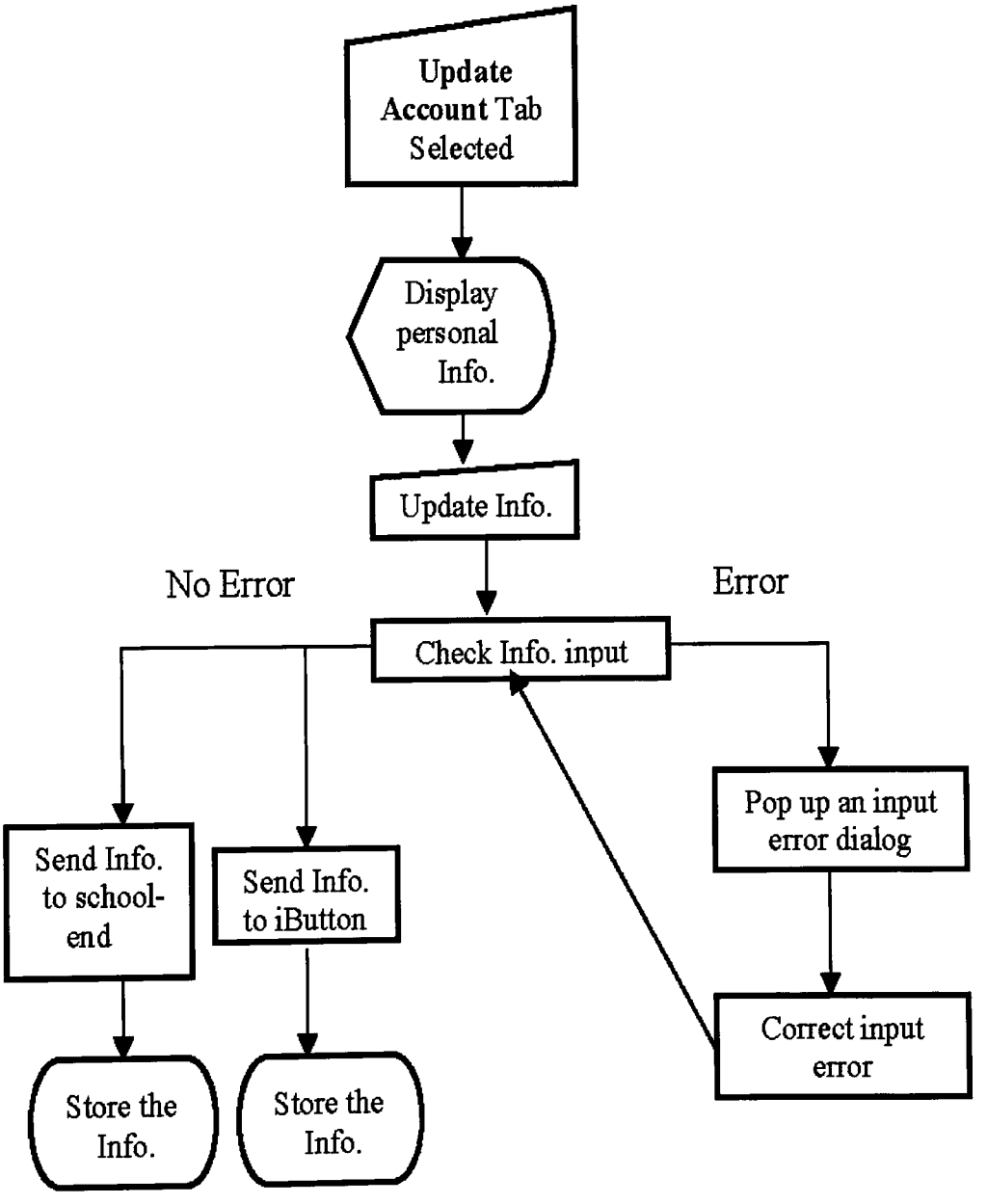
During the open and update account process, Java iButton must be always inserted. If it is removed during the process, an error dialog will pop up and all transaction will be aborted.

Create an account is selected for the client who doesn't have an account in our database. This will create a new account in database and store client information on both server end and Java iButton.



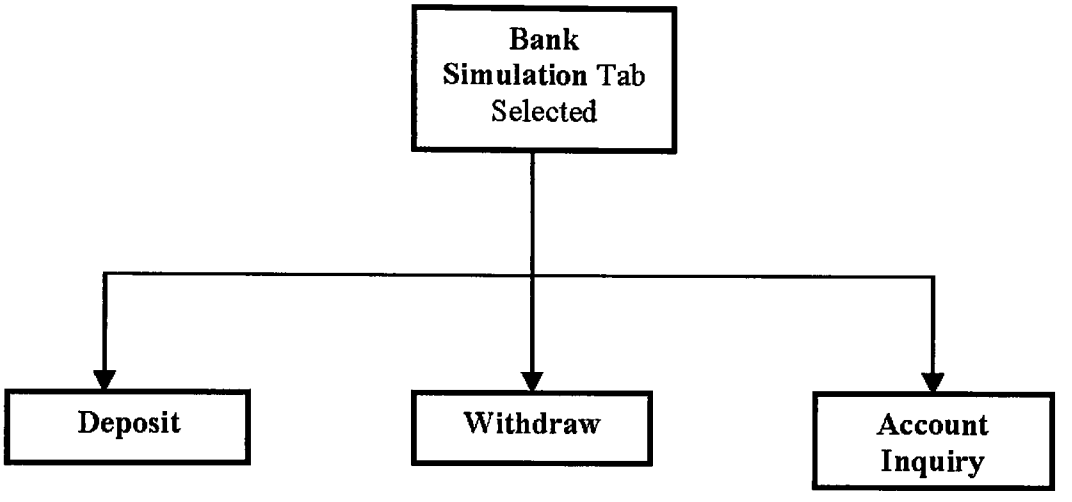
4.1.1.2 Update an Account

Update an account is selected when the client wants to change his/her personal information or pin number. This will update information on both server end and Java iButton.



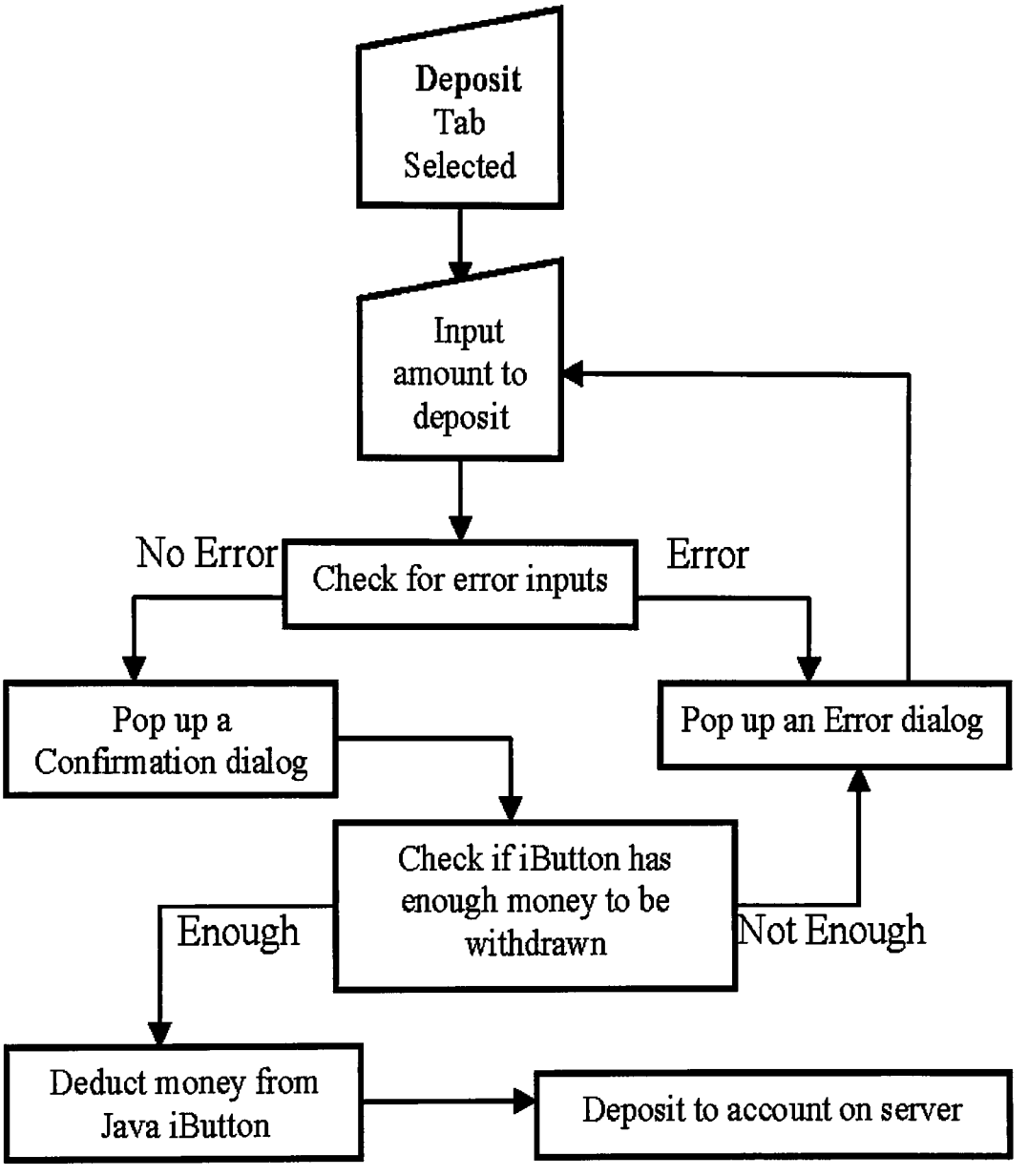
4.1.1.3 Bank Simulation

Bank Simulation is designed to demonstrate using Java iButton to deposit, withdraw and inquire account information. Java iButton must be always inserted. If it is removed during the process, an error dialog will pop up and all transaction will be aborted.



