INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Interactive Internet Banking System

Haiyu Huang

A Major Report

In

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of
Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

June 2002

Abstract

Interactive Internet Banking System

Haiyu Huang

With the popularity of the Internet, client/server software systems have been under tremendous growth. The banking system is one of the most important application domains that are naturally adopting the client/server solution.

Currently available banking systems are based on human-machine (between bank customers and application server) interaction model, which can only accomplish simple bank transactions. *Interactive Internet Banking System* (IIBS) is an interactive client/server application, which allows bank customers to access traditional bank services and interact with bank representatives through the Internet. The system is implemented using distributed objects paradigm with CORBA/Java as client/server middleware. This report gives a detailed description of the background, design, and implementation of the system.

Acknowledgements

I would like to take this opportunity to express my cordial gratitude to my supervisor Dr. Peter Grogono for his continuous guidance and insightful advice during the preparation of this major report. Like many others, I consider myself blessed to be under his supervision.

I am thankful to the staff in the Department of Computer Science, especially Mr. Stan Swiercz who helped me set up computer system necessary for implementing this project, and the Graduate Secretary Ms. Halina Monkiewicz for her kindly and prompt administrative services, all which made my graduate studies at Concordia University a pleasant experience. A special thanks also goes to my friend Eddie Chan for his assistance and encouragement both that are indispensable to complete this report.

Finally I would like to dedicate this work to my mother, as her expectations and love always gave me the will and strength to pursue higher goals.

CONTENTS

List of Figure

Chapter 1

1. Introduction

With the arrival of the Internet, distributed computing systems have become more and more important. The client/server application model is the state of the market and state of the practice for distributed computing at the time of this writing. Initial versions of the client/server model utilized the remote procedure call paradigm that extends the scope of a local procedure call. Presently, the client/server model is increasingly utilizing the distributed objects paradigm that extends the scope of local object paradigm (i.e., the application processes at different sites are viewed as distributed objects). Distributed objects are objects that can be dispersed across the network and can be accessed by users and applications across the network. Conceptually, enterprise-wide applications are decomposed into objects that can reside on the network. An object on one machine can send messages to objects on other machines; thus, we can view the entire network as a collection of objects. Distributed objects present a very powerful technology and paradigm that have the potential of addressing many problems facing the IT community today (i.e., reuse, portability, and interoperability) [5]. This is because applications can be constructed by using reusable components that encapsulate many internal details and can inter-operate across multiple networks and platforms.

Support of distributed object applications requires special purpose middleware that allows remotely located objects to communicate with each other. Examples of middleware for distributed objects include Object Management Group's (OMG's) CORBA (Common

Object Request Broker Architecture) and Microsoft's ActiveX/COM/DCOM. Both of these middleware packages use the distributed object model based on the object request broker (ORB) that receives an object invocation and delivers the message to an appropriate remote object.

Interactive Internet Banking System (IIBS), the project described in this report, is an Internet based client/server application that is implemented in the distributed objects paradigm by using CORBA/Java technology.

1.1. Problem Definitions

Interactive Internet Banking System (IIBS) is an interactive client/server application, which allows bank customers to access bank services and to interact with bank representatives through the Internet. The target users of the system will be the bank corporation and its customers. IIBS has two parts: the traditional online banking service and novel **online consultation service**.

- *Online banking service.* A bank customer can login to the system and perform basic bank transactions; for example, check account balance, view recent transactions, pay bills, and transfer money between accounts. In this case, the customer is interacting with bank's server machine.
- *Online consultation service.* The bank customers directly interact with bank representatives (bank employees). This involves two kinds of users, the bank customer and bank employee. After successfully logging in, the customer can access complete information about available banking customer service from the client side user interface. This information includes the total number of customers who are demanding consulting service and the list of representatives who are online. If the customer selects a particular

representative, the related information about this representative should be displayed. For example, a photo of this representative, number of customers waiting for this representative etc. can be displayed. Customers can also connect to and interact with the desired representative. Only one customer is allowed to connect to one representative at any time. If he wants to change to another representative, he must disconnect from the current representative and then select a new representative. On the other hand, bank employees can login to the system and interact with many customers. The waiting list and the current customer profile will be displayed on the representative's user interface. The representative can communicate with current customer, send documents to the customer, perform bank transactions related to the current customer based on his requests, disconnect from the current customer, and end the current session.

The system is responsible for connecting a particular customer to a particular representative and managing each representative's customer service waiting list. At any time, the customer or bank representative can exit from the system. Before exiting, the system will prompt users if a customer is connected to a representative and waiting for service or the representative has customers waiting for a particular service. The system is also responsible for updating related information whenever a change occurs. For example, if a customer exits from the system, the server will update the representative's waiting list and all other online customer's service information. Each individual's user interface will be updated accordingly. The customer/representative direct interactions make complex transactions possible. For example, customers can inquire on or negotiate on interest rates with bank representatives before purchasing a term deposit or guaranteed investment certificate (GIC). In this case, the customer is interacting with the representative through a bank's server machine: this is the novel part of the system.

1.2. Comparison to Existing Systems

The selection of this project is based on a comprehensive study and comparison to current existing systems. Basically there are two existing systems that customers use to do remote banking. These consist of Web based banking systems and telephone banking systems.

1.2.1. Web Based Banking System

Interactive Internet Banking System (IIBS) makes two improvements to current web based banking systems.

- *Direct customer/bank employee interaction:* Current available banking systems (e.g., CIBC PC banking) are based on a human-machine interaction model (bank customer and application server) that can do only simple bank transactions. IIBS adds the capability of server side interactive customer service, as a result, this allows customers to interact with a bank employee and to perform more complicated banking tasks without physically going to the bank.
- *Use of new technology:* Most of current existing systems are implemented using HTTP/CGI, which is based on a stateless connection [3]. This is often slow and not suitable for direct customer/bank employee interaction. The IIBS will use new Internet client/server middleware CORBA/JDBC as an implementation choice that improves performance.

1.2.2. Telephone System

The customer can do banking by calling a customer representative using the telephone. In comparison to this approach, IIBS has the following advantages:

- *Provides better service.* If a customer calls a banking representative and the line is busy, her only choice is to simply wait on the telephone line without being provided with any further information, and must simply wait as part of a queue. She cannot do anything else. With IIBS, the customer will get a "good picture" of available banking customer service. A list of available representatives will be shown on the user's interface where each representative will have waiting list information. The user can select a particular employee by clicking the name or simply clicking the connect button, which will then connect the customer to a representative or place the customer in the shortest waiting list according to load balance. IIBS will then clearly indicate to the user the estimated waiting time (by computing average waiting time per customer in past time and the number of customer in current waiting list). With this information, the user has the option to do something else, e.g., continue to surfing web or working on other documents. When a bank representative is available, a beep message will remind the customer and the user interface will be updated accordingly. For instance, the representative profile will be shown on her user interface. The same advantages will also be applied to the bank employee on the server side.
- *Multimedia capability:* Whereas a telephone system can transfer only voice messages, IIBS can take full advantage of a computer's multimedia capabilities. For example, a bank representative can send the customer a GIC investment certificate confirmation document with the representative's signature on it. (Due to Concordia University's Computer Science lab's limitations, this system will not implement any voice media functionality).
- *World Wide Web availability with low cost:* Since IIBS is intended to be implemented as a web based client/server application, individuals everywhere can perform banking

transactions world wide through the Internet without being charged international or long distance call rates.

This application model can also be naturally and easily extended to other domains. For instance, an *Interactive Internet Travel Agent* can allow travel agents to communicate with their remote worldwide customers without the need of having their employees being occupied with local customers in the office.

1. 3 Tools Used

IIBS is developed primarily in a Windows NT environment, and the following tools were used for its development.

- *Microsoft Visual J++ 6.0*: used for developing, compiling, debugging and running Java programs.
- *Visibroker for Java 3.3/4.0*: used for CORBA implementation support [7][11].
- *MySQL3.23.11:* used as database management to handle data storage, retrieval, management and recovery of all persistent data [6].
- *JDBC:* served as connectivity between database and server side CORBA object [8].
- *Internet explorer 4.0 or higher:* served as web browser to load client applet.

This report is structured as follows: Chapter 2 describes the technology used to implement the system. Chapter 3 presents the System Requirements and Analysis. System designs are described in Chapter 4. Chapter 5 discusses in detail the system implementation of using CORBA/Java/JDBC technologies and Chapter 6 concludes the report with a summary and comments on future enhancements. Appendix A details a user manual for using the system

and sample running sessions. Appendix B lists the Java source code for system implementation.

Chapter 2

2. Technology Overview

In this chapter, we provide an overview of the major client/server programming techniques that are used in the project. First we explain why this project uses CORBA as the client/server middleware and how CORBA and Java can be used together. Following this explanation, a panoramic review of the CORBA architecture and JDBC is presented.

2.1 Why CORBA and Java?

The Internet was built on an open client/server standards. To be open, the technology must run across multiple networks, operating systems, languages, and computer platforms. The Internet started out as a giant and very trendy unidirectional medium for publishing and broadcasting static electronic documents. The predominant HTTP client/server middleware was primarily designed to serve documents. Consequently, the Internet became a giant URL-based file server. In late 1995, the Internet evolved into a more interactive medium with the introduction of 3-tier client/server and CGI style [3]. The *Common Gateway Interface* (CGI) protocol allows Web server route the content of HTML forms to back-end server applications. HTTP/CGI is now the dominant Internet technology.

Unfortunately, HTTP/CGI is a slow, cumbersome, and stateless protocol, and it is not suitable for writing modern client/server applications. With HTTP/CGI protocol, the client

establishes a connection to the remote server. Next it issues a request, the server first processes the request (it may pass the method request and its parameters to the back-end program using CGI protocol, then the back-end program executes the request and returns results in HTML format to the server), returns a response, and then closes the connection. HTTP/CGI typically sets up a new connection for each request, and the client must wait for a response before sending out a new request. In addition, CGI is slow as it launches a new process to service each incoming client request.

To fully benefit from object-to-object interactions, the Internet needs a distributed object infrastructure, such as CORBA/ORB [1][2]. CORBA/ORB bypasses the CGI bottleneck and replaces it with ORB based object-to-object interactions. In the client/server interaction, clients (applets or beans) directly invoke methods on the server, as such, the client passes the parameters directly using precompiled stubs or it generates them on the fly using CORBA's dynamic invocation services. In either case, the server receives the call directly via a precompiled skeleton. With CORBA, we can invoke any IDL-defined method on the server. Furthermore, we can pass any typed parameter instead of just strings, which means that there is very little client/server overhead, compared with HTTP/CGI. With CGI, we must start a new instance of a program every time an applet invokes a method on a server, whereas with CORBA, we do not. CGI does not maintain a state between client invocations but CORBA does. With CORBA, Java clients and applets can invoke a wide variety of IDL-defined operations on the server. In contrast, HTTP clients are restricted to a limited set of operations. Server side applications are regular CORBA objects. Consequently, they are available on a permanent basis. There is no need to go through the overhead of spawning a CGI script for every invocation. This is especially important for building an interactive client/server application.

In a sense, the Java infrastructure starts where CORBA ends. CORBA provides a distributed object infrastructure that lets applications extend their reach across networks, languages, component boundaries, and operating systems. Java provides a portable object infrastructure that works on every major operating system. CORBA deals with inter network transparency, while Java deals with implementation transparency [3]. Java allows CORBA objects to run on everything from mainframes and network computers to cellular phones. Java simplifies code distribution in large CORBA systems. Its byte code provides a simpler and new way to develop, manage, and deploy the client/server application. Java is almost the ideal language for writing client and server CORBA objects. Its built-in multithreading, garbage collection, and error management make it easier to write robust network objects [4]. So CORBA and Java are two object infrastructures that complement each other very well. Actually, CORBA has now become part of the Java Core.

2.2 CORBA

The *Common Object Request Broker Architecture* (CORBA) is the most important distributed object framework proposed by a consortium called Object Management Group (OMG) [12]. The core of the CORBA is the *Object Request Broker* (ORB) that acts as the software bus over which objects transparently interact with other objects located locally or remotely. CORBA uses objects as a unifying entity for bringing existing applications to the software bus and thereby provides a solid foundation for client-sever based software system. CORBA achieved this by its fundamental principle, which is the separation of a service from its implementation. A CORBA object is represented to the outside world by an interface with a set of methods. Interface specifications in CORBA are written in *Interface Definition language*

(IDL). A specification defines a server's boundaries and its contractual interfaces with potential clients.

A particular instance of a CORBA object is identified by an object reference. The client of a CORBA object acquires its object reference and uses it as a handle to make method calls, as the object is located in the client's address space. The ORB is responsible for all the mechanisms required to find the object's implementation, preparing it to receive the request, communicating the request to it, and carrying the reply (if any) back to the client. In the following sections, we briefly introduce the CORBA architecture, its features, and services

2.2.1. OMG CORBA Architecture

In the fall of 1990, the OMG first published the *Object Management Architecture Guide* (OMG Guide) [12]. It was revised in September 1992. **Figure 2.1** shows the four main components of the architecture, **Figure 2.2** shows the layers relationships of the four components.

- *Object Request Broker (ORB)* defines the CORBA object bus.
- *CORBA services* define the system level object frameworks that extend the bus;
- *CORBA facilities* define horizontal and vertical application frameworks that are used directly by business objects;
- *Application Objects* are the business objects and applications. They are the ultimate consumers of the CORBA infrastructure.

The following sections provide a top-level view of the four components that make up the CORBA infrastructure.