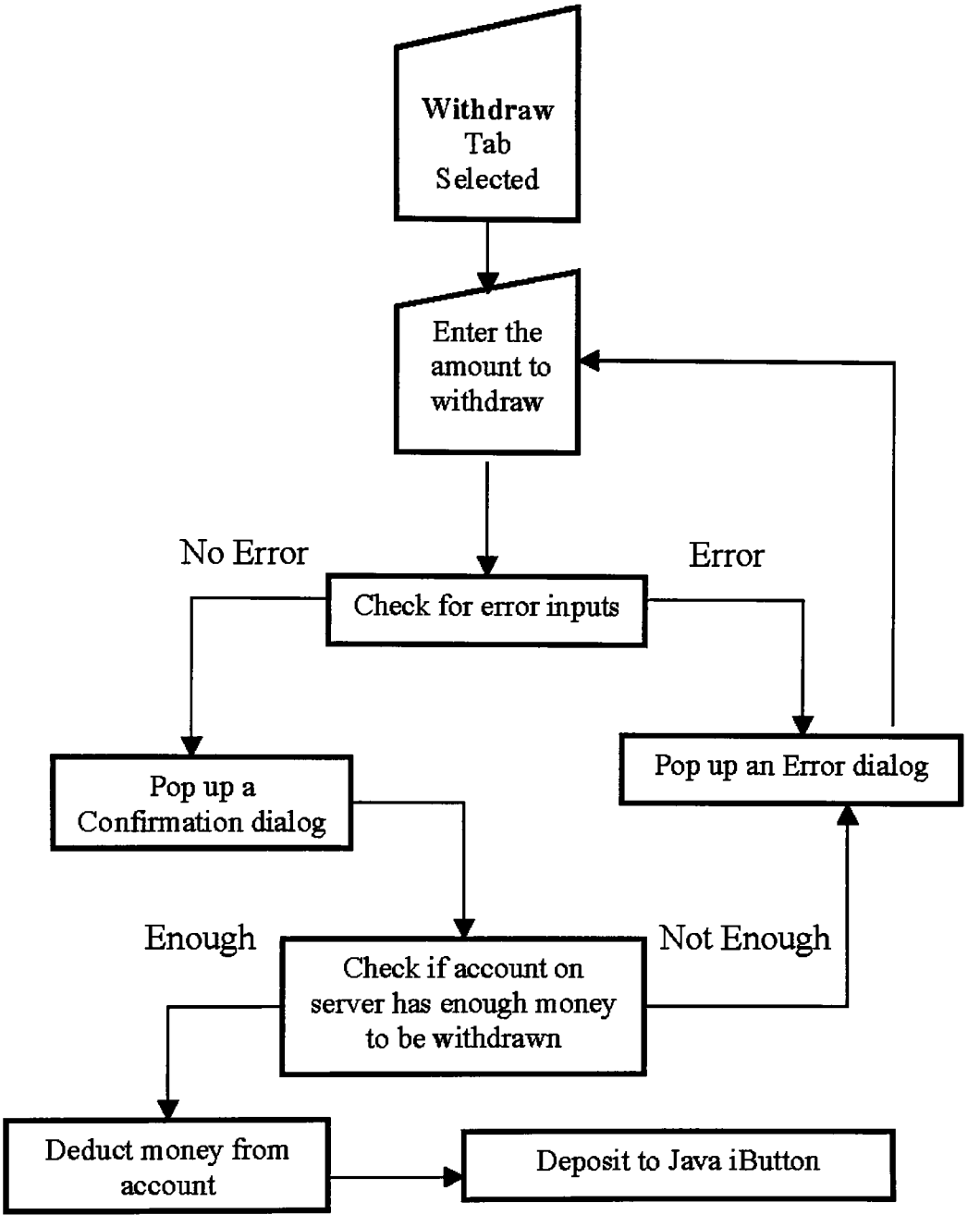
- **Deposit Money**

  An amount input will be checked during the deposit and withdraw process. If amount input equals to 0, an error will be thrown. And if deposit/withdraw amount exceeds $1,000,000, an error will be thrown. Deposit money is transferring money from Java iButton to server end. Input error checking will be performed before continue processing deposit request.



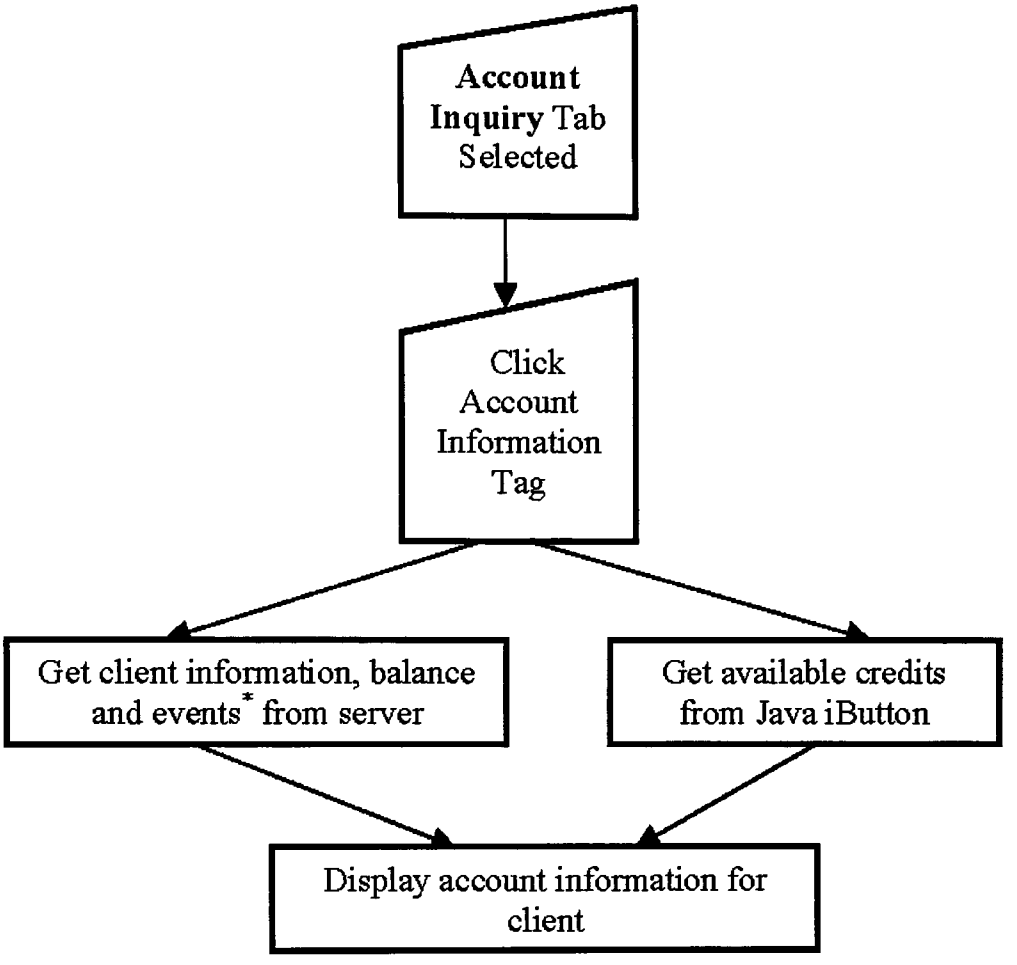
- **Withdraw Money**

  Withdraw money here means money will be withdrawn from server to Java iButton. All error checking will also be performed during the withdrawing process.



- **Account Inquiry**

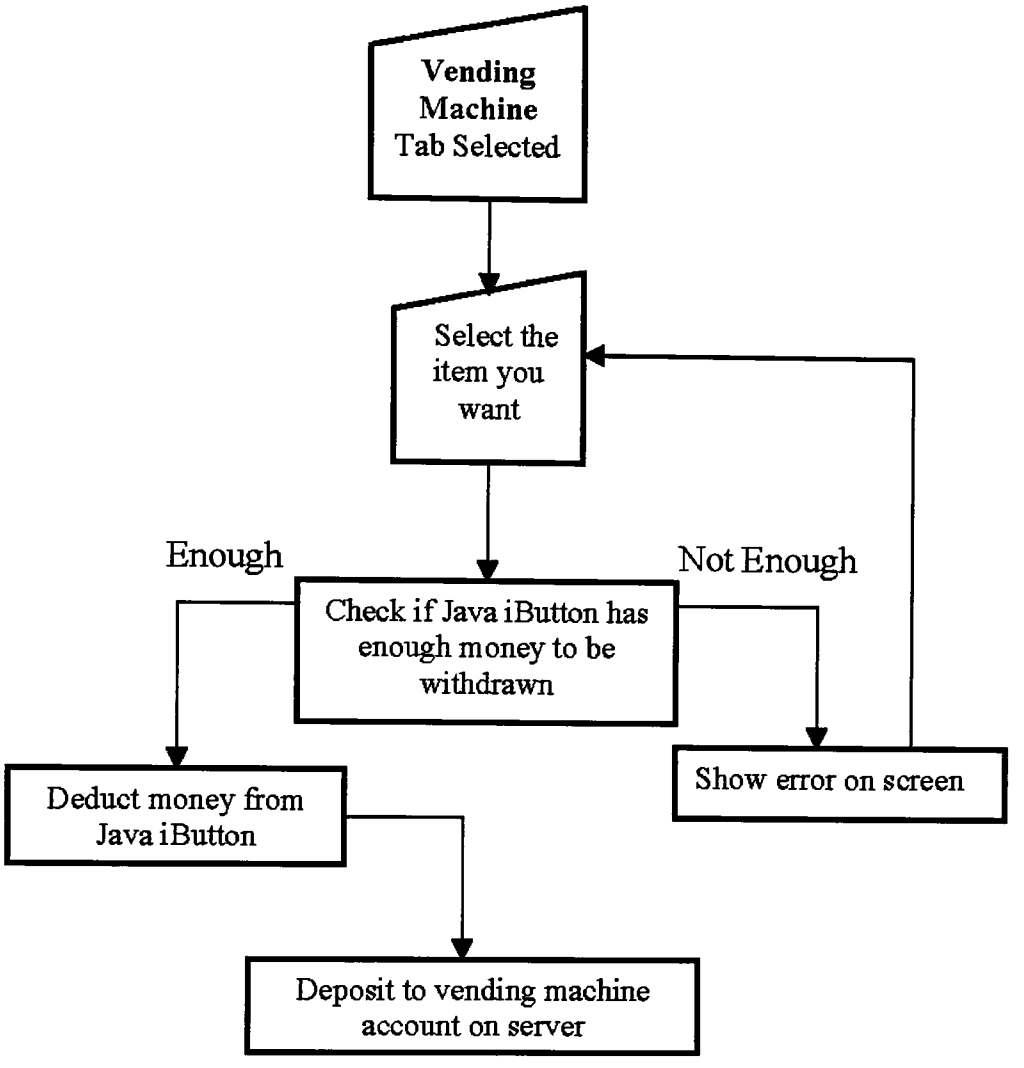
  When account inquiry tab is selected, client information will be gathered from server and available credits will be retrieved from Java iButton.



4.1.1.4 Vending Machine

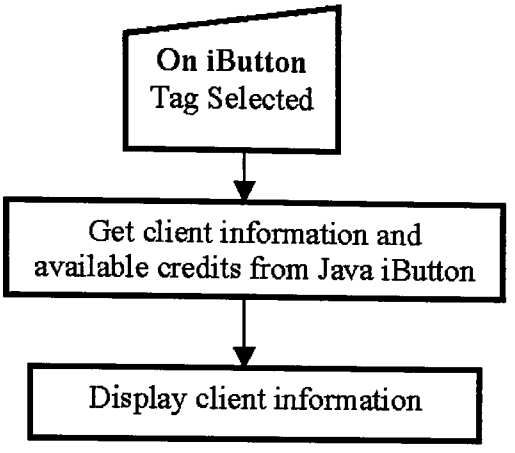
In this section, we test the Java iButton calculating ability. Vending machine is always connected with server so that all purchase amount can be transferred into vending machine's account on server. All transaction amounts are checked and deducted within Java iButton. It has the ability to process the data we passed in.

* events: It recorded all withdraw/deposit transactions made by clients.



4.1.1.5 Information on Java Powered iButton

This shows that Java iButton has the ability to store more information than the magnetic stripe card. The information stored on Java iButton will be: name, birthday, social security number, US address, US phone number, citizenship and available credits on Java iButton.



4.1.2 Secure Connection

In Java 2, we can custom our own socket type. Therefore, we can easily establish a secure connect with Java Secure Socket Extension (JSSE 1.0.2). Method Socket createSocket(String host, int port) in Interface RMIClientSocketFactory must be implemented with javax.net.ssl package. Because I put the server authentication in my design, we have to import server's certification and marked it as trusted certification. Or we will get an exception.

4.1.3 Data Encryption and Decryption

When using SSL connection, all data are encrypted before they are passed through the network. I also use class SealedObject in Java Cryptography Extension (JCE 1.2.1) to encrypt and decrypt data including client information object data type, account number, and

transaction amount. Data encrypted twice will be much safer than data encrypted once.

4.2 Server End

The basic server end design is pictured in Figure 11.

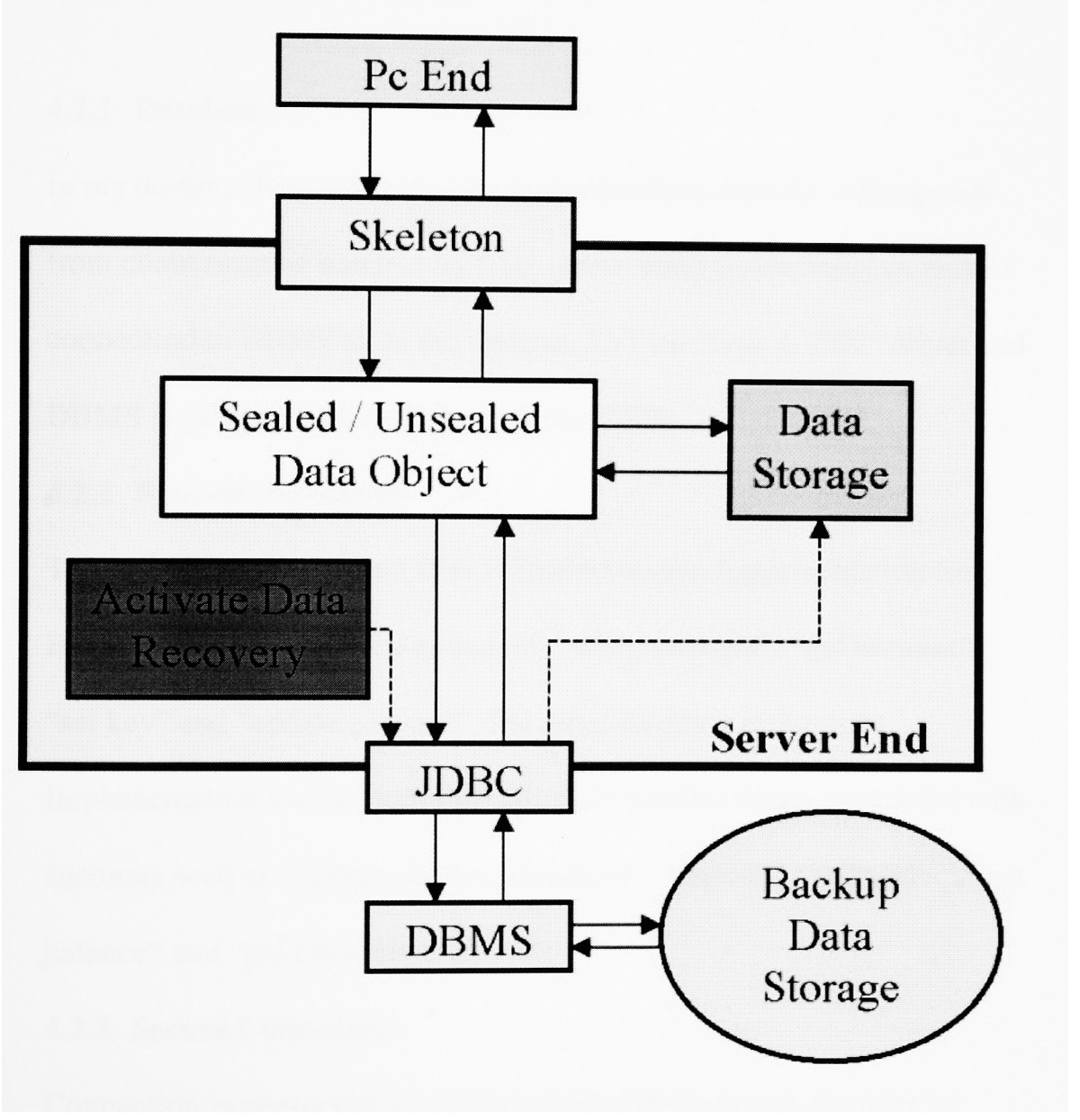


Figure 11. Server end design

One thing we have to notice is when server is down, both client and server have to restart. But all data will be recovered from database as soon as server reboots. All accounts will be recovered and reconstructed by calling recovery method.

4.2.1 Database

In my design, client can't connect to the database directly. All requests from client must be handled by RMI server. RMI server can then make a connection to DBMS to do the updates. So I use Type 4 JDBC driver and DBMS is using Cloudscape that is in the J2EE.

4.2.2 Request Processing

Two interfaces are created. One is BankManager. Implementation for BankManager interface takes care of "create account", "get account", "set key" and "update account". The other interface is Account. Implementation for Account interface only handles things associated with accounts such as "update client information", "deposit", "withdraw", "get balance" and "get client information".

4.2.3 Secure Connection

Connection in server end is different from the client end. In order to establish a secure connection, we have to do several things. First thing is

similar to pc end. We have to implement method ServerSocket createServerSocket (int port) in Interface RMIServerSocketFactory. Second, server has to set up key manager to do server authentication. Third, we have to export server's certification to pc end.

4.2.4 Data Encryption and Decryption

Server end uses the same data encrypt / decrypt method as pc end.

4.3 Java Powered iButton

4.3.1 Bank Applet

Applet run on the Java iButton is similar to Applet programming. Applet is designed to cooperate with pc end program. Bank applet can process: store client information (including pin), deposit, withdraw, get credits, and pin check. ISOException will be thrown if any error occurred during data processing. All data are processed within Java iButton.

4.3.2 Pin Authentication

Java iButton is JavaCard 2.0-compliant. I use javacard.framework package to do pin authentication. I set 3-minute timeout if user input wrong pin 8 times in a roll. Max. Pin length is set to 9. These values can be changed at anytime. The only reason I set these values is for easy debugging.

4.4 Classes

Classes in my project are divided into five packages:

- **Package project.interfaces:** Contains remote interfaces, exceptions thrown by remote interfaces , implementation for remote interfaces and data classes.
    - ***project.interfaces.Account:*** Remote interface.
    - ***project.interfaces.BankManger:*** Remote interface.
    - ***project.interfaces.Info:*** Data class for holding client information.
    - ***project.interfaces.AccountImpl:*** Implementation for Account interface.
    - ***project.interfaces.BankManagerImpl:*** Implementation for BankManager interface.
- **Package project.pcend:** Contains the main classes for pc end program.
    - ***project.pcend.PcEnd:*** This is the heart of the pc end program.
    - ***project.pcend.BankUser:*** Contains the main function.
    - ***project.pcend.ErrorDialog:*** Error dialog.
    - ***project.pcend.AnotherTransaction:*** AnotherTransaction dialog.
    - ***project.pcend.Confirmation:*** Deposit /withdraw confirmation.
    - ***project.pcend.PasswordCheck:*** Password check dialog.

- ***project.pcend.InfoError:*** Information input error dialog.
- ***project.pcend.BankHost:*** This class handles all Java iButton processes on pc end.
- ***project.pcend.BankHostException:*** Exceptions thrown by Java iButton during data processing.

- **Package project.security:** Contains SSL connection and object encryption and decryption .
  - ***project.security.ObjectDecryption:*** Doing data decryption.
  - ***project.security.ObjectEncryption:*** Doing data encryption.
  - ***project.security.RMISSLClientSocketFactory:*** Create client socket.
  - ***project.security.RMISSLServerSocketFactory:*** Create server socket.
- **Package project.server:** Contains server class.
  - ***project.server.ServerSystemServer:*** Main function of server end.
  - ***project.server.AccountDB:*** This class will create the table AccountData in database.
- **Package project.iButton:** Contains only one class for Java iButton.

- ***project.iButton.BankApplet:*** This applet will run on the Java iButton.

Next will be the users manual which covers environment setting, program installation and testing.

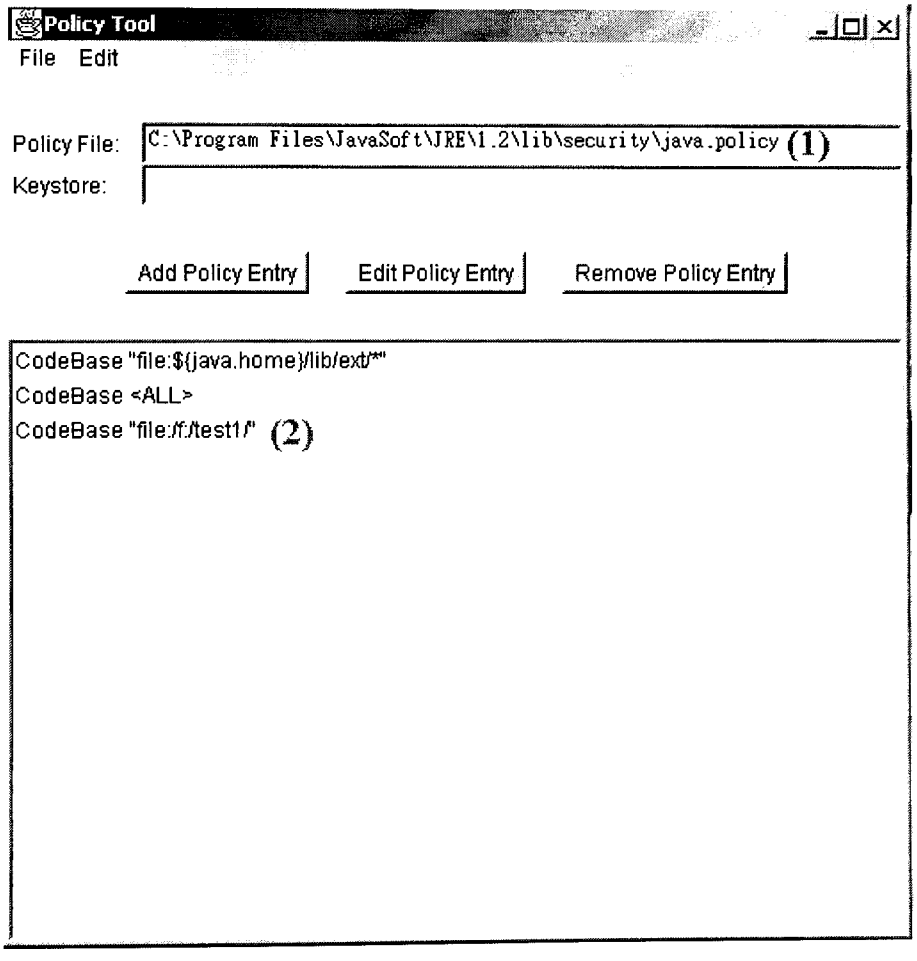
5. Users Manual

I assume that the user already setup jdk1.2.2 and j2sdkee1.3 appropriately.