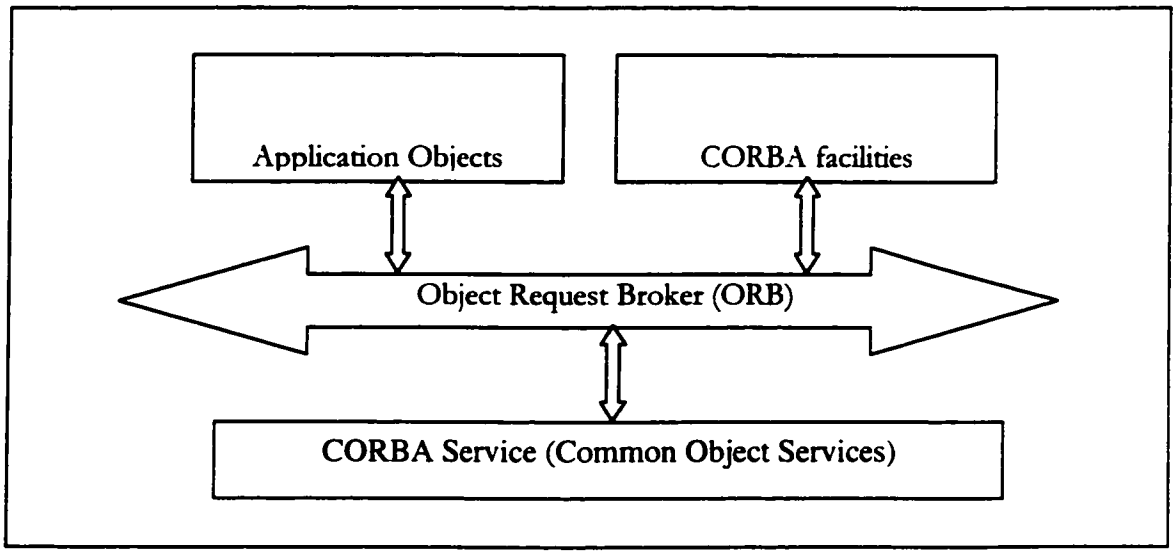

Figure 2.1 OMG CORBA Architecture

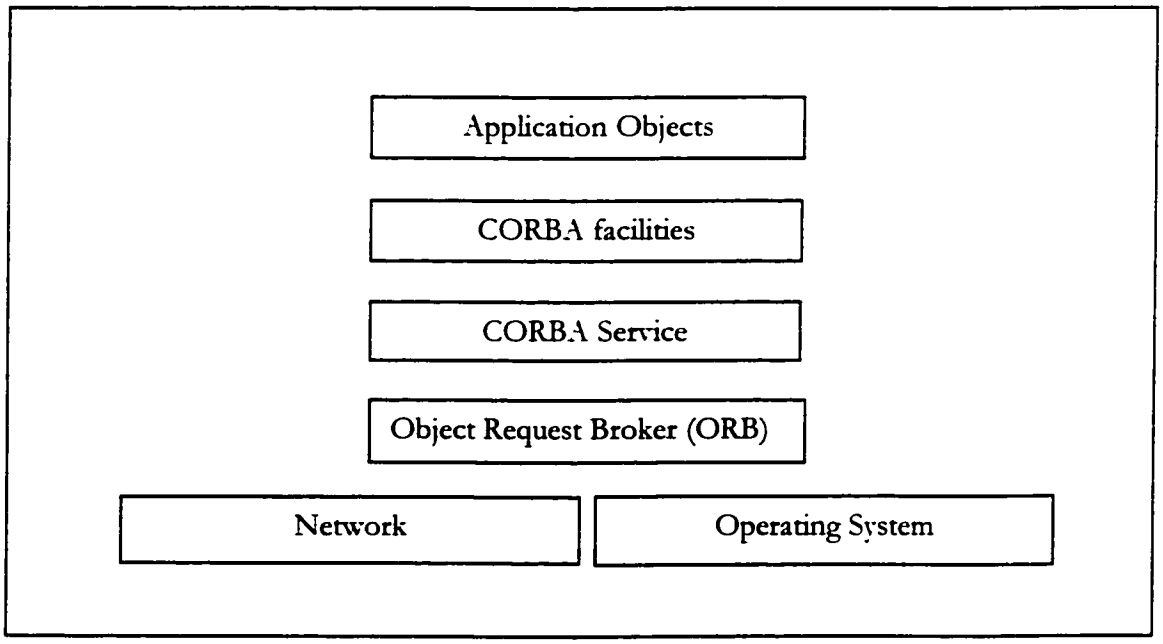

Figure 2.2 Layers Relationships of the Four ORB Components

2.2.2. Object Request Broker (ORB)

Object Request Broker (ORB) is the middleware that establishes the client/server relationship between objects. Using an ORB, a client object can transparently invoke a method on a sever object, that can be on the same machine or across network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client dose not have to be aware of the location of the object, the programming language, the operating system, or any other system aspects that are not part of an object's interface. The functions of various elements in the ORB structure are described below:

On Client Side:

- *The client IDL Stubs:* Provides the static interfaces to object services. Those precompiled stubs define how clients invoke corresponding services on the servers. From a client's perspective, the stub acts like a local call, it is a local proxy for the remote server object. The services are defined using IDL, and both client and sever stubs are generated by the IDL compiler. A client must have an IDL for each interface it uses on the server. The stub includes code to perform marshaling. This means it encodes and decodes the operation and its parameters into flattened message formats that it can send to the server
- *Dynamic invocation interface (DII):* Provides the mechanism to discover methods to be invoked at run time. CORBA defines standard API for looking up the metadata that defines the server interface, generating the parameters, issuing the remote call, and getting back the results.
- *Interface repository:* Makes it possible to obtain and modify the descriptions of all registered component interfaces, the methods they support, and the parameters they require.

Essentially the *Interface repository* is a run time distributed database that contains machine-readable versions of the IDL-defined interfaces.

- *ORB interface:* Consists of a few APIs to local services that may be of interest to an application. For example, CORBA provides APIs to convert an object reference to a string and vice versa.

On Server side:

- *Server IDL Stubs:* Provide static interfaces to each service exported by the server. These stubs, like the ones on the client, are created using an IDL compiler.
- *Dynamic skeleton interface (DSI):* Provides a run time binding mechanism for servers that need to handle incoming method calls for components that do not have IDL based compiled skeletons (or stubs).
- *Object adapter:* Sits on top of the ORB's core communication services and accepts requests for service on behalf of the server's objects. It provides the run-time environment for instantiating server objects, passing requests to them, and assigning them object IDs. This *object adapter* also registers the classes it supports and their run-time instance with the *Implementation Repository*. Every CORBA ORB must support at least the minimal standard object adapter called *Basic Object Adapter* (BOA).
- *Implementation Repository:* Provides a run-time repository of information about the classes that a server supports and the objects that are instantiated, and their IDs. It also serves as a common place to store additional information associated with the implementation of ORBs.
- *ORB interface:* Consists of a few APIs to local services that are identical to those provided on the client side.

2.2.3. CORBA Services

CORBA services are collections of system-level services packaged with IDL-specified interfaces. These services augment and complement the functionality of the ORB. They are used to create a component, name it, and introduce it into the environment. OMG has published standards for fifteen object services [12]:

- *Life Cycle Service:* Defines operations for creating, copying, moving, and deleting components on the bus.
- *Persistence Service:* Provides a single interface for storing components persistently on a variety of storage servers, including *Object Databases* (ODBMSs), *Relational Databases* (RDBMSs), and simple files.
- *Naming Service:* Allows components on the bus to locate other components by name; it also supports federated naming contexts. This service also allows objects to be bound to existing network directories or naming contexts, including ISO's X.500, OSF's DCE, Sun's NIS+, Novell's NDS, and the Internet's LDAP.
- *Event Service:* Allows components on the bus to dynamically register or unregister their interest in specific events. The service defines a well-known object called an *event channel* that collects and distributes events among components that know nothing of each other.
- *Concurrency Control Service:* Provides a lock manager that can obtain locks on behalf of either transactions or threads.
- *Transaction Service:* Provides two-phase commit coordination among recoverable components using either flat or nested transactions.
- *Relationship Service:* Provides a way to create dynamic associations (or links) between components that know nothing of each other. It also provides mechanisms for

traversing the links that group these components. This service also can be used to enforce referential integrity constraints, track containment relationships, and establish any type of linkage among components.

- *Externalization Service:* Provides a standard way for getting data into and out of a component using a stream-like mechanism.
- *Query Service:* Provides query operations for objects. It is a superset of SQL. It is based on the upcoming SQL3 specification and the Object Database Management Group's (ODMG) *Object Query Language* (OQL).
- *Licensing Service:* Provides operations for metering the use of components to ensure fair compensation for their use. The service supports any model of usage control at any point in a component's life cycle. It supports charging per session, per node, per instance creation, and per site.
- *Properties Service:* Provides operations that let you associate named values (or properties) with any component. Using this service, you can dynamically associate properties with a component's state-for example, a title or a date.
- *Time Service:* Provides interfaces for synchronizing time in a distributed object environment. It also provides operations for defining and managing time-triggered events.
- *Security Service:* Provides a complete framework for distributed object security. It supports authentication, access control lists, confidentiality, and non-repudiation. It also manages the delegation of credentials between objects.
- *Trade Service:* Provides a "Yellow Pages" for objects; it allows objects to publicize their services and bid for jobs.

- *Collection Service:* Provides CORBA interfaces to generically create and manipulate the most common collections.

2.2.4. CORBA Facilities

CORBA facilities, formerly known as *Common Facilities*, are collections of IDL-defined frameworks that provide services of direct use to application objects. *CORBA facilities* are currently divided into two broad categories: horizontal and vertical.

The horizontal facilities are application-generic and have the highest potential for reuse. They sit between the *CORBA services* and *Application Objects* (described below). Unlike the vertical facilities, these facilities are potentially useful across business domains. There are only four horizontal facilities:

- The Printing Facility,
- The Secure Time Facility,
- The Internationalization Facility,
- The Mobile Agent Facility.

The vertical facilities are more application specific and tend to recur primarily within the application niche for which they are designed.

Partially because of their added complexity and because they build on *CORBA Services*, the *CORBA facilities* are not completed yet. Building those facilities is a never-ending project. This work will continue until CORBA defines IDL interfaces for every distributed service as we know of today, as well as ones that are yet to be invented. When this happens, CORBA will provide IDL-interfaces for virtually every networked service.

2.2.5. Application Objects

Application Objects are at the topmost part of the OMA hierarchy. Since they are typically customized for an individual application and do not need standardization, this category identifies objects that are not affected by OMG standardization efforts.

2.3 JDBC

The *Java Database Connectivity* (JDBC) is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases. JDBC consists of two parts: The high level JDBC API and multiple low level drivers for connecting to different databases. Rich Cattell provides more extensive and in-depth coverage about JDBC [8].

2.3.1 JDBC API

The JDBC API is a set of java classes and interfaces for accessing virtually any kind of tabular data. It allows a java application to establish database connections, perform SQL-based database query, update transactions, and process data result sets and database metadata etc. Starting from JDK 1.1, JDBC API has become part of Java JDK core; it is included in the *java.sql* package. For a complete description of JDBC classes and interfaces, refer to JDK 1.1 documentation [10].

2.3.2 JDBC Database Drivers

All JDBC API calls are passed to a JDBC driver manager. The driver manager in turn passes the request to the JDBC database driver that can handle the request. JDBC Database drivers

are DBMS specific but they all implement the core *Java.sql* interfaces and classes, this is the mechanism under which the JDBC is capable of accessing almost every SQL database [8].

Chapter 3

3. System Requirements Analysis

In this chapter, we first provide an analysis of the problems of *Interactive Internet Banking System* (IIBS) using use case diagrams. This is followed by a brief description of functional and non-functional requirements of the system.

3.1 Use Case Analysis

Figure 3.1 shows the use case diagram for the IIBS.

There are three actors in the system:

- *The customers:* Function as bank clients to perform basic banking transactions and possible online consultation with bank representatives.
- *The representatives:* Function as bank employees to perform online consultation with bank customers and perform necessary bank transactions on behalf of the customer currently being served.
- *The System administrator:* Functions as the bank Database administrator to setup and maintain bank database (e.g., add/remove a bank employee from database).

The different use cases carried out by these actors are described below:

1. **Use Case: Login:**

- *Purpose:* Allows valid bank customer and representative to login to the system.

Actors: Bank customer and bank representative.

Base Scenario:

1) *Customer:* The bank customer gives account number and password, and clicks the login button. A message is sent to the server for verification and authentication. If both account and password are valid, the main user interfaces are displayed; otherwise an access denied message is shown.
2) *Representative:* The bank representative gives employee ID and password, and clicks the login button. A message is sent to the server for verification and authentication. If both employee ID and password are valid, the main user interfaces are displayed; otherwise an authentication failure dialog box is shown.