5.1 Configuring Policy and Security Files

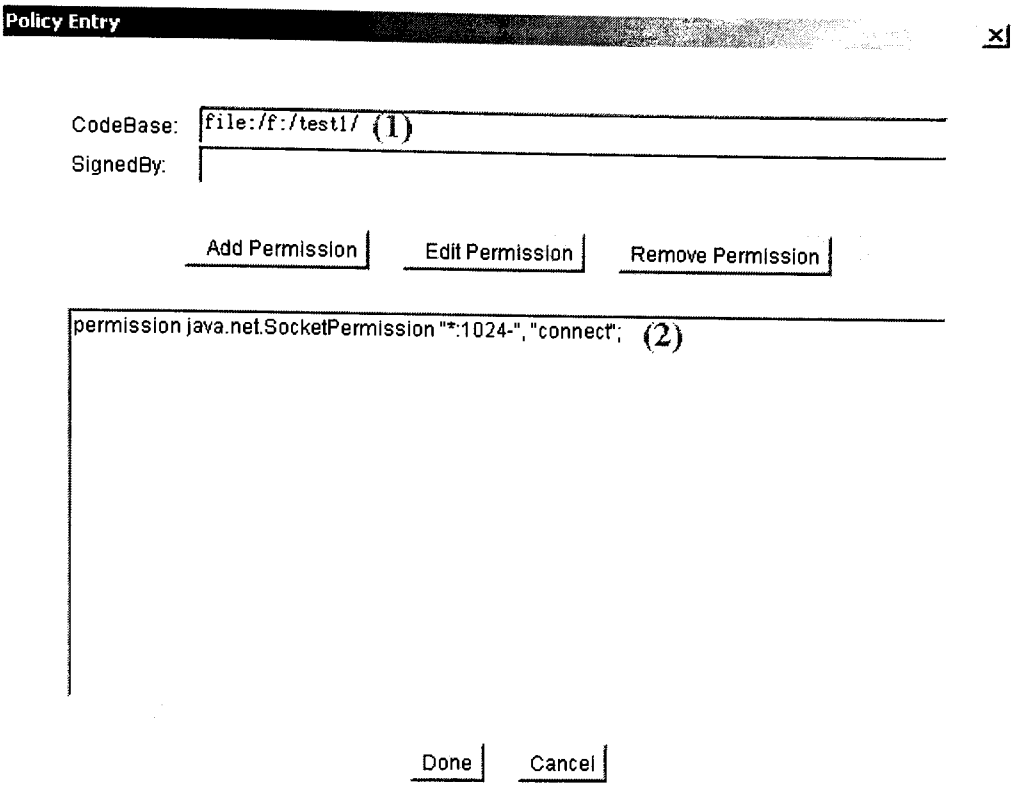
- **Policy file**

You can use *policytool* command to edit or create policy file. Only server needs this policy file. For example,



(1) Location of your policy file.

(2) Policy Entry we add.



(1) Your working root.

(2) Give socket permission for RMI and JDBC use.

- **Security file**

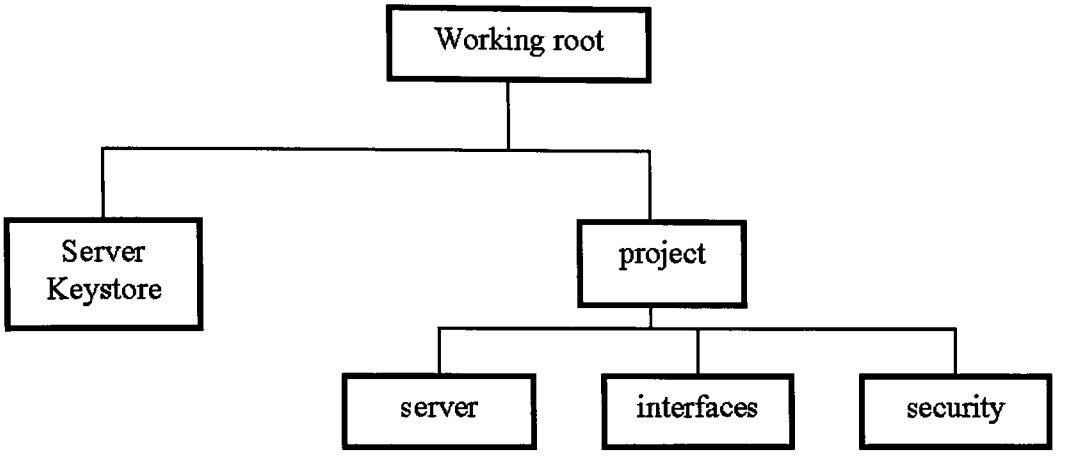
Because we will use JCE and JSSE, we have to install these two APIs first. Both client and server need to configure this file. Installation guides can be found in Appendix B. After we add two lines into security file, installation is complete.

```
security.provider.2=com.sun.crypto.provider.SunJCE
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
```

5.2 Server End Setting on Solaris 8 and Windows 2000

Execute jar xvf server.jar to extract server files. After you extracted that jar

file, you can see the file structure as following.



5.2.1 Environment Setting

- **Solaris 8**

  Set the environment variable J2EE_HOME to where you installed J2EE. For example,

  setenv J2EE_HOME=/home/stu4/s16/yxc3650/j2sdkee1.3

  Set the environment variable JAVA_HOME to where you installed J2SE. For example,

  setenv JAVA_HOME=/ home/stu4/s16/yxc3650/jdk1.2.2

- **Windows 2000**

  Set the environment variable J2EE_HOME to where you installed J2EE. For example,

  set J2EE_HOME=C:\j2sdkee1.3

  Set the environment variable JAVA_HOME to where you installed J2SE. For example,

  set JAVA_HOME=C:\jdk1.2.2

- **Both for Solaris 8 and Windows 2000**

  To avoid classpath issues, copy and rename the following jars from J2EE_HOME/lib/cloudscape and J2EE_HOME/lib/system to JAVA_HOME/jre/lib/ext:

  cloudscape.jar , client.jar *rename* to cs_client.jar

  RmiJdbc.jar *rename* to cs_RmiJdbc.jar

  tools.jar *rename* to cs_tools.jar

  Also remember to include J2EE_HOME/bin into your CLASSPATH.

5.2.2 Running Server End Program

- **Both for Solaris 8 and Windows 2000**

  If this is the first time to run server program, follow these steps to create account database first.

  (1) cloudscape –start - to start Cloudscape DBMS.

  (2) java -Dcloudscape.system.home=J2EE_HOME/cloudscape project.server.AccountDB HostName* PortNumber* - to create AccountDB database for first time use. You will see message "Table AccountData was created" later.

  * HostName: server IP. * PortNumber: port number

  In working root, execute these commands in order:

  (1) rmiregistry 1234 - to start rmiregistry.

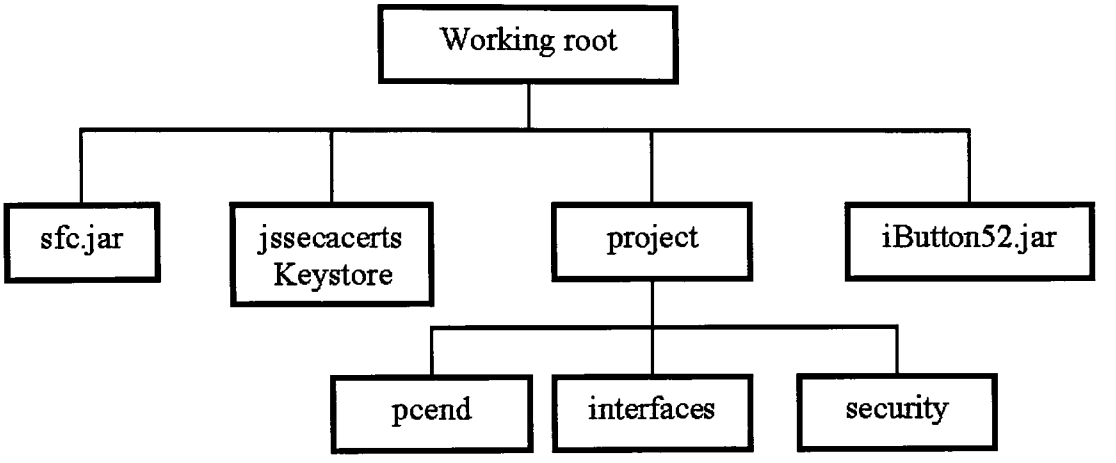
  (2) java –Dcloudscape.system.home=J2EE_HOME/cloudscape -Djava.rmi.server.codebase="file:/full path*/" project.server.BankSystemServer HostName PortNumber – to start server service.

  *full path: full path to your working root.

After these steps, you will see message "Server started".

5.3 Pc End Setting on Solaris 8 and Windows 2000

Execute jar xvf client.jar to extract client files. After you extracted that jar file, you can see the file structure as following.



Put sfc.jar* and iButton52.jar* into JAVA_HOME/jre/lib/ext/ and put jssecacerts into JAVA_HOME/jre/lib/security/ .

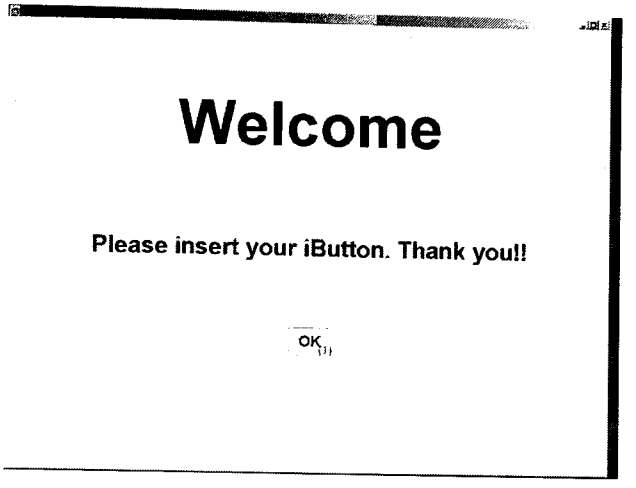
*sfc.jar: contains classes for GUI. *iButton52.jar: contains JavaCard API.

5.3.1 Running Pc End Program

In working root, execute:

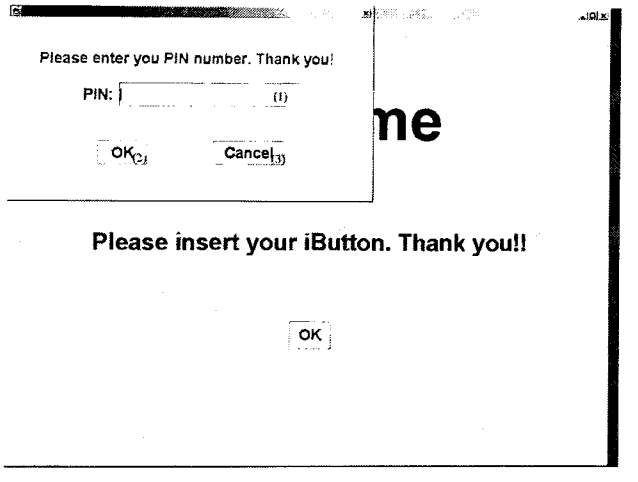
java project.pcend.BankUser HostName PortNumber – to start pc end program. You will see the following GUI.

- **Welcome page**



Click (1) OK button, it will bring out open account page if the account didn't exist. Or pop up pin check dialog if account exists.

- **Pin check dialog**

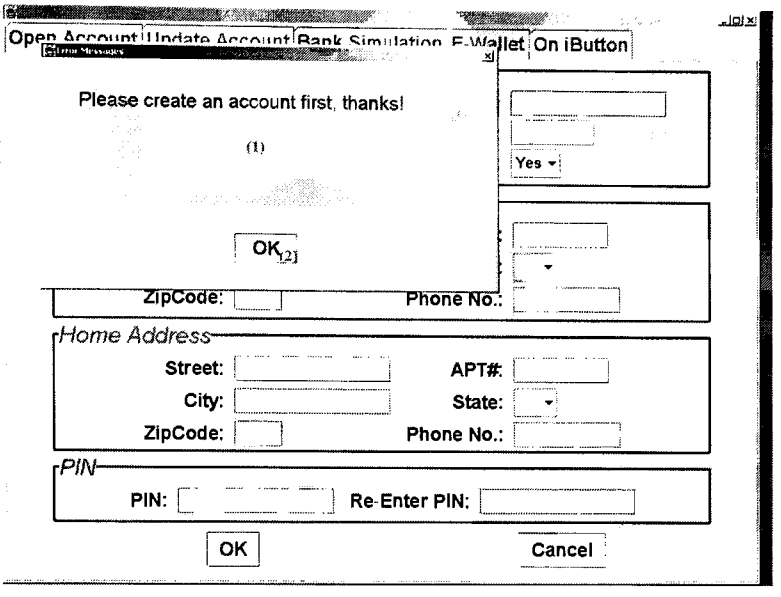


In (1) field, input your pin no more than 9 letters.

Click (2) OK button to check pin.

Click (3) Cancel button to abort.

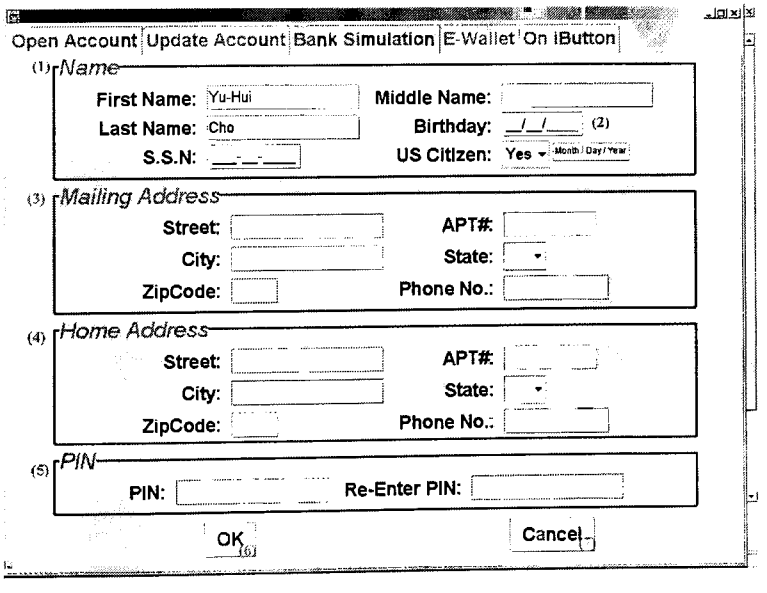
- **Open Account page**



(3) shows need to create account first message.

Click (2) OK button to clear the message dialog.

After clear the massage dialog, you can start to input information.



In (1) section, input your name, SSN, citizenship and birthday.

Notice (2) Birthday field, the format is month / day / year

In (3) section, input your US address and phone number.

In (4) section, input your home address and phone number.

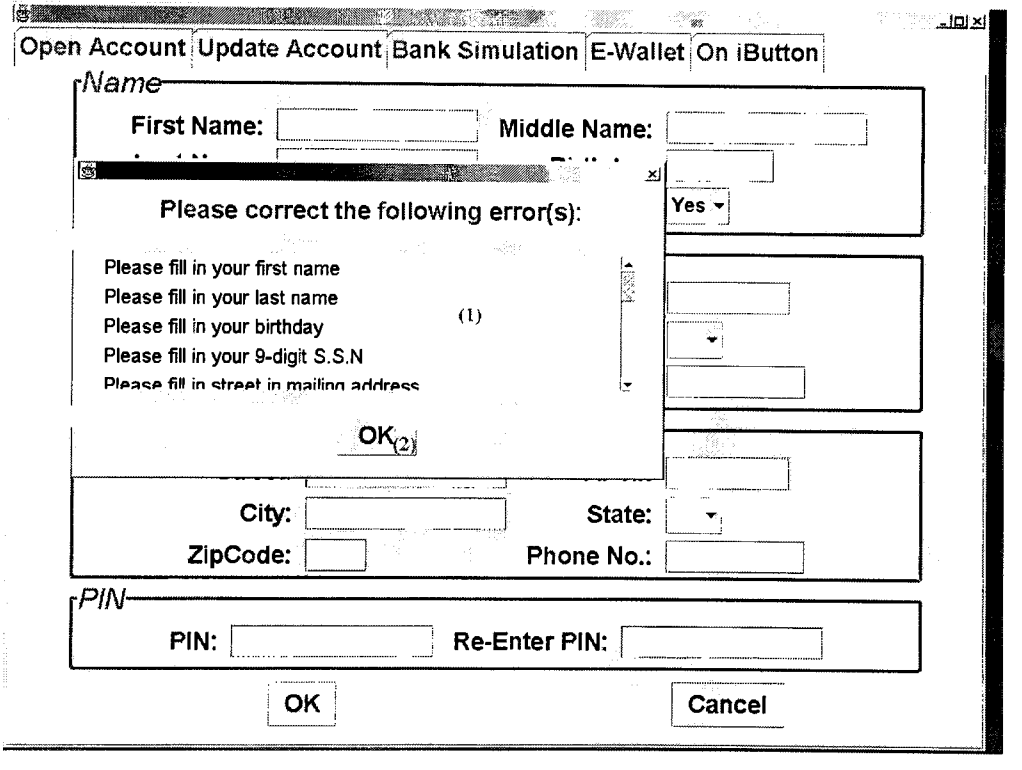
In (5) section, input your pin number.

Click (6) to store information.

Click (7) to abort and back to Welcome page.

- **Error input dialog**

If input contains error(s), an error dialog will pop up.



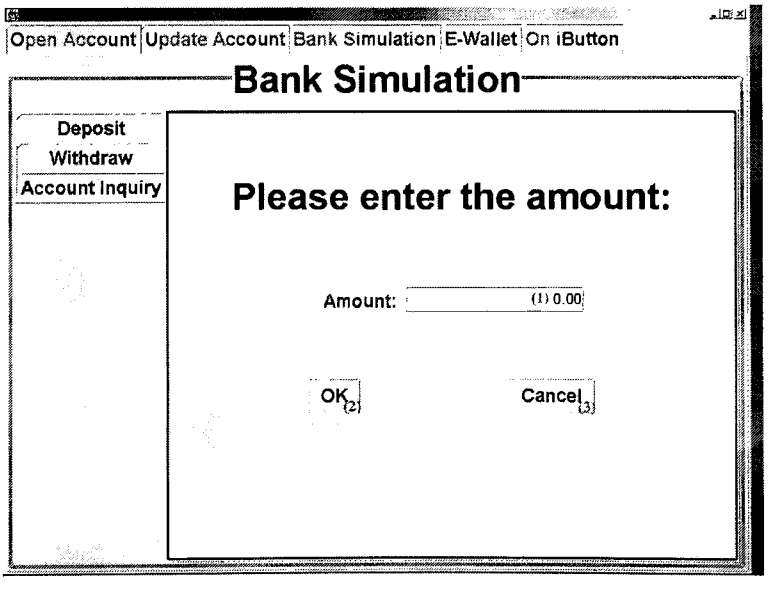
In (1) section, it shows error(s) to be corrected.

Click (2) OK button to clear dialog.

- **Bank Simulation page**

Click (1) OK button to proceed bank simulation.

- **Deposit page**

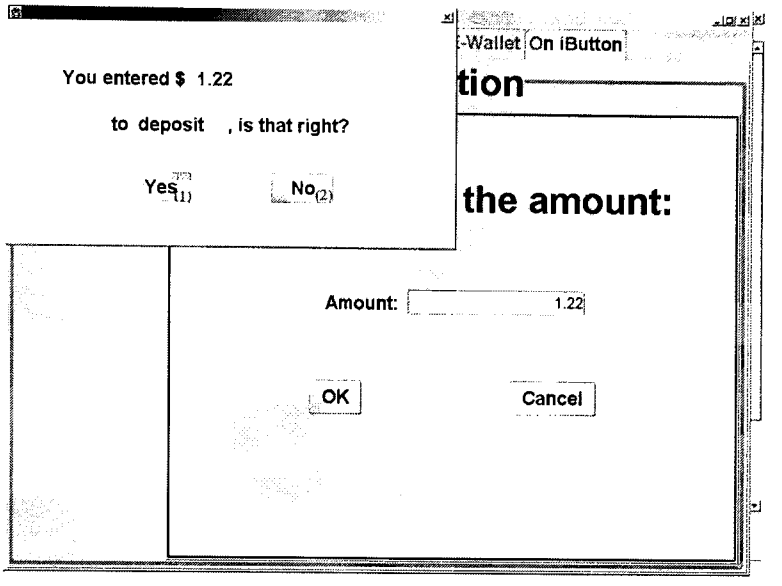


In (1) field, input the amount you want to deposit. If you didn't put any amount and you click (2), you will get an error.

Click (2) OK button to proceed transaction.

Click (3) Cancel button to abort.

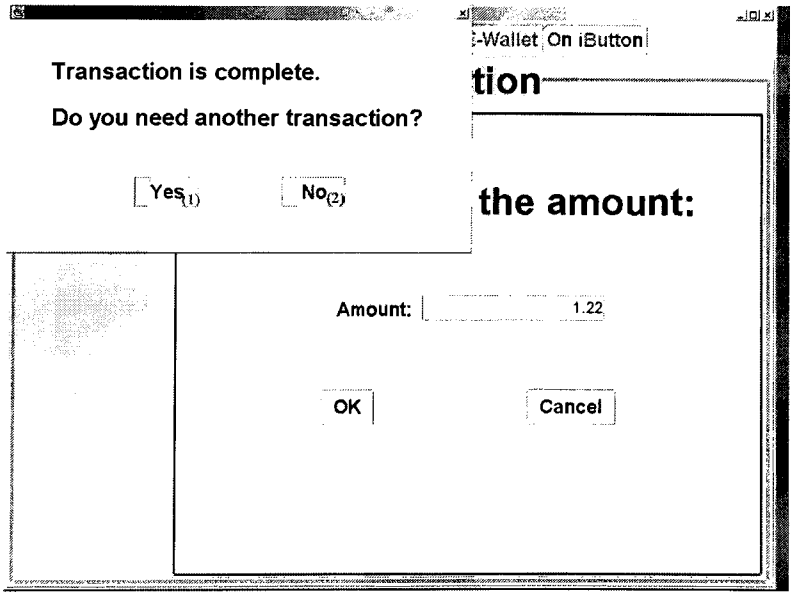
- **Transaction Confirmation Dialog**



Click (1) OK button to proceed transaction.