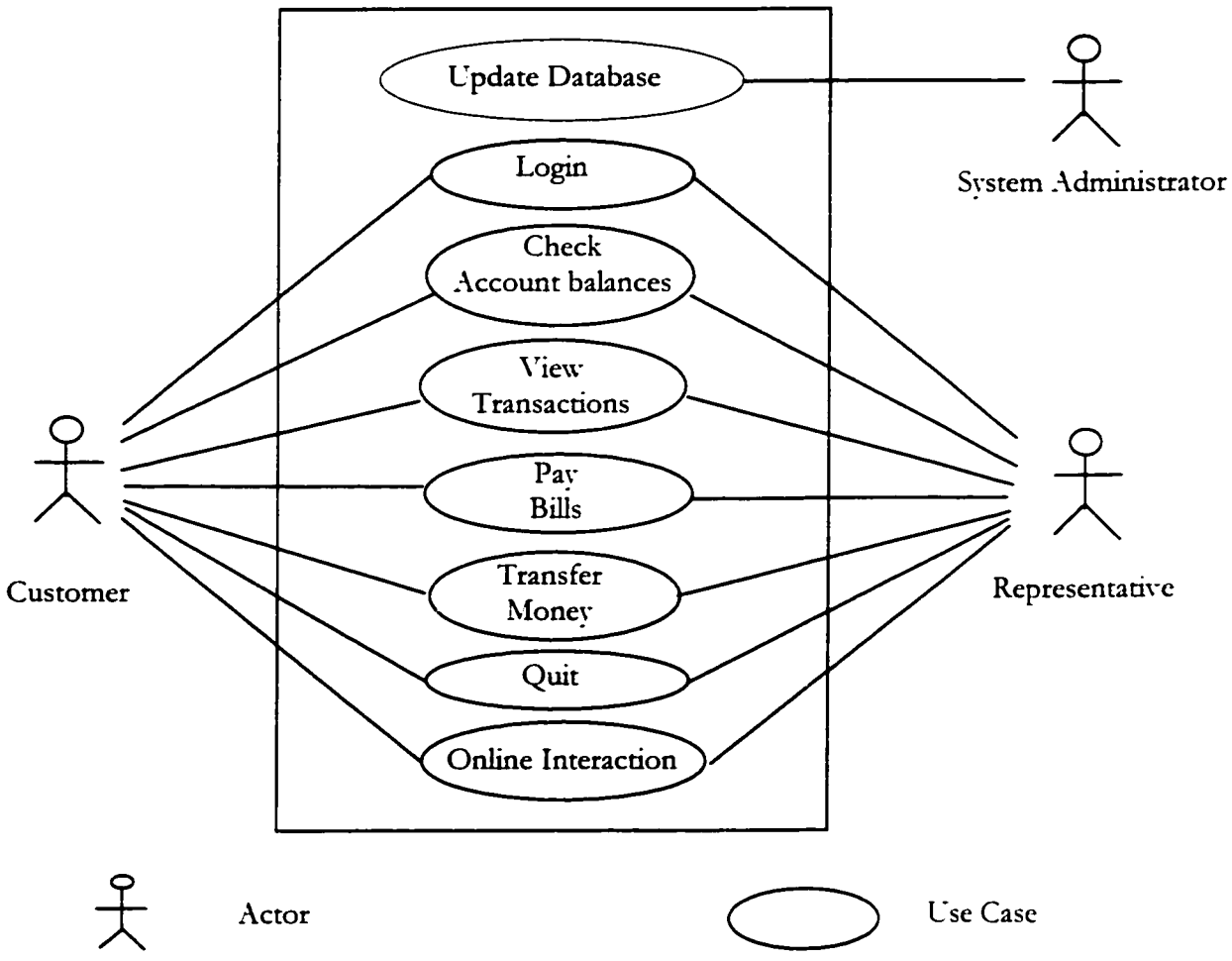

Figure 3.1 User Case Diagram

2. **Use Case: Check account balance:**

- *Purpose:* Allows the customer to check her own account balance or the representative to check accounts balance of the current customer.
- *Actors:* Bank customer and bank representative.
- *Base scenario:*
  1) *Customer:* A customer clicks the account balance button, a message is sent to server, where it then checks the account information of this customer and sends the account balance back to the customer. If the particular customer has multiple accounts, balances for all accounts will be shown. The customer can then select all available accounts from the account list to view its balance.
  2) *Representative:* A bank representative selects the account balance menu, account balance of the current customer will be displayed.

3. **Use Case: View transactions:**

- *Purpose:* Allows a customer to view all transactions of his own accounts or a representative to view all transactions on accounts of the current customer.
- *Actors:* Bank customer and bank representative
- *Base scenario:*
  1) *Customer:* The customer clicks the view transactions button, a message is sent to the server where the server checks the account information of this particular customer and sends account transactions to the user. If the customer has multiple accounts, then transactions for all accounts will be shown. The customer can select all available accounts from the account list to view all their transactions.

2) *Representative:* A bank representative selects view transaction menu, the account transactions of the current customer will be displayed.

4. Use Case: Pay bills

- *Purpose:* Allows a customer to pay bills or a representative to pay bills on accounts of the current customer.
- *Actors:* Bank customer and bank representative.
- *Base scenario:*
  1) *Customer:* The customer clicks the pay bills button, a message is sent to the server where the server checks all the registered bill payment services for this customer, and sends all bill accounts information to the customer (including the amount due, balance forward etc). Then the customer can pay all the bills with a specified amount of money. The customer can select the account from which to pay the bills.
  2) *Representative:* The bank representative clicks the pay bill menu and pays bill on behalf of the current customer.

5. Use Case: Transfer money

- *Purpose:* Allows a customer to transfer money between different accounts or a representative to transfer money between different accounts of the current customer.
- *Actors:* Bank customer and bank representative
- *Base scenario:*
  1) *Customer:* The customer clicks the transfer money button, message is sent to server where the server checks the account information of this particular customer and sends this information to the user. If the particular customer has multiple accounts, he/she can transfer money between these accounts.

2) *Representative:* The bank representative transfers money between accounts of current customer based on the customer's request.

6. Use Case: Quit

- *Purpose:* Allows a customer/representative to exit from the system.
- *Actors:* Bank customer and bank representative
- *Base scenario:*

1) *Customer:* The customer clicks the exit button to quit from the system. Prior to exiting, an appropriate message prompt will be provided by the system to confirm the action if the customer is connected to a representative and is waiting for service.
2) *Representative:* The representative exits from the system. A confirmation dialog box will be provided by the system to confirm the action if the representative has customers waiting for services.

7. Use Case: Online interaction

- *Purpose:* Allows a bank customer to interact with a representative directly.
- *Actors:* Bank customer and bank representative
- *Base scenario:* The customer clicks the customer service button, a message is sent to server where the server will send back the information about all online representatives and customers, such as the total number of customers who are demanding consulting service, the list of representatives who are online, etc. If the customer selects a particular representative, the related information should be displayed, e.g., a photo of this representative, number of customers waiting for this representative, etc. Customers have the choice to connect to the desired representative to interactive with. Customers can talk to the representative and ask her to perform complex transactions such as buying term deposits or GICs. Once representatives login to the system, representatives will be

able to view their waiting list and customer profiles. The representative can communicate with her current customer, send documents to them, perform bank transactions, disconnect from the current customer, end the current session, or move to the next customer in her waiting list.

3.2 Functional Requirements

Based on the above use case analyses, the IIBS should meet the following functional requirements.

- *Login:* Both the customer and representative should be able to login to the system with a valid user name and password. If the login process is successful, the window based main user interface will appear, otherwise a message indicating access denied will be displayed. Users can login as many times as they wish until a successful login occurs. But multiple logins are not allowed for both customers and representatives. If the customer/representative tries to login to the system before exiting from a previous session, the system will prompt the user with the message: "You are already logged in". The customers/representatives will have to quit their current session before attempting to login again.
- *View account balance:* The customer should be able to view his/her own accounts' balance. The system should be able to recognize the customer and display account information related to this particular customer. For example, some customers only have one checking account and others may have both checking and saving accounts. If the customer has multiple accounts, he should be able to view the balances of all accounts. The customer

should only be able to see his account information and not other customer's accounts. Bank representatives can view the account balances of their current customer.

- *View transactions:* The customer should be able to view her own accounts' transactions. For simplicity, all transactions for a particular account will be shown. For each transaction, the credit/debit, amount of money, transaction date, and brief description should be presented. In other words, some customers only have one checking account while others may have both checking and saving accounts. If the customer has multiple accounts, she should be able to view all transactions related to her accounts only and not view any other customers. The bank representative can view the account transactions of their currently serving customers.
- *Pay bills:* The customer should be able to pay registered bills from selected bank account. The system should display the bill information for each registered bill payment, including bill account number, balance forward, and amount due. The user should be able to enter the amount to pay and select the bank account from which to withdraw money to pay the bill. The system should also default the amount due as the payable amount, but also allow the user to change this amount as well. They can process this payment by clicking pay button. If the user overpays any bill, then she should be credited for this amount (will have negative bill account forward balance). If the customer's selected bank account does not have sufficient funds, a proper message should be displayed, indicating that the user has to either reduce the amount to pay or select another account which has a sufficient balance. A basic assumption for bill payments is that registered bill companies have access to customer's respective bill accounts and can update the bill account data regularly. For this project, customer's bill accounts are initialized when the bank database

is loaded. Bank representatives should be able to perform bill payments on behalf of the current customer.

- *Transfer money:* The customer should be able to transfer money between his own accounts. To submit a transfer request, the customer has to enter the amount to transfer and select from and to where the money will be transferred. As a result, the balance will deduct from the transfer-from account and increase the transfer-to account.
- *Quit:* A customer should be able to quit from the system at any time. If the customer is connected to a representative and is waiting for some service or the representative has customers waiting for service, the system will prompt the user before exiting. Otherwise the user can exit from the system immediately. The system should update related information whenever a user exits from system. For example, if a customer is exiting the system, the server will update the related representative's waiting list (the next customer in the waiting list should be moved forward in the list), and all other online customer's service information should be updated (e.g., number of customer demanding online consultation should be reduced by one). A quit message should be broadcast to all other online customers and representatives.
- *Customer/representative online interaction:* This is the main focus of the project. The system should provide the ability of allowing bank customers and representatives to talk to and communicate with each other and fulfill complex transactions. The customer should be able to view the profile of all online representatives. Each profile should include the photo, name, branch ID, telephone, and number of clients waiting for this representative. The customer should be able to connect to their favorite representative and communicate with them. If the representative has a waiting list, then the customer should be told the order in which he is in the waiting list, and this information should be

updated whenever applicable (e.g. a customer in the waiting list quits from the system). The customer should be informed whenever the service is available, whereby at this time, the customer can send the representative a message and receive a response immediately. Here, an interactive conversation record will be displayed on both customer and representative's user interface. A customer can also receive documents (in HTML/JPEG/GIF format) from the representative. While a customer is waiting for a representative in the waiting list, the customer can still perform many other basic banking services from the same system such as view transactions and account balances. The representative should be able to view the customer waiting list and customer profile of currently serving customer, including name, account number, address and telephone. The representative is not allowed to alter the order of the waiting list, i.e., the waiting list is processed in the order as they are entered. But at any time, the representative can disconnect the currently serving customer and start to serve the next one. When a customer is disconnected, he will be removed from the waiting list. The representative can send messages and receive responses from the current customer, send documents, or perform bank transactions on behalf of the customer according to the customer's request. All transactions modified should automatically be applied to the accounts of currently serving customer.

3.3 Non-functional Requirements

Non-functional requirements define the system properties such as its usability, reliability and constraints. IIBS should meet the following non-functional requirements.

- *Usability:* The system should be a window based application with a friendly and easy-to-use graphical user interface. The user interface should be clear, simple, and

understandable. Users with minimum computer literacy should be able to directly use the system efficiently without any special training.

- *Portability:* The system should be able to be setup and run on any platform as long as the server machine has Java runtime environments and CORBA supporting software (VisiBroker 3.3 for Java or higher), and the client machine has Java runtime environments and Java enabled web browser.
- *Efficiency:* The system should response to user's requests without any noticeable delay in all use case scenarios. This is achieved through CORBA/Java technology. The only exception to a quick response may occur in the login process since this process requires more time for the client to locate the server object and for it to register a call back to the client side.
- *Robustness and reliability:* The system should not crash or fail during the operation of any typical use case scenario. In the event of an error or failure, the system should provide the user with meaningful feedback and allow them to recover gracefully.
- *Security and safety:* To ensure the security, safety, and integrity of IIBS, identification facilities and processes should be implemented in order to have different levels of privileges to access the system. This will ensure that unauthorized access to the system will be prevented. The private information of each individual stored in the system should be secure and private, that is, a customer has permission and access to only her own information and the bank representative can access the information of all registered customers. The system administrator can access all information stored within the bank database. A user ID and password will be used to identify the user and enforce the appropriate access privileges.

Chapter 4

4. System Design

In this chapter, we first provide an overall architecture design of the *Interactive Internet Banking System* (IIBS), which includes the description of all classes that are used to build the system. This is followed by presenting Class Diagrams using UML notation. Finally we give a detailed description of the bank database that is used in the system.

4. 1 Architecture Design

The IIBS is a full-fledged 3-tier client/server CORBA/JDBC application. The overall architecture of the system is illustrated in **Figure 4.1**. From the view of a client/server perspective, the system contains three major components:

- Tier-1: Client.
- Tier-2: Server.
- Tier-3: Database Server.

The following sections will describe each of the components in more details.

4.1.1. Tier-1: Client

Tier-1 Client consists of two client side applications: *Client Side Application for Customer* (CSAC) and *Client Side Application for Representative* (CSAR). The CSAC refers to the client side application used by bank customers. It will be implemented as a Java applet for the purpose

of worldwide available thought web browsers (It also can be compiled and run as standalone application for testing and debugging purpose). It will be automatically downloaded by a web browser when the user opens the specific URL. The CSAR refers to the client side application used by bank representatives. It will be implemented as a standalone application which can overcome the limitation imposed on Java applet such that it can not read/write on a local machine, but this will require pre-installation of the application. Since CSAR is used internally by authorized bank employees, the installation should not have any problem.