Click (2) No button to re-enter deposit amount.

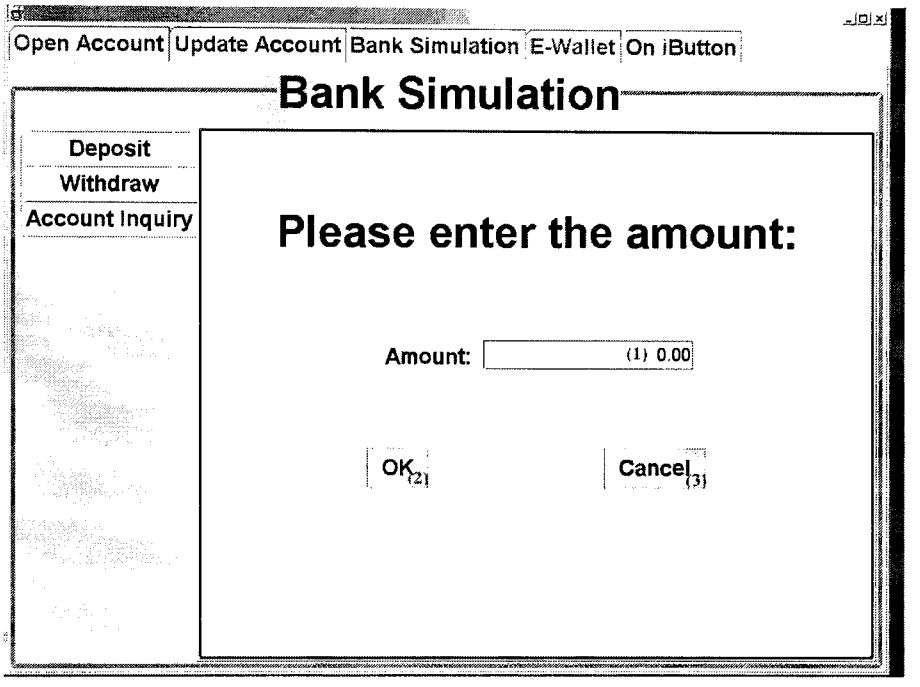
- **Another Transaction dialog**



Click (1) Yes button to continue another transaction.

Click (2) No button to terminate this section.

- **Withdraw page**

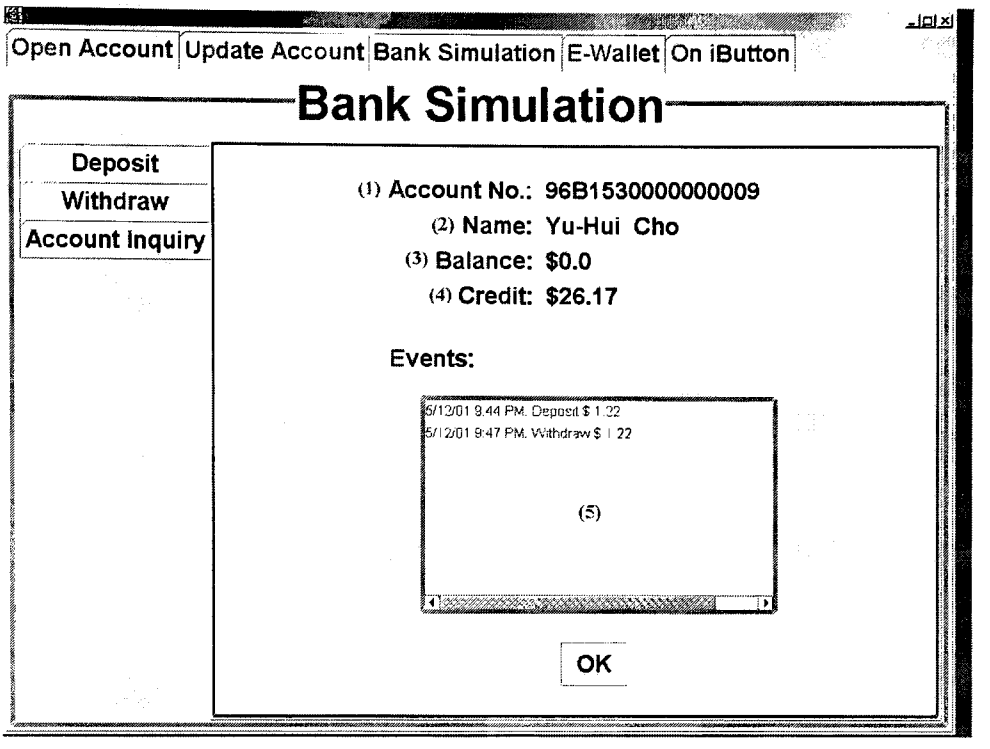


In (1) field, input the amount you want to withdraw. If you didn't put any amount and you click (2), you will get an error.

Click (2) OK button to proceed transaction.

Click (3) Cancel button to abort.

- **Account Inquiry page**



(1) shows account number.

(2) shows client's name.

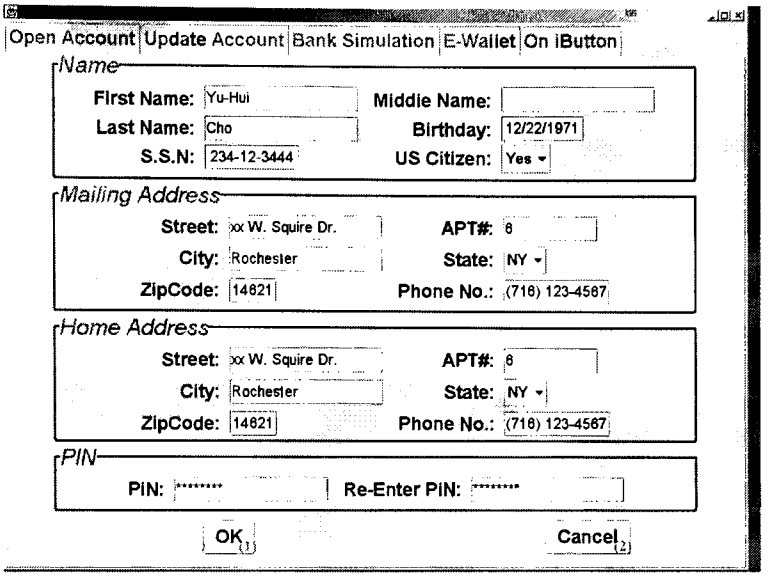
(3) shows balance on server.

(4) shows credits available on Java iButton.

(5) records date and time when withdraw / deposit transaction occurred.

Click OK button will bring up another transaction dialog.

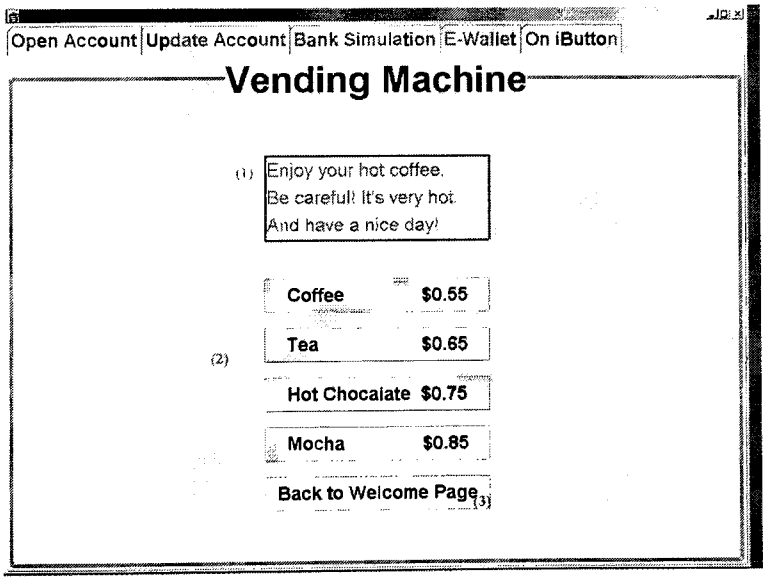
- **Update Account Page**



Click (1) OK button to update account.

Click (2) Cancel button to abort.

- **E-Wallet page**

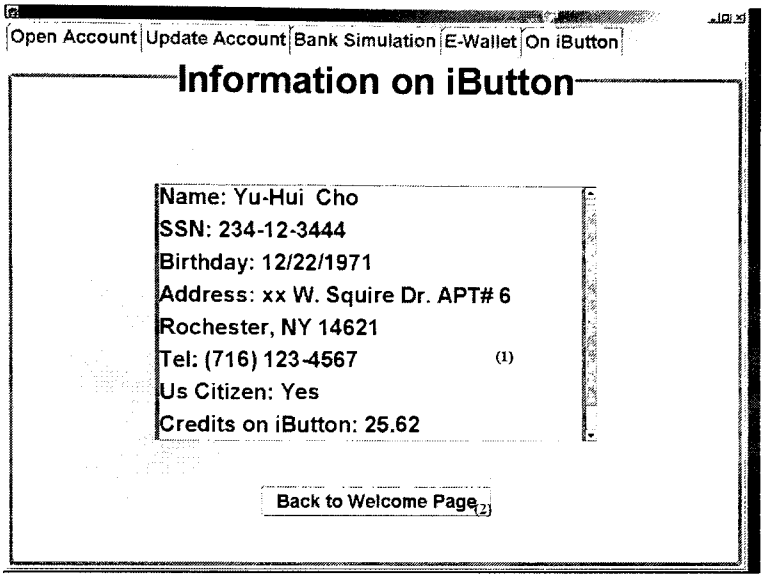


(1) shows messages (including error messages).

(2) Merchandises

Click (3) to go back to Welcome page.

- **ON iButton page**

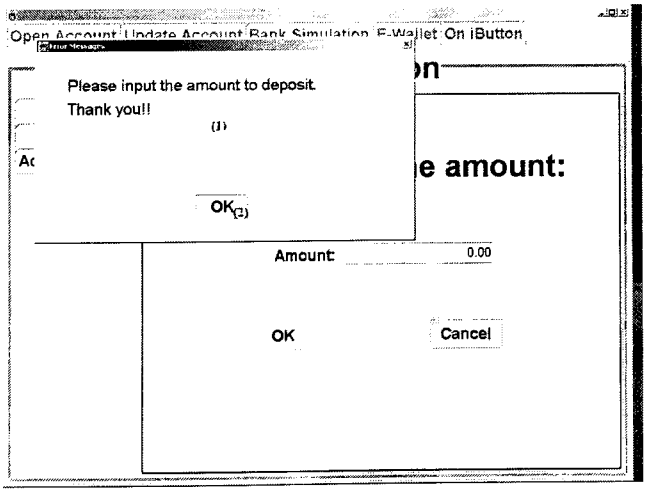


(1) shows information retrieved from Java iButton.

Click (2) to go back to Welcome page.

You may see the following error dialog when you use this program.