4.1.1.1. Client Side Application for Customer (CSAC)

The CSAC allows the bank customer to access the basic bank services and interact with the bank representatives. It achieves its goals through the following classes:

- *Class IBClient*: This class will implement standard Java applet interfaces and function as the main program of CSAC. It provides the following six functions:
  1) Create and initialize UI widgets for application user interface.
  2) Initialize the first page, which is login page for customer.
  3) Initialize CORBA.
  4) Locate server side CORBA object Bank Manager.
  5) Create client side callback object.
  6) Implement client quit logic (e.g., tell the server to remove it from its register list).
- *IBClientCorbaImpl:* An instance of this class is the client side CORBA object representing a bank customer. This is a callback object created on the client side application, which will be registered with the server when a customer logins to the system. Afterwards, the server can remotely invoke callback methods defined in this callback object. When a customer requests to connect to a representative, the server will pass the callback object

reference to the *Client Side Application for Representative* (CSAR), so the representative can directly communicate with the customer as long as it obtains the object reference. This is how the client/representative interaction is implemented

- *Class IBClientLogin*: This class will implement the login interface and login logic. It has the following functions:
  1) Create and initialize UI widgets for login interface.
  2) Register client callback object reference with the server.
  3) Load the client main menu if the login is successful, otherwise issue an "Access Denied" warning message.
- *Class IBClientMainMenu*: This class will implement the main menu user interface. It provides the following functions:
  1) Create and initialize UI widgets.
  2) Load corresponding windows and screens according to the user selection.
- *Class IBClientServiceInfo*: This class will implement the user interface and logic of customer service information. The customer service information is information about current available consultation services, for example, the total number of customers who are demanding consulting service and the list of representatives who are online. This class will provide the following functions:
  1) Create and initialize UI widgets.
  2) Display customer service information.
  3) Load representative profile according to the client selection.
  4) Connect to the representative according to load balance or connect to the specific representative according to the customer selection.

- *Class IBClientService:* This class will implement the customer side interface and logic for customer/representative online interaction. It provides the following functions:
  1) Create and initialize UI widgets.
  2) Load representative profile that the client is waiting for.
  3) Implement the client/representative interaction logic, include connecting, disconnecting, sending message, exiting etc.
- *Class IBClientPayBill:* This class will implement customer bill payment service. It performs the following fucntions:
  1) Create and initialize UI widgets.
  2) Load all available bill payment services (for example Bell, Videotron etc).
  3) Display customer's bill information according to the customer selection (the bill account number, the amount due etc.)
  4) Perform bill payment operation if the user issues this command.
- *Class IBClientBalance:* This class will implement the display of customer account balances. It performs the following functions:
  1) Create and initialize UI widgets.
  2) Display the customer's bank account balances.
- *Class IBClientTransaction:* This class will implement customer account transactions. It does the following functions:
  1) Create and initialize UI widgets.
  2) Display customer's bank account transactions.
- *Class MessagePanel:* This class is used to display the login fail message.
- *Class WarningPanel:* This class is used to show a warning message before exiting from the system if a client is on waiting list or is being served.

- *Class IBClientContainer*: This class is a Java applet container. It is a container frame to hold Java applet of *IBClient* so CSAC can run as a standalone application.

4.1.1.2. Client side application for representative (CSAR)

The CSAR allows the bank representative to interact with the bank customers and perform bank transactions on behalf of the current customer. It achieves its goals through the following classes:

- *Class IBRep*: This class is used to create the main program of the CSAR . It provides the following five major functions:
  1) Create and initialize UI widgets.
  2) Initialize CORBA infrastructure.
  3) Instantiate representative side callback object, which is an instance of class *IBRepCorbaImpl* (described below).
  4) Locate remote server side CORBA object Bank Manager.
  5) Implement representative quit logic (e.g., tells the server to remove it from its register list).
- *Class IBRepCorbaImpl*: An instance of this class is the client side CORBA object representing bank representative. It is a callback object created on the client side application for representative, which will be registered with the server when the representative logins to the system. Afterwards, the server can remotely invoke callback methods defined in this callback object. When the representative starts to serve a customer, the server will pass this callback object reference to *Client Side Application for Customer* (CSAC), so the customer can directly communicate with the representative as

long as it obtains the object reference. This is how the client/representative interaction is implemented.

- *Class IBRepLogin*: This class will implement the login interface and logic for the representative. It performs the followings functions:
  1) Create and initialize UI widgets for login process.
  2) Register the representative callback object with the server.
  3) Load representative side main user interface if the login is successful, otherwise load a dialog with access denied message.
- *Class IBRepService*: This class is used to implement the representative side main logic for client/representative online interaction. It will contain a list of CORBA callback object references that represent the customers on this representative's waiting list, those object references are assigned by the server side CORBA object Bank Manager in terms of load balance or customer choice (as service waiting list). It differs from the *IBClientService* in the information access privileges. For example, the *IBRepService* can access all bank client information, but *IBClientService* can only access its own client information. This class provides the following functions:
  1) Load and manage customer's waiting list and display the first customer's profile.
  2) Implement client/representative interaction logic, e.g., start serving customer, disconnect, send message/attach file, exit from system etc.
- *IBRepServiceUI:* This class is used to create the representative side main user interface for client/representative online interaction.
- *Class IBTermDeposit:* This class is designed for the representative to perform term deposit transactions for the customer. It provides the representative with a user interface to input the necessary information (including principle amount, interest rates, time period

etc.) and create a term deposit for the customer. A successful transaction will create a tuple in the customer's term deposit account table.

- *Class MessageDialog:* This class is used to display a login fail message.

4.1.2. Tier-2: Server

Tier-2 Server mainly consists of the server side main program (*IBServer*), the class that can interact with the Tier-1 Client through CORBA/IIOP (*IBBankMgrCorbaImpl*), and the class that can interact with Tier-3 Database Server using SQL/JDBC or any other form of middleware (*IBDBManager*). The followings are descriptions of those server side classes.

- *Class IBServer.* This is the server side main program. It provides the following functions:

1) Initialize CORBA infrastructure.
2) Instantiate server side CORBA object bank manager (an instance of IBBankMgrCorbaImpl) and export it to ORB bus.
3) Create database manager object (an instance of *IBDBManager*) and load system database.

- *Class IBServerUI:* This is the class that is responsible to create and initialize the server user interface, which will show the server initialization and session information, i.e., the list of online customers and representatives and their IP addresses.
- *Class IBBankMgrCorbaImpl:* This is the server side core CORBA object representing the bank manager on the server side. It actually serves as the server side coordinator and transaction manager. It can communicate with both bank customers and representatives through registered client side callback objects, and ask the database manager *IBDBManager* to access the database. It provides the following functions:

1) Manages two lists of callback object references (representing clients and representatives respectively).
2) Registers/removes clients and representatives.
3) Connects/disconnects clients to specific representative i.e., add/remove the clients in this representative's waiting list.
4) Asks the bank manager to access system database on behalf of clients and representatives.

- *Class IBDBManager:* This is the server side Java class that will implement JDBC logic. It is the only class in the system that can directly access the third tier database using SQL/JDBC protocols. It can query and update client and bank employee information stored on the database at the request of the bank manager.
- *Class IBDBCreator:* This class is designed to create the initial bank database. It will first drop all existing bank database tables, then create new tables, and populate the tables from data files.
- *Class IBDatabase:* This class is used to provide a graphic user interface for the bank database. Users can view the contents of the tables by selecting the table name.

4.1.3. Tier-3: Database Server

Tier-3 consists of the database management system DBMS and the bank database, based on the selected DBMS. Tier-1 Client objects never directly access the database, but instead they access the third tier data source via the Tier-2 sever objects (the database manager, an instance of *IBDBManager*, and the bank manager, an instance of *IBBankMgrCorbaImpl*). The database manager on the Tier-2 server side will access the Tier-3 database and send the

results back using SQL/JDBC protocols. The system will choose MYSQL as the DBMS [6]. The bank database will be described in more details in section 3 (4.3 Database design).

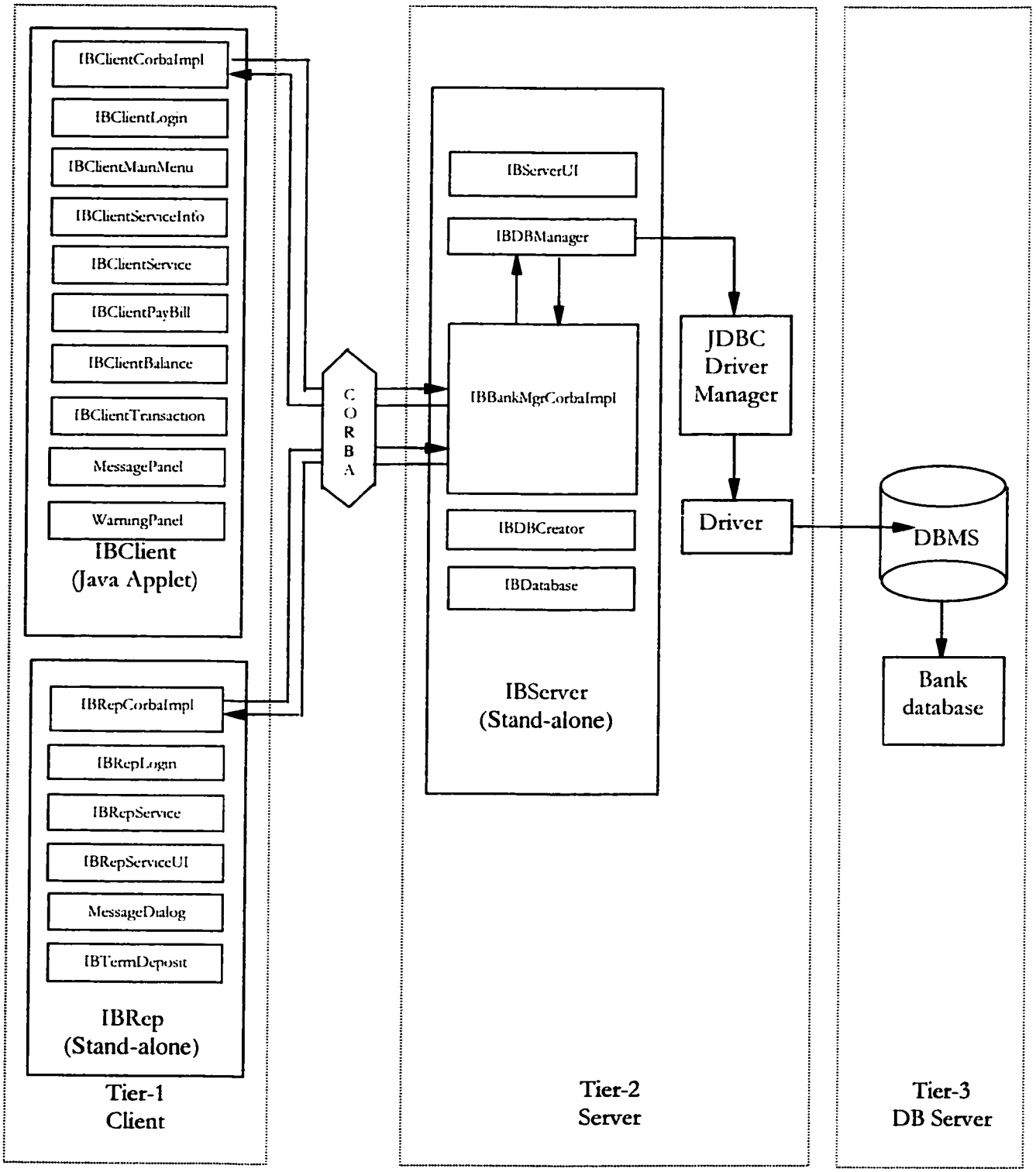

Figure 4.1 Architecture of Interactive Internet Banking System

4. 2 Class Diagrams

This section presents the class diagrams for the *Interactive Internet Banking System* (IIBS). A class diagram describes the types of objects in the system and various kinds of relationships that exist among them. The relationships may include aggregation (represented by symbol ◊), inheritance (represented by symbol Δ), and dependence (represented by connection line). The class diagram is drawn using UML (Unified Modeling Language) notations.

Figure 4.2 illustrates the class diagram for the *Client Side Application for Customer* (CSAC). It describes the classes that are used to create the client application and their relationships from the perspective of implementation.

Figure 4.3 shows the class diagram for the *Client Side Application for Representative* (CSAR). It describes the classes that are used to create the representative application and their relationships from the perspective of implementation.

Figure 4.4 shows the class diagram for server side application. It describes the classes that are used to create the server application and their relationships from the perspective of implementation.

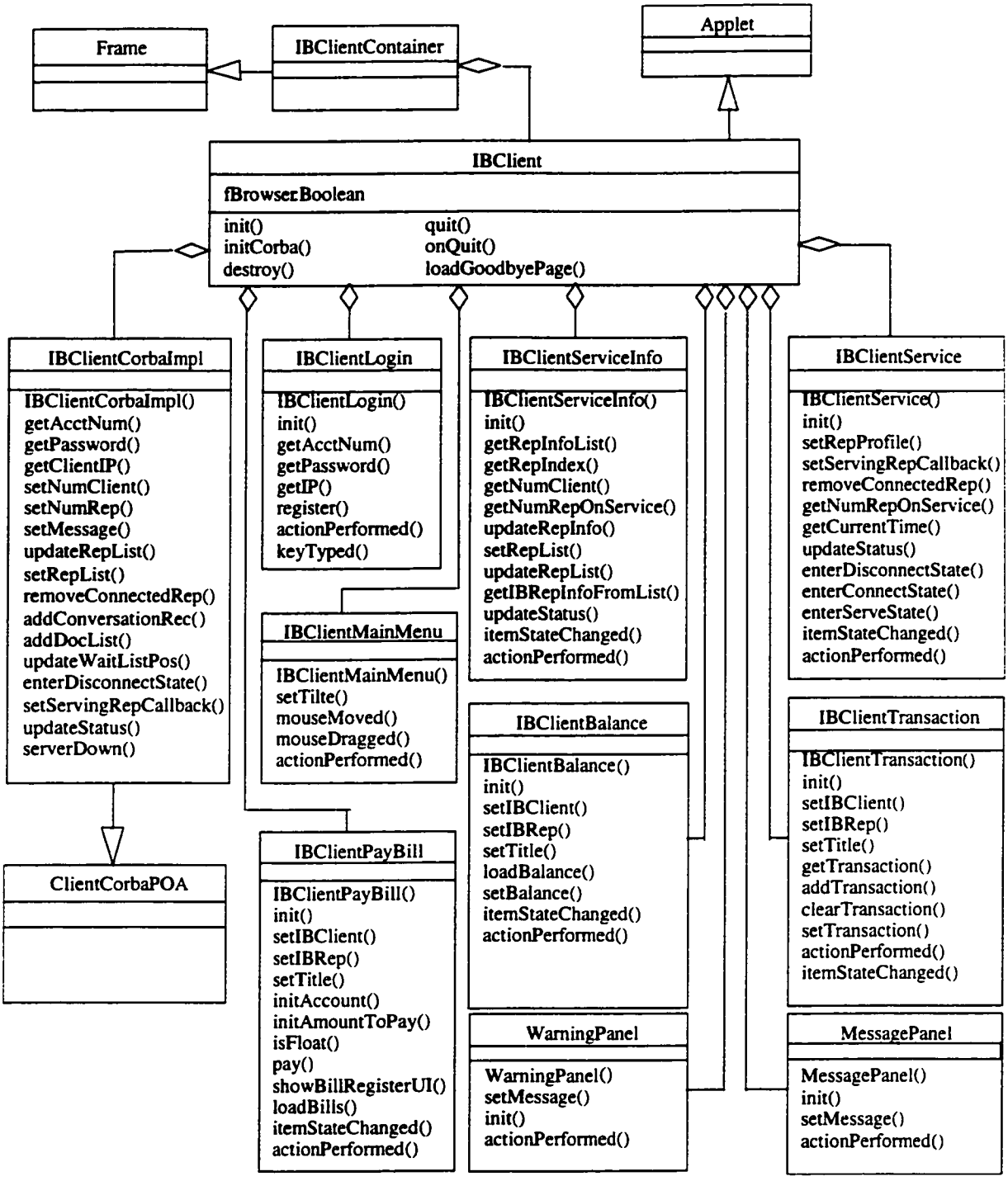

Figure 4.2 Class Diagram for CSAC

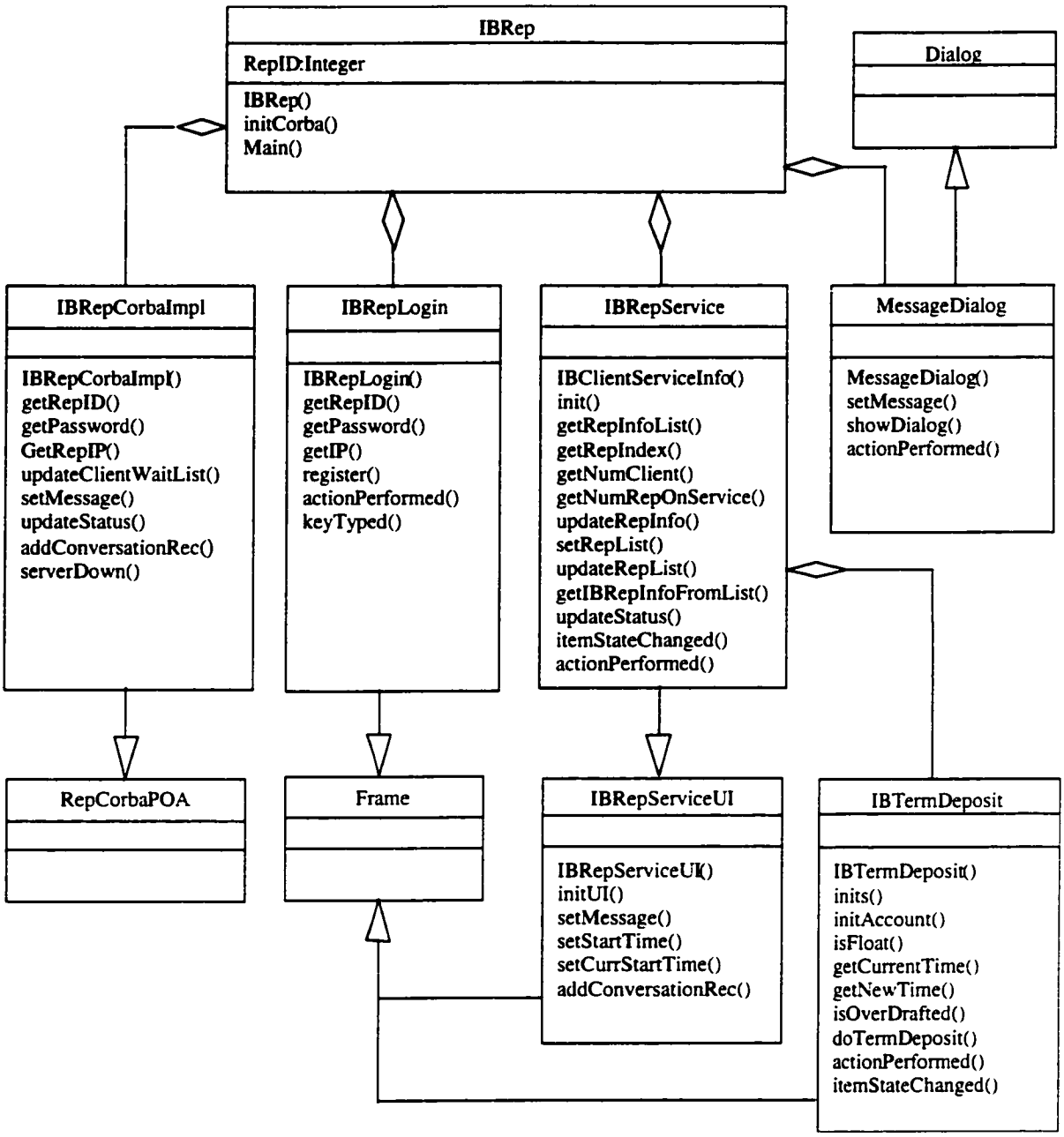

Figure 4.3 Class Diagram for CSAR

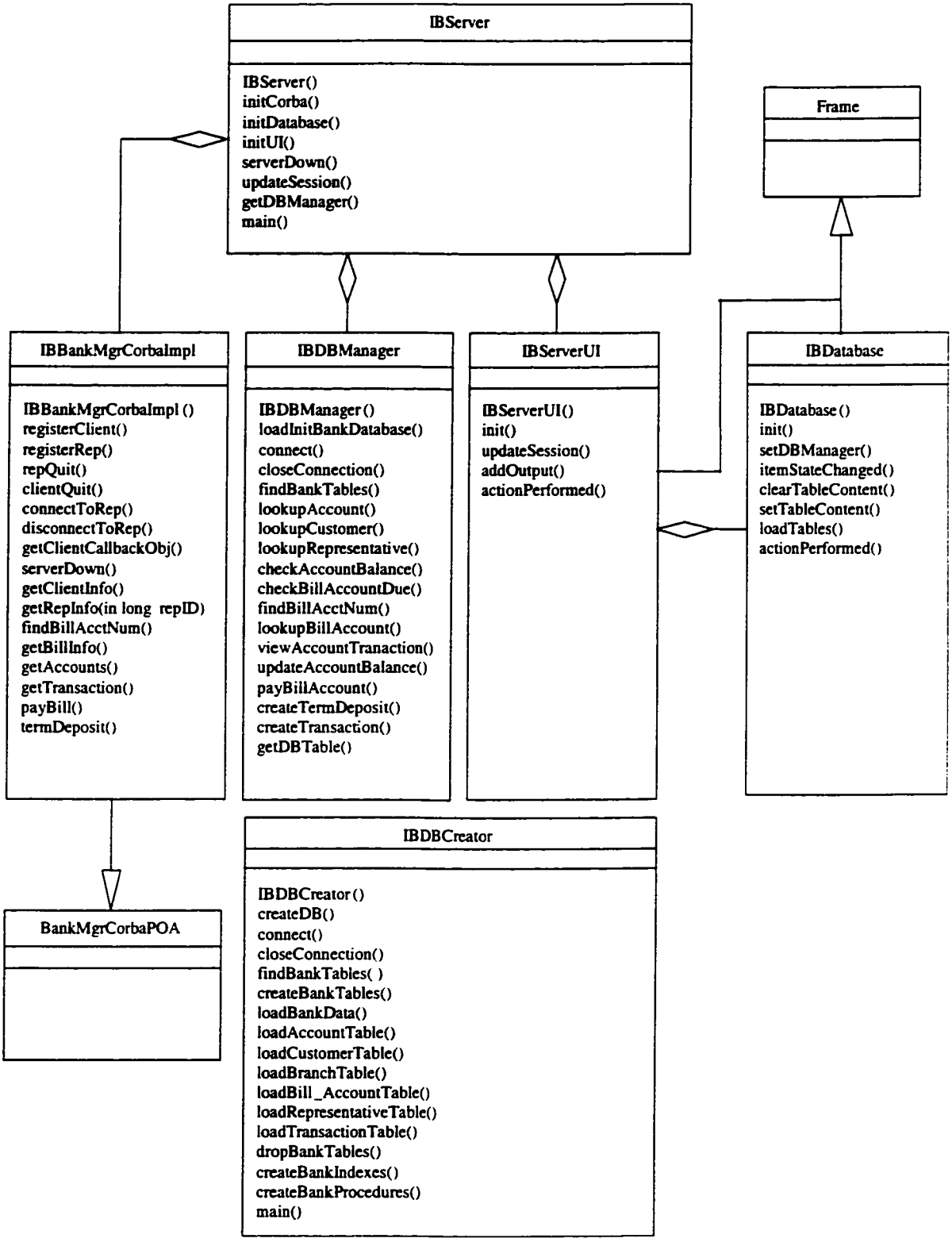

Figure 4.4 Class Diagram for Server Side Application

4. 3 Database Design

This section describes the design of the database that is used in the system.

4.3.1. Design Rationales

The database of the *Interactive Internet Banking System* (IIBS) stores the data related to customers, accounts, bills and representatives. It is designed under the following considerations:

- A bank account is identified by the account number and account type. One customer can have more than one account, but only one account number for each customer. The different accounts for one customer are distinguished by account type. So the account number serves as unique id for each customer, it is used in customer related tables to identify this customer, and force the integrity constraints.
- The Bill-account table stores information about the bills. The primary key for the bill account is bill account number that will be incremented automatically whenever a new bill tuple is added to the table. Different bills for the same customer are stored in the same table and distinguished by bill name (e.g. Bell, Videotron etc.), and the bills for different customer are distinguished by bank account number.
- The Term-account table stores information about the customer's term deposit. Whenever a term deposit transaction is made and confirmed, a new tuple will be created in this table to store related data. The term account has a term account id as its primary key, and has the bank account number as foreign key to reference the specific customer.

- The Transaction table stores all the transaction data related to a particular bank account. Therefore we need both bank account number and account type in transaction table as foreign keys to reference the particular account, to which the transaction applies.
- The Representative table has an attribute Photo, which is actually a file name to identify the image file (JPG format) of the representative, the image files for all representatives are stored in a predefined directory of Tier-2 server machine.

4.3.2. ER-Model

The database is comprised of seven tables (entities). The inter-relationships between those entities are shown in the ER model shown in **Figure 4.5**.