- **No amount input error dialog**

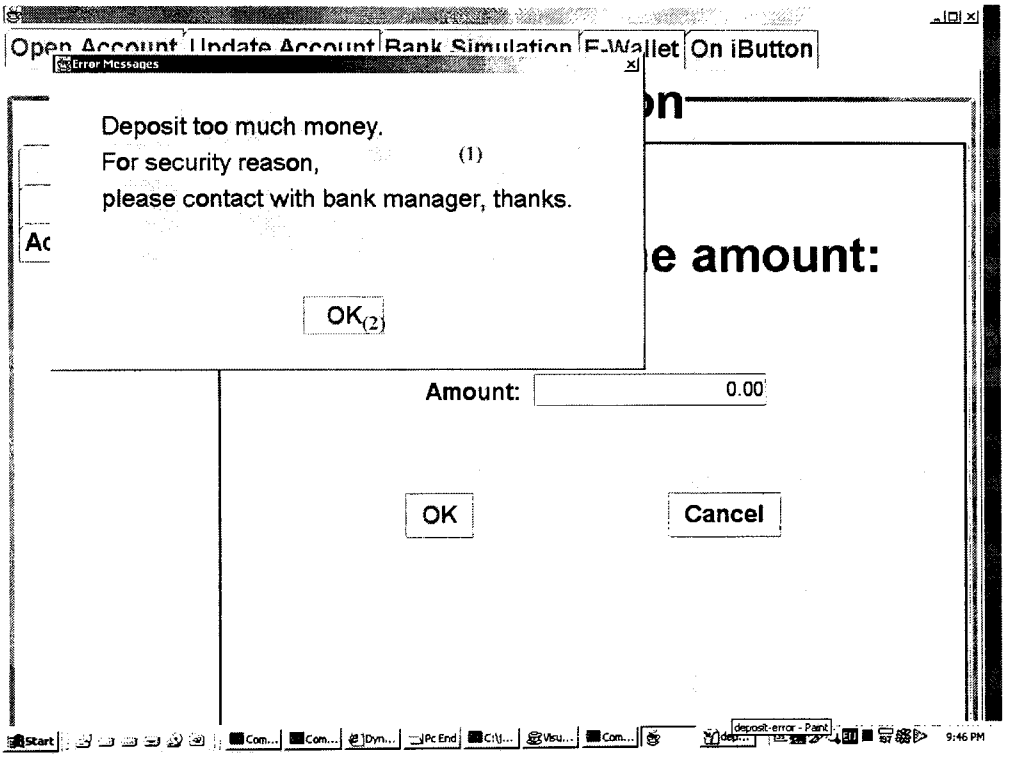


This dialog will pop out when no amount input but click OK button.

(1) shows error message.

Click (2) OK button to clear dialog.

- **Input too much amount error dialog**

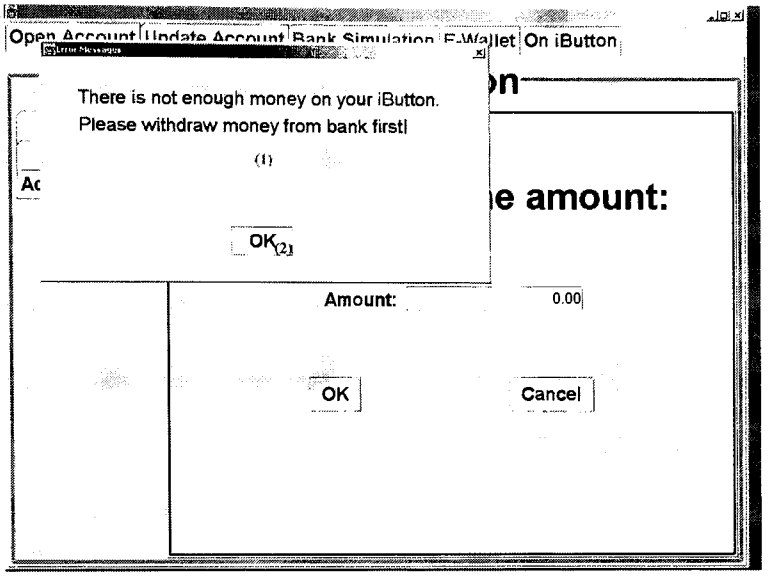


This dialog will pop out when input over $1,000,000.

(1) shows error messages.

Click (2) OK button to clear dialog.

- **Not enough money error dialog**

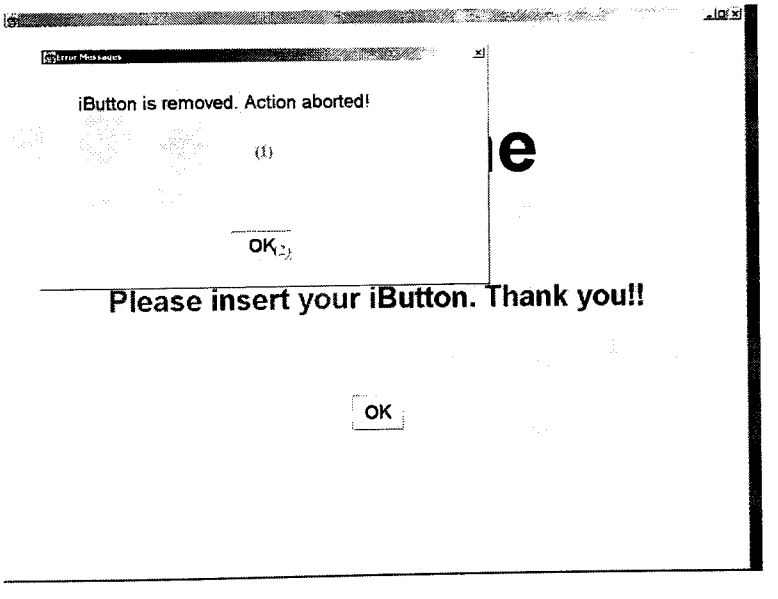


This dialog will pop out when there is not enough money on Java iButton or account.

(1) shows error messages.

Click (2) OK button to clear dialog.

- **Java iButton removed error dialog**

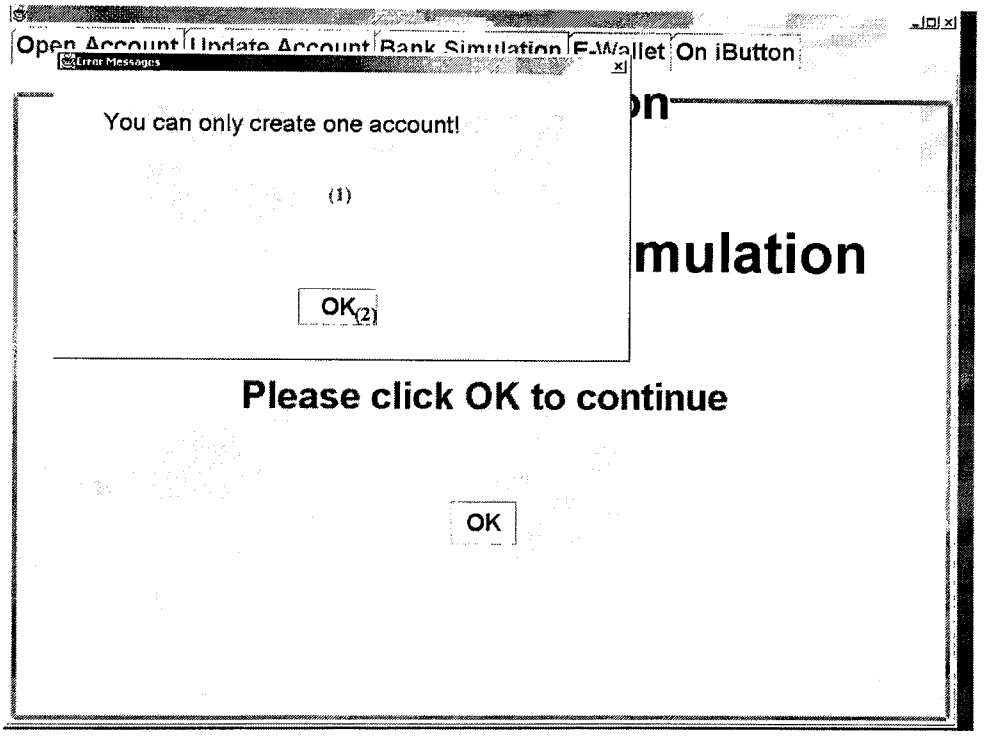


This dialog will pop up when Java iButton is removed.

(1) shows error messages.

Click (2) OK button to clear dialog.

- **Create account error dialog**

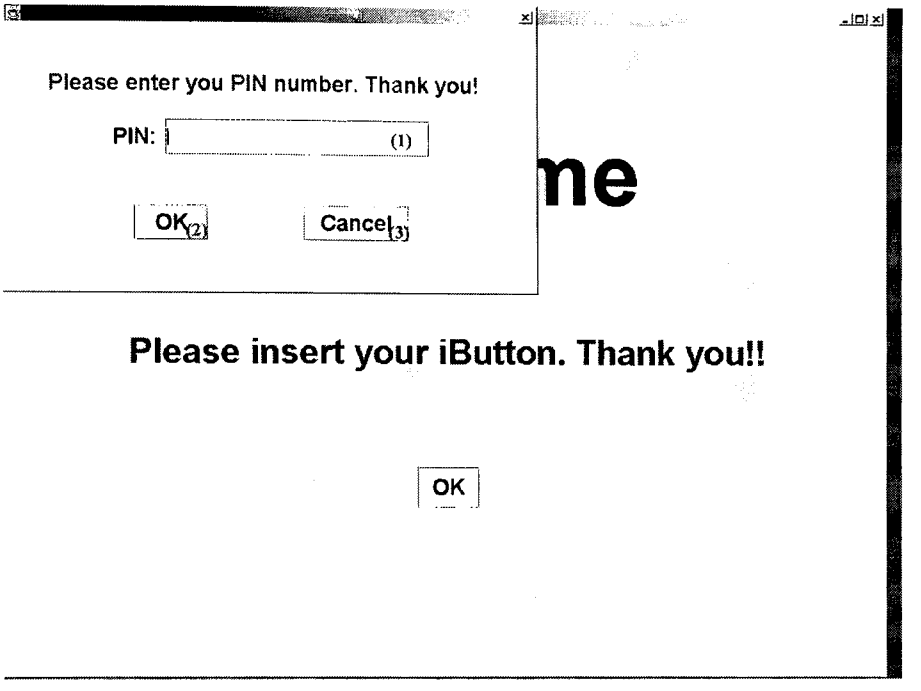


This dialog will pop up when you try to create another account.

(1) shows error messages.

Click (2) OK button to clear dialog.

- **Re-enter pin error dialog**



Re-enter the correct pin in (1).

Click (2) OK button to check pin.

Click (3) Cancel button to abort.

5.4 Java Powered iButton Setting on Solaris 8 and Windows 2000

5.4.1 Hardware Installation

It is easy to install hardware. Connect Blue Dot Receptor to serial adapter first. Then connect adapter to serial port.

5.4.2 Extra Software Installation

See Appendix C for more details.

5.4.3 Loading applet into Java iButton

Use help option in iB-IDE 1.10 for details.

6. Future work

There are still a lot of things we can add on. Below are some directions we can consider to make this project more complete.