4.3.3. Database Schema

The notions that describe the database schema are as followings:

- Primary key: in bold and underline
- Foreign key: in bold, italic and underline

The followings are the database schema for the seven entity tables:

1. *Account* (**Acct-num**, **Acct-type**, Password, Branch-ID, Balance, Date)

Where Primary keys: Acct-num

Acct-type

2. *Customer* (**Acct-num**, Name, Address, Telephone)

Where Primary key: Acct-num

3. *Term-account* (**Term-ID**, ***Acct-num***, Capital, Interest-rate, Start-Date, End-Date)

Where Primary key: Term-ID

Foreign key: Acct-num to reference the particular customer

4. *Branch* (**Branch-ID**, Address)

Where Primary key: Branch-ID

5. *Bill-account* (**Bill-acct-num**, Bill-acct-name, ***Acct-num***, Balance-forward, Amount-due)

Where Primary key: Bill-acct-num

Foreign key: Acct-num to reference the particular customer

6. *Representative* (**Rep-ID**, Name, Branch-ID, Address, Photo)

Where Primary key: Rep-ID

7. *Transaction* (**Trans-ID**, ***Acct-num***, ***Acct-type***, Description, Debit-credit, Amount, Trans-Date)

Where Primary key: Trans-ID,

Foreign key: Acct-num and Acct-type to reference the particular account

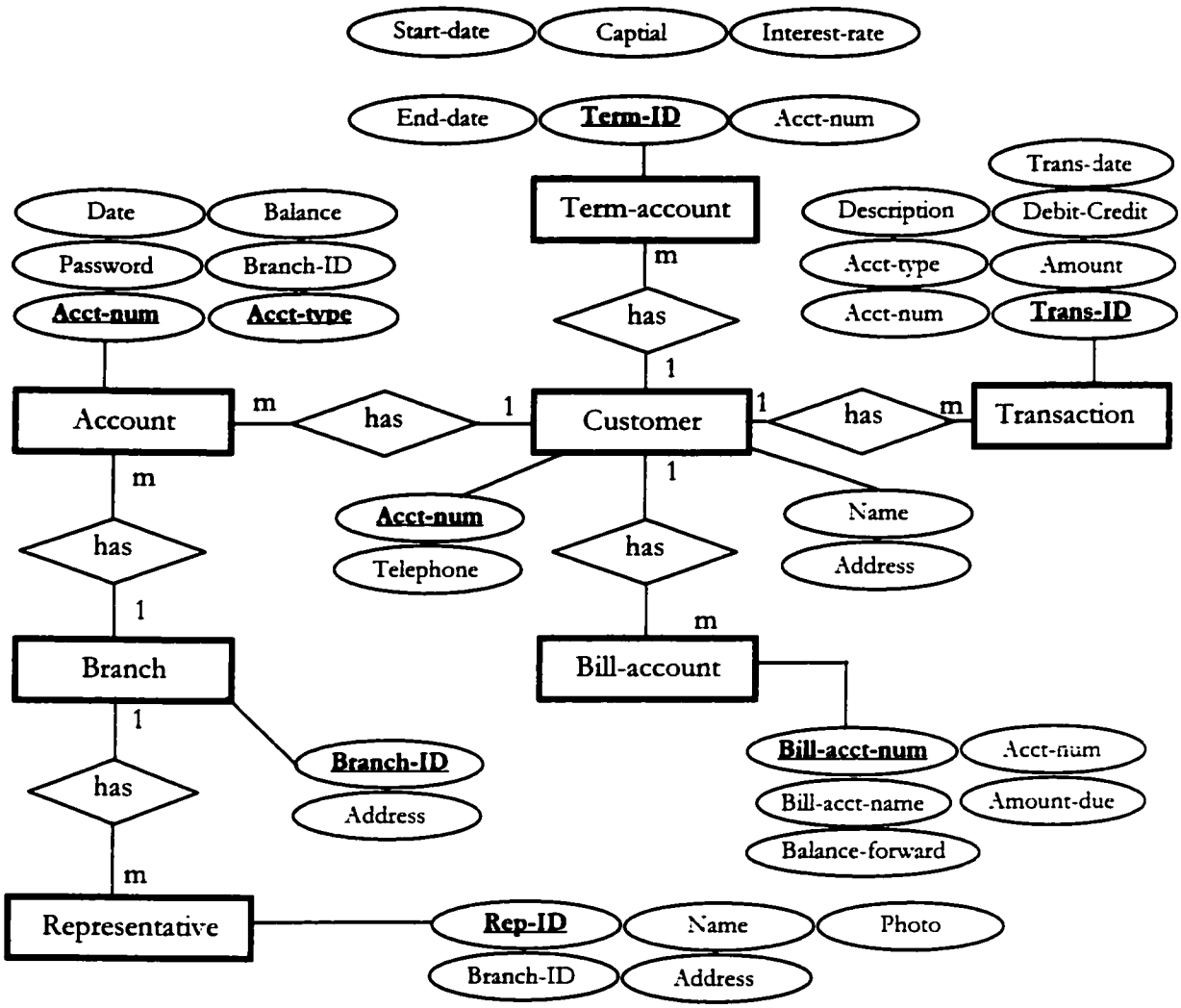

Figure 4.5 ER Model

Chapter 5

5. System Implementation

This chapter presents a detailed description on how *Interactive Internet Banking System* (IIBS) is implemented using CORBA/Java/JDBC technology. First, the general process to develop the CORBA based client/server application will be explained and then how the system is implemented by following this process is described.

5. 1 General Process of Developing CORBA Based Client/Server Application

To develop the distributed, object-based application with CORBA, we must first identify the objects (especially the distributed objects) required by the application, and then follow the general steps:

- Write a specification for each distributed CORBA object using the Interface Definition language (IDL), which is a *descriptive language* to describe the interfaces being implemented by the CORBA remote object.
- Use the IDL compiler (usually shipped with CORBA development tools) to generate the client stub code and server POA servant code (server skeleton code), which provides the interfaces for the remote object.
- Write the client application code.

- Write the server object code and server application code.
- Compile the client and server code.
- Start the server and web server (if clients use applet to access server).
- Run the client application.

5.2 IIBS Implementation

IIBS is a CORBA based client/server system, so it utilizes the CORBA objects to realize client/server communications. Apart from the client and server interactions between Tier-1 and Tier-2, the system also involves significant two-way communications between the customers and bank representatives inside Tier-1. The system implementation employs the ***callbacks*** technique to efficiently establish the communications between the client and server and two kinds of clients, the bank customers and employee. The callback is a call from a server to a client. It reverses the client and server roles, by allowing a client to become a server. So the CORBA object that is created in the client side is a callback object and it is registered to the server by passing its object reference when the client logins to the system. Since the server holds the object reference, it can remotely invoke it and pass this reference to other clients, allowing the other clients to talk to each other directly. The callbacks make the IIBS a very sophisticated and flexible client/server system.

5.2.1 Define Object Interfaces

To implement the system, three CORBA objects are defined, which are crucial for client/server communications. They are described below:

1. **ClientCorba**

This is the client side callback object representing bank customer. The following is the IDL that defines the interface of this object (e.g. the callback methods that can be remotely invoked):

```
module Client
   {
     interface ClientCorba
     {
        // client login
        string getAcctNum();
        string getPassword();
        string getClientIP();
        // client service info
        void setNumClient(in string s);
        void setNumRep(in string s);
        void setMessage(in string s);
        void updateRepList(in CRepInfo repInfo, in long flag);
        void setRepList(in RepInfoList repInfoList);
        // client service
        void removeConnectedRep(in long repID);
        void addConversationRec(in string s);
        void addDocList(in string s);
        long updateWaitListPos(in long step);
        void enterDisconnectState();
        void setServingRepCallback(in Representative::RepCorba
        repObjRef);
        // others
        // status: 0=free, 1=waiting list, 2=being served
        void updateStatus(in long status);
        void serverDown();
     };
};
```

2. RepCorba

This is the client side CORBA object representing bank representative. The following is the IDL that defines the interface of this object (e.g. the callback methods that can be remotely invoked):

```
module Representative
   {
     interface RepCorba
     {
        // client login
        string getRepID();
        string getPassword();
        string getRepIP();
        void   updateClientWaitList(in CClientInfo clientInfo, in long
        flag);
        void   setMessage(in string s);
```

```
        void    updateStatus(in long status);
        void    addConversationRec(in string s);
        void    serverDown();
    };
};
```

3. BankMgrCorba

This is the server side core CORBA object representing the bank manager. The following is the IDL that defines the interface of this server object:

```
module Server
{
    interface BankMgrCorba
    {
      long  registerClient(in Client::ClientCorba clientObjRef);
      long  registerRep(in Representative::RepCorba repObjRef);
      void  repQuit(in long repID);
      void  clientQuit(in long clientID);
      void  connectToRep(in long clientID, in long repID);
      void  disconnectToRep(in long clientID, in long repID);
      Client::ClientCorba getClientCallbackObj(in long clientID);
      void  serverDown();
      // utility
      CClientInfo getClientInfo(in long clientID);
      CRepInfo getRepInfo(in long repID);
      // Database access functions
      string findBillAcctNum(in long bankAcctNum, in string
      billName);
      CBillInfo getBillInfo(in string acctNum);
      CAccountArray getAccounts(in long clientID);
      string getTransaction(in long clientID,in stringacctType);
      void payBill(in string acctNum, in float amount, in long
      bankAcctNum, in string bankAcctType);
      void termDeposit(in long acctNum, in string acctType, in
                       float capital, in float interestRate, in
                       string startDate, in string matureDate);
    };
  };
```

5.2.2 Generate Client Stubs and Server Servants

VisiBroker's idl2java compiler is used to generate client stubs and server servant (server skeletons) Java code for each of above three objects. The following is the description of those IDL generated code:

1) Java source files generated by compiling *ClientCorba* IDL

- *ClientCorbaHelper.java:* Declares the ClientCorbaHelper class, which defines helpful utility methods.
- *ClientCorbaHolder.java:* Declares the ClientCorbaHolder class, which provides a holder for passing ClientCorba objects.
- *ClientCorba.java:* The ClientCorba interface declaration.
- *ClientCorbaOperations.java:* This interface provides to declare the method signatures defined in the ClientCorba interface in the object's IDL file.
- *ClientCorbaPOA.java:* POA servant code (implementation base code) for the ClientCorba object implementation on the server side.
- *ClientCorbaPOATie.java:* Class used to implement the ClientCorba object on the server side using the tie mechanism (this is not used in this project).
- *ClientCorbaStub.java:* Stub code for the ClientCorba object on the client side.

2) Java source files generated by compiling *ClientRep* IDL

- *RepCorbaHelper.java*: Declares the RepCorbaHelper class, which defines helpful utility methods.
- *RepCorbaHolder.java:* Declares the RepCorbaHolder class, which provides a holder for passing RepCorba objects.
- *RepCorba.java:* The RepCorba interface declaration.
- *RepCorbaOperations.java:* Declares the method signatures defined in the RepCorba interface in the object's IDL file.
- *RepCorbaPOA.java:* POA servant code (implementation base code) for the RepCorba object implementation on the server side.

- *RepCorbaPOATie.java:* Class used to implement the RepCorba object on the server side using the tie mechanism (this is not used in this project).
- *RepCorbaStub.java:* Stub code for the RepCorba object on the client side

  Note: The server side refers to the place where the object is created, and the client side refers to the place from which the object is remotely invoked. The callback object is actually created on client sides (either on the side of bank customer or bank representative), so the meaning of client and server is reversed in this case.

3) Java source files generated by compiling BankMgrCorba IDL
   - *BankMgrCorbaHelper.java:* Declares the BankMgrCorbaHelper class, which defines helpful utility methods.
   - *BankMgrCorbaHolder.java:* Declares the BankMgrCorbaHolder class, which provides a holder for passing BankMgrCorba objects.
   - *BankMgrCorba.java:* The BankMgrCorba interface declaration
   - *BankMgrCorbaOperations.java:* This interface provides to declare the method signatures defined in the BankMgrCorba interface in the object's IDL file
   - *BankMgrCorbaPOA.java:* POA servant code (implementation base code) for the BankMgrCorba object implementation on the server side.
   - *BankMgrCorbaPOATie.java:* Class used to implement the BankMgrCorba object on the server side using the tie mechanism (this is not used in this project).
   - *BankMgrCorbaStub.java:* Stub code for the BankMgrCorba object on the client side

5.2.3 Implementation of the Objects

The stub and servant code generated from IDL files is the implementation base for CORBA objects. To actually implement these objects, the following three implementation classes are needed:

- Class *IBClientCorbaImpl* extends *Client.ClientCorbaPOA* and implements all methods defined in *ClientCorba* IDL. This class is the implementation of client side CORBA callback object *ClientCorba.*
- Class *IBRepCorbaImpl* extends *Representative.RepCorbaPOA* and implements all methods defined in *RepCorba* IDL. This class is the implementation of client side CORBA callback object *RepCorba.*
- Class *IBBankMgrCorbaImpl* extends *Server.BankMgrCorbaPOA* and implements all methods defined in *BankMgrCorba* IDL. This class is the implementation of server side core CORBA object *BankMgrCorba.*

5.2.4 Implementation of the Client Applications

IIBS consists of two client applications in Tier-1: *Client Side Application for Customer* (CSAC) and *Client Side Application for Representative* (CSAR). Here clients refer to both the bank customers and representatives in the system. The following subsections will describe how *CASC* and CSAR are implemented using related CORBA objects and Java classes.

5.2.4.1 Implementation of CSAC

CSAC implementation is based on the several major Java classes. The following section will describe the implementation highlights of each class:

- *IBClient:* This class is a Java class that extends *java.awt.Applet*. When it is downloaded by a web browser, its standard initialization function is called to perform necessary initialization. The initialization involves the following two things:

  1) Instantiate all the classes used by CSAC.

  2) Initializes CORBA infrastructure. This includes creating callback object *ClientCorba* representing the bank customer and locating server side CORBA object *BankMgrCorba* representing the bank manager. The callback object will be registered with the server when the client logins to the system. The server or other clients (get callback object reference from the server) will remotely invoke its methods. The object reference of the bank manager will be used to access the services provided by the server, for example, register the client, access the database etc.

  When the user closes the browser, it has to perform clean up by remotely invoking *BankMgrCorba::clientQuit* method, which will remove this client from the bank manager's register list and update all registered clients' user interface. For example, if this customer is still in a particular representative's waiting list, it has to be removed from this list.

- *IBClientLogin:* This Java class is derived from *java.awt.Panel*. It implements the first page of CSAC. It allows the user to login by entering the account number and password. If the login succeeds, it will load the main menu, otherwise, it will popup a message box with an access denied error message. The registration is performed by remotely invoking *BankMgrCorba:: registerClient(Client.ClientCorba clientObjRef)* method. Please note that the callback object reference of *ClientCorba* representing the customer is passed in as a parameter. On the server side, this method will remotely invoke *ClientCorba::getPassword* and *ClientCorba::getAcctNum* method (the *callbacks* is working) to get the account number and password entered by the user, and verify with the user data in database. If it finds

the match, it places the callback object reference into the register list and returns with clientID>0, otherwise it returns with clientID<=0 (the password/account number is invalid or the client has already been registered).

- *IBClientMainMenu:* This Java class is derived from *java.awt.Panel.* It provides the main menu to the bank customer to access all of the bank services available in the system. It simply loads the corresponding user interface according to the user selection.
- *IBClientBalance:* This Java class implements the display of the client's account balance. It does this by invoking remote method *BankMgrCorba::getAccounts().* This method returns an array of CORBA object *CAccount,* which contains multiple account information about the user. Current implementation supports up to five accounts (saving account, check account etc). Where each *CAccounts* is defined in IDL as the following:

```
struct  CAccount
{
    char type;
    long branchID;
    float balance;
};
```

- *IBClientPayBill:* This Java class implements the bill payment service. It first loads all available bill payment information related to the user, including the bill account number, balance forward, and amount due. The current implementation supports two bill payment services (Bell and Videotron). Because the bill information is kept in the server database, it does this by remotely invoking *BankMgrCorba::getBillInfo.* This method will return a *cBillInfo*, which is a CORBA object defined by in IDL as the following:

```
struct CBillInfo
{
    string acctNum;
    string name;
    float  balanceForward;
    float  amountDue;
};
```

The actual bill payment operation is performed by invoking *bankMgrCorba::payBill*, which will deduct the specified amount from selected bank account and update the user's bill account data (balance forward and amount due).

- *IBClientServiceInfo:* This Java class is implemented to display the customer service information and allows the customer to connect to the representatives. The server maintains a list of registered bank customers and another list of bank representatives. This information will be delivered to all registered customers consistently and must be updated whenever applicable (e.g. when a representative quits from the system). When the client is registered to the server, it remotely invokes *BankMgrCorba::registerClient* with callback reference *ClientCorba* as a parameter. this method calls *ClientCorba::setRepList(CRepInfo[] cRepInfoList)* (the *callbacks* is working), which will return *cRepInfoList*. The *cRepInfoList* is an array of *CRepInfo* CORBA objects, where it contains all the information related to setup customer service information on the client side. Whenever a change occurs, the server will loop through all its registered clients and call *ClientCorba::updateRepList(CRepInfo cRepInfo, int flag)*, which will effectively update the service information on the client side with the latest data. Each *CRepInfo* in array *cRepInfoList* is defined in IDL as the following:

```
struct CRepInfo
{
    long   ID;
    string name;
    long   branchID;
    string address;
    string telephone;
    string photoFile;
    string IP;
    long   numClients;
    long   estimateWaitTime;
};
```

This class also implements the connection to the desired representative. It does this by remotely invoking *BankMgrCorba::connectToRep(int clientID, int repID)*, which will assign the customer's callback object reference to the representative and place it in the waiting list.

- *IBClientService:* This Java class implements the customer/representative interaction on the client side. This interface displays the profile of the selected representative, the user's order in the representative's waiting list, and the start time of the connection. The user can interact with a representative by sending and receiving messages. This conversation between the customer and the representative is displayed in this interface.

  The representative's profile is displayed when the customer connects to the representative. The order of waiting list is set by callback method ClientCorba::*updateWaitListPos*, which is called in *BankMgrCorba::connectToRep*. When the connected representative is ready to serve the customer, it will call *ClientCorba::setServingRepCallback (Representative.RepCorba repCorba)*. This method will assign the representative's callback object *repCorba* to the customer, so the interaction can start. The customer sends messages to the representative by calling *RepCorba::addConversationRec*.

  At any time, the user can disconnect from the representative. The disconnection operation is performed by invoking *bankMgrCorba::disconnectToRep(int clientID, int repID)*, which will remove the client from the representative's waiting list and broadcast to registered clients to update customer service information (the number of customer demanding service is decreased by one, the waiting list order of the clients behind this customer will advance by one etc). This will be shown by user interfaces being updated accordingly.

- *IBClientTransaction:* This Java class is implemented to display the transactions of this client's bank accounts. It does this by remotely invoking *bankMgrCorba::getTransaction(int clientID, String acctType)*, which will return a formatted string containing transaction information.
- *IBClientContainer:* This is a Java class derived from *java.awt.Frame*. It implements a frame to hold applet *IBClient* and makes the *CSAC* a standalone application.

5.2.4.2 Implementation of client side application for representative (CSAR)

CSAR is a standalone Java application. The following section describes the implementation highlights of the major classes:

- *IBRep:* This Java class extends *java.awt.Frame*, it implements CSAR main program. This is very similar to the *IBClient*, in that the main functionalities of this class are initialization and clean up. The initialization process instantiates all the classes used by CSAR, and initializes the CORBA infrastructure, which will create client callback object *RepCorba* representing the bank representative.

  Similar to the *IBClient*, when the user closes the window, it has to perform clean up by remotely invoking *BankMgrCorba::repQuit* method. This will remove this representative from the bank manager's register list and update all registered clients' user interface. For example, if this representative has a customer waiting list, it has to remove all the customers from this waiting list and updated all the related customers' user interfaces.
- *IBRepLogin:* This Java class implements the login interface and logic for CSAR. It allows the user to login by entering the employee ID and password. If the login succeeds, it will load the main interface for the representative, otherwise it will popup a dialog box with an access denied message. This implementation is very similar to

IBClientLogin, except the difference is the callback object; *RepCorba* is used to register to the server instead of *ClientCorba*.

- *IBRepService:* This Java class implements the customer/representative interaction on the representative side. The main interface displays a customer waiting list and the profile of the customer currently being served. The user can interact with the current customer by sending and receiving messages (including attachment files). The conversation between the customer and bank representative is displayed in this interface. The representative can also perform bank transactions for the customer based on the customer's request. The waiting list expands if new clients connect to this representative or shrinks when the clients disconnect from the representative or the representatives remove the first client from the list. The representative can not alter the order of the waiting list, but at any time, he/she can disconnect the first customer in the waiting list. Because the waiting list is part of customer service information, which will be shared among all customers, the actual operations on the list are performed on the server side by *bankMgrCorba::connectToRep(int clientID, int repID)* (when the customers connect to this representative) and *bankMgrCorba::disconnectToRep(int clientID, int repID)* (when the customers disconnect from the representative or the representative remove the first client from the list). The user sends messages to the customer by calling *ClientCorba.addConversationRec*.

5.2.5 Implementation of the Server

The server program consists of the main program, the server object that can interact with Tier-1 client through CORBA/IIOP (the bank manager), and the server object that can

interact with Ter-3 database server using SQL/JDBC (the database manager). The following section describes the implementation highlights of the main program, the bank manager, and the database.

5.2.5.4 The Main Program

The main program is implemented by two Java classes: *IBServer* and *IBServerUI*. The *IBServer* extends the *IBServerUI*. The *IBServerUI* extends *java.awt.Frame*, and is responsible for creating the user interface for the server. The server main program performs the following operations to set up the system server.

- *Initialize the server CORBA infrastructure.* It instantiates the server side CORBA object, the bank manager (which is an instance of *IBBankMgrCorbaImpl)*, and exports it to ORB bus. The bank manager exposes all the services that the server can provide to its clients through remote invocations.
- *Initialize bank database.* It creates the database manager object (which is an instance of *IBDBManager)*, and optionally loads system database (if the database manager detects there is no table in the given database). The database manager provides all the functions that can be used to access the system database.
- *Initialize the server interface.* The interface will show the server initialization and session information, including the list of online customers and representatives and their IP addresses.

5.2.5.4 Implementation of the Bank Manager

The class *IBBankMgrCorbaImpl* is the implementation of the server side CORBA object representing the bank manager. The methods of this class reflect the services that the server can provide to its clients. The following methods are implemented in this class:

1) *int registerClient (Client.ClientCorba clientObjRef)*

   Paremeter: Client side callback object reference

   Return: The client ID

   This function allows the client to register to the server. It places the client in the registered client list, and sends the client customer service information (information about all registered representatives) through the client callback.

2) *int registerRep (Representative.RepCorba repObjRef)*

   Parameter: Representative side callback object reference

   Return: The representative ID

   This function allows the representative to register to the server. It places the representative in the server's registered representative list, adds this representative to the representative list on the client user interface, and broadcasts this news to all registered clients.

3) *void clientQuit (int clientID)*

   Parameter: The client ID, which is returned from function registerClient

   This function is called by the CSAC to perform clean up when the customer quits from the system. It removes the client from the registered client list. If the client is in a representative's waiting list, it removes this client from the list and updates all registered clients'user interface.

4) *void repQuit (int repID)*

Parameter: The representative ID, which is returned from function registerRep

This function is called by the CSAR to perform clean up when the representative quits from the system. It removes the client from the registered representative list and updates all of the registered client's customer service information, removes this representative from the representative list on client user interface, and broadcasts this news to all clients. If this representative has customers waiting, it informs all those waiting clients and updates the interface of the waiting clients through the callback.

5) *void connectToRep(int clientID. int repID)*

Paramaters: clientID: The client ID, which is returned from function registerClient

repID: The representative ID, which is returned from function registerRep

This function allows the client to connect to a particular representative. It places the client in this representative's waiting list and updates all registered customer service information: the number of total clients and the number of clients on this representative's waiting list increase by 1.

6) *void DisconnectToRep((int clientID, int repID)*

Parameters: clientID: The client ID, which is returned from function registerClient

repID: The representative ID, which is returned from function registerRep

This function allows the client to disconnect from a particular representative, or a representative removes the first client from his/her waiting list. It performs the opposite operation as *ConnectToRep.*

The following functions allow the clients to access the bank database. The bank manager simply delegates to the corresponding function to the database manager, which implements actual database access.

7) *String FindBillAcctNum(int bankAcctNum, String billName)*

Parameters: bankAcctNum: The bank account number

billName: The name of the bill account

return: The number of the bill account

This function returns the account number of the named bill account associated with the bank account.

8) *CBillInfo getBillInfo(in string acctNum)*

Parameter: acctNum: The bill account number

Return: The reference to CBillInfo, which contains information about bill account

This function returns the information about the bill account.

9) *CAccount[] getAccounts(int clientID)*

Parameter: The client ID, which is returned from the function registerClient

Return: Array of CAccount object references, which contain information customer's bank accounts

This function returns the bank account information about this customer.

10) *String getTransaction(int clientID, String acctType)*

Parameters: ClientID: The clientID

AcctType: The bank account type (saving, checking etc)

Return: The formatted string contains information about the transactions occurred in this bank account.

This function returns the transactions occurred in this bank account.

11) *void payBill(String acctNum, float amount, int bankAcctNum, String bankAcctType)*

parameters: acctNum: Bill account number

amount: The amount to pay

bankAcctNum: The bank account number from which money is withdrawn

bankAcctType: Bank account type (saving, check, etc)

This function performs pay bill transaction.

12) void termDeposit(int acctNum, String acctType, float capital, float interestRate,

String startDate, String matureDate)

Parameters: acctNum: The bank account number

acctType: Bank account type

capital: Capital of the term deposit

interestRate: Interest rate of the term deposit

startDate: Start date of the term deposit

matureDate: Mature date of the term deposit

This function performs term deposit transaction. It creates a term deposit for the customer and the capital is withdrawn from the given bank account.

5.2.5.4 Implementation of the Database

The database implementation is based on three Java classes, *IBDBCreator*, *IBDatabase*, and *IBDBManager*.

- *IBDBCreator:* This is the Java class that implements the database initialization. It drops all existing tables, creates all empty tables, and populates the tables with data read from data files. The following six text files are used to initialize the correspondent six tables:

  1) *account.dat:* Data for table Account

  2) *customer.dat:* Data for table Customer

  3) *branch.dat:* Data for table Branch

  4) *representative.dat:* Data for table Representative

  5) *bill.dat:* Data for table Bill-Account

  6) *transaction.dat:* Data for table Transaction

  Table Term-account is initially empty. There is no data file for this table.

The data file format is very simple, where each line represents a table entry (a tuple) with each string separated by the TAB, for example, the following two lines are taken from the *account.dat*:

100000 A12345 C 01 3000.99 2000/6/20

100000 A12345 S 01 2000.99 2000/6/20

- *IBDatabase:* This is the Java class that implements a simple graphical user interface for bank database. It is accessible from server side and is used for demonstration purposes.
- *IBDBManager:* This is the Java class that implements the database manager. Database manager implements actual SQL/JDBC logic. The following functions are implemented to enable the clients to access the database.

1) public void connect() throws SQLException

Connect to bank database.

2) public void closeConnection() throws SQLException

Disconnect from bank database.

3) public CAccount_ lookupAccount(String acctNum, String password) throws SQLException

Retrieves all account information given an account number and password.

4) *public boolean lookupCustomer(String acctNum, IBClientInfo clientInfo) throws SQLException*

Retrieves customer information given an account number. The return value is filled inside *clientInfo*.

5) *public IBRepInfo lookupRepresentative(String repID, String password) throws SQLException*

Retrieves the bank representative's information given an ID and password.

6) public float checkAccountBalance(String acctNum, String acctType) throws SQLException

Gets the account balance of the bank account identified by the given account number and type (check, saving etc.).

7) public float checkBillAccountDue(String acctNum) throws SQLException

Gets the amount due of given bill account identified by the given bill account number.