- **Use Java iButton to authenticate web page access with SSL:**
  We can generate the key within iButton. It is much safer than generating a key within pc environment.
- **Allowed money to transfer between accounts**
  We can put concurrency topic into the design and make bank simulation more complete.
- **Let Java iButton performs multiple functions:**
  Add more functions to Java iButton. Right now, there is only one applet running on Java iButton. We can add more functions such as access control for pc, Medical/Patient Care Data and etc.
- **Use Dynamic Classloading:**
  Using dynamic classloading, clients can get the class files when they need to. When we have new addon files to distribute, this would be the easiest way to get the job done.
- **Use RMI Activation:**
  In my project, I have to restart client and server when server crashes. Using RMI activation will make this situation better. Activation can make the stubs more persistent and help us to manage remote services.
- **Combined RMI with Jini:**

  With RMI, if any of the components fail, recovery is nontrivial. Data recovery is an important subject in distributed system. Jini can help us to solve this problem.

7. Summary

This project let us explorer the power of Java. It covers:

- Remote Method Invocation (RMI)
- Database (JDBC + DBMS)
- Java 2 security (JCE + JSSE)
- Java Powered iButton

Because we are dealing with money, we have to pay more attention to security. RMI is a good tool to build distributed systems, but it still contains some security problems. In this project, I use JCE and JSSE to make RMI more secure. Also SSL and object encryption can provide a secure environment for data transfer. I think using more than one algorithm to encrypt data can lower the security risks. I choose Java iButton to replace student ID card because Java iButton makes the information and money much safer due to its good design. Thanks to database. In distributed system, it is not allowed to lose any data when server crashes. It must always have a method to recover the data.

After going through the whole design and implement phase, I have more clear ideas on these topics and it also makes me more familiar with the design pattern.

8. Bibliography

References

[1] http://www.javasoft.com/products/jdbc/features.html

[2] http://www.ibutton.com/store/jringfacts.html

[3] http://java.sun.com/products/jdbc/datasheet.html

[4] iButton Homepage, http://www.ibutton.com/index.html

[5] "**An introduction to the Java Ring**",

http://www.javaworld.com/javaworld/jw-04-1998/jw-04-javadev.html

[6] "**iButtons: The first ready-to-buy 2.0 Java Card API devices**",

http://www.javaworld.com/javaworld/jw-04-1998/jw-04-ibuttons_p.html

[7] OpenCard architecture, documents, and source code for this article

http://www.opencard.org/

[8] "**Understanding Java Card 2.0**",

http://www.javaworld.com/javaworld/jw-03-1998/jw-03-javadev_p.html

[9] jGuru: Remote Method Invocation (RMI),

http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html#IntroRMI

[10] **RMI**, http://splash.javasoft.com/docs/books/tutorial/rmi/index.html

[11] http://www.javaworld.com/javaworld/jw-01-2000/jw-01-howto.html

[12] http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmi-objmodel5.html

[13] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides,

"Design Patterns – Elements of Reusable Object-Oriented Software", 1999

[14] Marco Pistoia, Duane Reller, Deepak Gupta, Milind Nagnur, Ashok K. Ramani. **Java 2 Network Security Second Edition**. Prentice Hall, 1999.

[15] Brett Spell. **Professional Java Programming**. Wrox Press Ltd, 2000.

[16] Stephen Asbury, Scott R. Weiner. **Developing Java Enterprise Applications**. John Wiley & Sons Inc, 1999.

[17] Rickard Öberg. **Mastering RMI**. John Wiley & Sons Inc, 2001.

Appendix A. Java Powered iButton Detail

A.1 DS1957B-406 Java-powered iButton, model 96 release 2.2

DS1957B Java™-powered iButton® Features:

- High capacity, 134 kbyte, fast NV SRAM supports multiple independent applications
- Ample memory for over 30 certificates (X.509v3) and hundreds of user names/passwords
- True Time Clock is tamper-resistant so time/date stamps and expiration dates can be trusted
- Physically secure computer chip keeps keys safe and executes 1024-bit public key cryptography for the safe exchange of information
- FIPS 140-1 validated as a secure cryptographic module
- Self-clocked processor prevents tampering with program execution
- Standard cryptographic classes include SHA-1 for secret key digital signatures, RSA public key, DES algorithm
- Math accelerator performs RSA encryption in less than one second using 1024-bit modulus and exponent
- Wear-tested for 1 million insertions and more than 10-year life

The DS1957 Java-powered iButton can be updated for Web-based applications not yet invented. Because its memory contents can be revised after issuance as often as desired, future software can be downloaded from the Internet, adding new services to the iButton.

As Public Key Infrastructure (PKI) emerges in the marketplace, users will want to get rid of the cumbersome user name/password sign-on methodology wherever possible. A much more secure method of logging onto computers is based on a challenge/response algorithm that never reveals the private key stored in a hardware token. This security protocol requires keeping the key private at all costs.

If logically attacked, the firewall of the iButton prevents access to the private keys. If physically attacked, the iButton has a tamper response that erases the private key rather than reveal it in order to preserve confidentiality. These protective measures comply with United States Government specifications for secure e-commerce under the NIST FIPS 140-1 cryptographic module validation program program. In addition to the strong 1024-bit keys for secure log-on, the Java-powered iButton has a true time clock inside its cryptographic boundary for time stamping the e-signatures that are now legal signatures under laws enacted by the U.S. congress and other governments.

The iButton is a safe place to keep the private key and a certificate issued by a trustworthy entity. In this way, the private/public key pair stored in the iButton is associated with the individual. Software development tools for integrating the certificates issued by certificate authorities like Verisign, Entrust Technologies, Baltimore Technologies, and Microsoft are available for free download at www.ibutton.com/pki.html.

Source:

https://store.ibutton.com/cgi-

bin/ncommerce3/ProductDisplay?prrfnbr=89213&prmenbr=776

Appendix B. JCE and JSSE Installation Guide

B.1 JCE Installation Guide

Instructions: To make the JCE 1.2.1 framework an installed extension, move or copy the `jce1_2_1.jar`, `US_export_policy.jar`, and `local_policy.jar` files from the `lib` subdirectory extracted in the previous step to the standard place for the JAR files of an installed extension:

1. <java-home>\lib\ext [Win32]
2. <java-home>/lib/ext [Solaris]

Here <java-home> refers to the directory where the runtime software is installed, which is the top-level directory of the JRE or the jre directory in the Java™ 2 SDK software. For example, if you have J2SDK v 1.2.2 installed on Solaris in a directory named `jdk1.2.2`, you need to install the JAR files in the following directory:

jdk1.2.2/jre/lib/ext

Similarly, if you have Java™ 2 Runtime Environment v 1.2.2 installed on Solaris in a directory named `jre1.2.2`, you need to install the JAR files in the following directory:

jre1.2.2/lib/ext

To also make the "SunJCE" provider an "installed" extension, move or copy `sunjce_provider.jar` to that same directory for installed extensions.

B.2 JSSE Installation Guide

```
------------------------------------------------------------------------
Installation
------------------------------------------------------------------------

JSSE 1.0.2 is supplied as an extension to the Java 2 platform.  JSSE is
implemented via a Java Cryptography Architecture (JCA) security
provider class called "SunJSSE."

Note:

      (Windows and Solaris use different pathname separators, so
      please use the appropriate one ("\", "/") for your
      environment.)

      <java-home> refers to the directory where the Java 2 Runtime
      Environment (JRE) was installed.  The Java 2 SDK (aka JDK)
      contains the JRE, but at a different level in the file
      hierarchy.  For example, if the Java 2 SDK or JRE was installed
      in /home/user1, <java-home>  would be:

            /home/user1/jre1.2.x                [JRE]
            /home/user1/jdk1.2.x/jre            [SDK]

1)  Download JSSE 1.0.2.

      You can save the downloaded file anywhere on your local disk.
      Note that JSSE 1.0.2 requires that you have Java(tm) 2 SDK v
1.2.1
      or greater or Java(tm) 2 Runtime Environment v 1.2.1 or greater
      already installed.

2)  Uncompress and extract the downloaded file.

      This will create a directory named jsse1.0.2, with two
      subdirectories named doc and lib.

3)  Install the JSSE jar files.

      The JSSE lib subdirectory contains the extension files jsse.jar,
      jcert.jar, and jnet.jar.  You can either install these files in
      the JDK/JRE ("installed extension") or bundle them with your
      applet or application ("bundled extension").  If you
      wish to install them as an installed extension, place
      them in the following directory:

            <java-home>/lib/ext

4)  Register the SunJSSE provider.

      JSSE 1.0.2 comes standard with a Cryptographic Service Provider,
      or "provider" for short, named "SunJSSE".  Although the "SunJSSE"
      provider is supplied with every JSSE 1.0.2 installation, it still
      needs to be configured explicitly, either statically or
      dynamically, before its services can be accessed.
```

4a) Static registration of SunJSSE provider.

Add the "SunJSSE" provider to your list of approved providers. This is done statically by editing the security properties file:

```
<java-home>\lib\security\java.security [Win32]
<java-home>/lib/security/java.security [Solaris]
```

One of the types of properties contained in the java.security file is of the following form:

```
security.provider.n=providerClassName
```

This declares a provider, and specifies its preference order "n". The preference order is the order in which providers are searched for requested algorithms (when no specific provider is requested). The order is 1-based; 1 is the most preferred, followed by 2, and so on.

Add the above line to java.security, replacing providerClassName with com.sun.net.ssl.internal.ssl.Provider, and substituting n with the priority that you would like to assign to the "SunJSSE" provider. For example, to add the Sun internal SSL provider to the standard provider shipped with the JRE, your entries would look like:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
```

"SunJSSE" would now be the second preferred provider

Source: http://www.javasoft.com

Appendix C.

C.1 iB-IDE 1.10 Instalation Guide

- **Installation Instructions**

  Run the setup.class with java setup

  Make sure you are using the java executable that comes with your JDK, i.e. it should be located in your [JDK]/bin directory. You may need to explicitly call that JDK. For example: c:\jdk1.2.1\bin\java setup

- **Installation Issues**

  On Windows2000, the install program cannot make any desktop icons or start menu items. To run the program, find a file in the install directory called iB-IDE_1_10.bat and double click it to run. There is also a file called index.html that is an entry point to the help files for the iB-IDE.

  On Linux and Solaris, make sure that you have write permissions to the JDK's jre/lib/ext directory.
  If you see the error "You must have write permissions to the JDK directory", it may also mean that some files in the jre/bin, jre/lib, or jre/lib/ext folders are read-only. Please make these files writeable so the install can provide you with newer versions.

  If the install fails because it cannot find tools.jar, try running the setup class by explicitly defining the path to your JDK's version of java.exe. For example, try: c:\jdk1.2.1\bin\java setup

If you are running Linux, you will need to install Java's CommAPI on your machine. Our install does not have the Linux CommAPI included.

The emulators now support the debugging of the cryptographic functions of the Java Powered iButton. However, you will need to install Java's JCE, or Java Cryptography Extension and include the provided jar file in your classpath when you run (You can also put this file in your [JDK]\jre\lib\ext directory). Download it from www.javasoft.com.

On Solaris and Linux platforms you should specify the entire installation directory. Do not use a relative directory for the target location. For example, instead of installing to the "iB-IDE" directory, use "/usr/home/jqpublic/iB-IDE".

Source: http://www.ibutton.com/iB-IDE/