8) public String findBillAcctNum(String bankAcctNum, String billName) throws SQLException

Finds the specific bill account number associated with the bank account.

9) public boolean lookupBillAccount(String acctNum, CBillInfo cBillInfo) throws SQLException

Retrieves the bill information of the specific bill account. The bill information data is returned in *cBillInfo*

10) public String viewAccountTranaction(String acctNum, String acctType, String startDate, String endDate) throws SQLException

Retrieves all transactions incurred in the specific bank account during the specific period of time.

11) public boolean updateAccountBalance(String acctNum, String acctType, String amount) throws SQLException

Updates the bank account balance with the specific amount.

12) public void payBillAccount(String acctNum, String amount) throws SQLException

Pays the specified bill account with the specific amount.

13) public boolean createTermDeposit(String acctNum, float capital, float interestRate, String startDate, String matureDate) throws SQLException

Creates a term deposit entry in Term-account table with specified data.

public boolean createTransaction(String acctNum, String acctType, String desc, String debitCredit, String amount, String debitCredit, String amount) throws SQLException

Creates a transaction entry in Transaction table with specified data.

5.2.6 Compilation of the Client and Server Code

Microsoft Visual J++ 6.0 is used to compile and debug the program. Follow the following steps to configure the IDE and compile the code.

- Create a Visual J++ 6.0 project, called IBBS.
- Add all of the Java source files (including generated client stub server servant code) to this project.
- To enable the CORBA and JDBC support, it is important to set the class path of IIBS project properties as the following: Open IIBS project properties dialog, click the Classpath tab, and add the following three class path entries.

  C:\Inprise\vbroker\lib\vbjorb.jar

  C:\Inprise\vbroker\lib\vbjdev.jar

  G:\comp793\project\mySqlJdbcDrv (contains the MySQL3.23.11 JDBC driver Java class)
- As long as the above class paths are set, you just click Build and the IDE will generate the client and server Java byte code, which can run under any Java Virtual Machine (J++, JDK, Visibroker for Java, JBuilder etc.). In order to run the server program, you will need Visibroker for Java for CORBA support. See Appendix A for more details.

Chapter 6

6. Conclusions

In this project a fully-fledged 3-tier client/server CORBA/Java/JDBC application called *Interactive Internet Banking System (*IIBS*)* was designed and implemented. This application provides both traditional online banking services and a novel online consultation service. The system was implemented and tested at Concordia University's Computer Science labs and met all its functional and non-functional requirements.

The system was implemented using distributed objects paradigm with CORBA/Java/JDBC technology. From my experiences with developing IIBS, I found CORBA to be a very powerful middleware for developing client-sever based software system. CORBA deals with inter network transparency while Java is definitely an ideal language for writing client and server CORBA objects. The combination of CORBA and Java brings object oriented programming into distributed computing environment, which allows the programmers to build very complex client/server systems. This has been clearly demonstrated in the implementation of this project.

IIBS is a complete bank system with great value for practical use. To make the system more sophisticated and adapted for commercial use, the following 2 major enhancements Security and More multimedia capabilities should be further incorporated:

- *Security issues:* Commercial bank system naturally raises security concerns. To ensure secure information transmission on the Internet, the system needs to implement secure

encryption and decryption schemes. Client side data including user name, password, and other transaction data should be encrypted prior to sending to server, and then the server has to decrypt the client data according to the encryption scheme used in client side. This principle also applies to the data transmissions from server to client.

- *More multimedia capabilities*: IIBS has two parts: the traditional online banking service and a novel online consultation service. The second part makes it possible for the system to take full advantage of a computer's multimedia capabilities to fulfill more complex operations. For example, opening an investment account for existing customers. In this case, the representative will fill the form related to investment account with information collected from the customer, sign it and send the form to the customer and ask the customer to sign it. This will require the client side application for customer to be able to read and write files on customer's local machine (i.e., read the file containing customer's signature file, insert it into the document and save it). Because the client side application for customer is implemented as an applet, to be granted access outside the normal applet sandbox, the applet needs to be developed and distributed as a signed applet using Java security features and the browser needs to be configured to allow such access privileges [9].

Finally the system needs to be tested with real users for all functional and non-functional requirements before it is adopted for commercial use.

Bibliography

1. CORBA: Architecture and Specification (OMG, 1997)
2. Alan Pope, The CORBA Reference Guide (Addison Wesley Longman, Inc., 1998)
3. Robert Orfali and Dan Harkey, Client/Server Programming With Java and CORBA (John Wiley and Sons, Inc., 1998)
4. James Gosling et al., Java Programming Language, Second Edition (Addison Wesley, 1998)
5. Orfali et al., The Essential Distributed Object Survival Guide (Wiley, 1996)
6. MySQL online manual: http://web.mysql.com/Manual_chapter
7. VisiBroker for Java online reference (include installation and programmer's guide): http://www.inprise.com/techpubs/books/vbj/vbj45/framesetindex.html
8. Rick Cattell et al., JDBC Database Access With Java (Addison Wesley, 1997)
9. Gary McGraw and Edward W. Felten, Securing Java: Getting Down to Business With Mobile Code (John Wiley and Sons, Inc., 1999)
10. Sun web site: http://www.javasoft.com
11. VisiBroker home page: http://www.inprise.com
12. OMG's web site: http://www.omg.com

Appendix A: User Manual

This appendix gives detailed instructions to setup and run the client/server application of the *Interactive Internet Banking System* (IIBS).

A.1 Setup and Run the Server

The following steps are involved to set up and run the server:

On server machine:

- Install Visibroker for Java 3.3/4.0. (Used for both CORBA support and Java virtual machine) [7][11].
- Install JDBC driver for MySQL3.23.11 [6].
- Set appropriate CLASSPATH environment variables for Java virtual machine, in my case, the client and server Java byte code are located in G:\comp793\project, and VisiBroker for Java 4.0 is installed in C:\Inprise\vbroker, set CLASSPATH as the following:

  > set

  CLASSPATH=%CLASSPATH% ;C\Inprise\vbroker\bin;G:\comp793\project\mySqlJ dbcDrv
- Start VisiBroker gatekeeper (used as web server and fire wall, it is not necessary if the client runs as standalone application)

  > Start gatekeeper
- Start Server

  Start IBServer

See **Figure A.1**

- To browse the bank database, click the database button (for demonstration purpose).

  See **Figure A.2**

A.2 Run the Clients

The system has two client applications: the *Client Side Application for Customer* (CSAC) and the *Client Side Application for Customer* (CSAR). The following section describes how to run these two client applications and their graphical user interface.

A.2.1 Run CSAC

To run CSAC, open the URL http://hostname:8088/IBClient.html in any Java applet enabled browser (for example, Internet Explorer or Netscape). The hostname is the server machine name (in my case, the server is running on *Spokane*. Open http://spokane:8088/IBClient.html in Internet Explore 5.0, the first page of login page appears as **Figure A.3**).

The following are the descriptions of the user interfaces for the CSAC:

- *Login page (***Figure A.3***):* Allow the customer to login into the system.
  1) *Account No box:* Allow the customer to enter the account number that uniquely identifies the customer.
  2) *Password box:* Allow the customer to enter the password associated with the account number.
  3) *Login button*: Click this button to login to the system. If successful, the *Main Menu page* appears as **Figure A.4**, otherwise an access denied message appears.

4) *Cancel button:* Click this button to dismiss the login page.

- *Main Menu page (***Figure A.4***):* Allow the customer access to all available bank services

  1) *Account Balance button:* Click this button to check account balance, see **Figure A.5** for *Account Balance page.*

  2) *View Transaction button:* Click this button to review the transactions incurred in the account. **Figure A.6** shows the *Transactions page.*

  3) *Pay Bill button:* Click this button to pay registered bills. **Figure A.7** shows the *Pay Bill page.*

  4) *Transfer Money button:* Allow the customer to transfer the money between different accounts.

  5) *Customer Service button:* Click this button to view the customer service information and interact with bank representatives. **Figure A.8** shows the *Customer Service Information page.*

  6) *Exit button:* Click this button to exit from the system

- *Account Balance page (***Figure A.5***):* Display the account balance

  1) *Account choice box:* Allow the customer to select the account type (check account, saving account).

  2) Balance edit box: Display the balance of the selected account.

  3) Back to main button: Click this button to go back to main menu.

- *Transactions page (***Figure A.6***):* Allow the customer to view the transactions occurred in selected accounts.

  1) *Account choice box:* Allow the user to select the account type (check account, saving account). The transactions are displayed in the edit box below the *Account choice box.*

  2) *Back to main button:* Click this to go back to main menu.

- *Pay Bill page (***Figure A.7***):* Allow the customer to view the bill accounts and pay bills. The left side *list box* lists all available bill payment services supported by the system. The user can select what bill to pay by clicking the list item; the system will display corresponding bill information on the right side *panel.* The amount to pay is automatically computed by the system, but the user can modify this by manually entering the amount. The *account choice box* allows the user to select the account from which the money is withdrawn. To pay the bill, just click the *Pay button.*
- *Customer Service Information page (***Figure A.8***):* This page gives the customer the overall customer service information and allow her to connect to a specific bank representative.
  1) *Message box:* Display service related messages sent from the server (for example, a new representative is ready for service, etc.).
  2) *Client demanding service box:* Display the total number of clients who are connected to bank representatives.
  3) *Representatives on service box:* Display the total number of representatives who are online for service.
  4) *Representative list box:* Display a list of representatives who are online for service. If the customer clicks the item in the list, the corresponding representative profile will be displayed on the right side panel. The profile includes the name, photo, phone, and the number of customers who are waiting for this representatives, etc.
  5) *Connect to This Rep button:* Click this button to connect to the representative who is currently selected in the list box, the *Client Service page* appears as **Figure A.9**.
  6) *Connect button:* Connect to the representative who has the shortest waiting list.
  7) *Back to Main button:* Go back to main menu.
  8) *Exit button:* Click this button to exit from the system.

- *Client Service page* (**Figure A.9**)*:* Allow the customer to interact with the bank representative.

  1) *Message box:* Display the service related messages sent from the server.
  2) *Representative profile panel:* Display the information of the representative with whom the customer is interacting with.
  3) *Start time box:* Display the time when the customer is connected to the representative.
  4) *Waiting list box:* Display the customer's order in the representative's waiting list.
  5) *Conversation record:* Display the conversations between the customer and the representative.
  6) *Document from representative:* Display the files sent from the representative.
  7) *Message edit box:* Allow the customer to type and edit the texts that are to be sent to the representative.
  8) *Sent button:* Click this button to send texts in *Message Edit box* to the representative
  9) *Clear button:* Click this button to clear the contents of the *Message Edit box*
  10) *Back button:* Click this button to go back to up level menu (*Customer Service Information page*).
  11) *Main button:* Click this button to go back to the *Main Menu page.*
  12) *Disconnect button:* Disconnect from the current connected representative.
  13) *Exit button:* Exit from the system.

A.2.2 Run CSAR

The CSAR is a standalone application. The following steps are involved to run this application.

In bank employee's machine:

- Installed JDK 2.0 or later version (for Java virtual machine support)
- Installed Java byte code for this application (*IBRep.class*)
- Run the byte code against the installed Java machine as the following:

  >Start Java IBRep

The login window appears as **Figure A.10**.

The CSAR user interfaces are described below:

- *Login window* (**Figure 10**): Allow the representative to login to the system.
  1) *Employee ID edit box:* Allow the representative enter the employee ID that uniquely identifies the representative.
  2) *Password edit box:* Allow the representative enter the password associated with the ID.
  3) *Login button*: Click this button to Login to the system. If successful, the main window with the title "Interactive Bank for Representatives" appears as **Figure A.11**, otherwise the system displays a dialog box with message "Access Denied".
  4) *Cancel button:* Click the button to abort the login operation.
- *Main Window* (**Figure A.11**): Allow the representative interact with the customer and perform requested transactions on behalf of the customer.
  1) *Transaction menu:* This menu has three submenus:
     (a) *Pay Bill:* Pay bill for the current customer.
     (b) *Transfer money:* Transfer money between accounts for the current customer.
     (c) *Deposit:* Makes a term deposit / GIC for the current customer (principle is taken from check/saving account). See **Figure A.12**.
  2) *View menu:* This menu has two submenus:
     (a) *Account balance:* Display the account balances of the current customer.
     (b) *Transaction:* Display account transactions of the current customer.

3) *Tools menu:* This menu has only one submenu:

Interest rate: Display bank's latest interest rate table.

4) *Message box:* Display service related messages sent from the server

5) *Waiting list box:* This list box displays a of list customers who are waiting for service.

6) *Client profile panel:* Display the information about the current customer (the first one in the waiting list).

7) *Start time box:* Display the time the representative starts to serve customers.

8) *The start time of current session box:* Display the time the representative starts to serve the current customer.

9) *Conversation Record box:* Display the conversations between the representative and the current customer.

10) *Documents sent to customer list box:* Display the list of files sent to the current customer.

11) *Message edit box:* Allow the representative to type and edit the texts that are to be sent to the customer.

12) Attached documents list box: Display the files to sent to the customer.

13) *Attach button:* Click this button to launch a stand file browser window. The user can select the files to be sent to the customer from this window.

14) *Sent button:* Click this button to send texts in *Message Edit box* to the customer.

15) *Clear button:* Click this button to clear the contents of the *Message Edit box*.

16) *Start button:* Click this button to start to serve the first customer in the waiting list. This button gives the representative full control of the service pace. That is, only when the representative wants to serve this customer does he press this button.

17) *Disconnect button:* Click this button to disconnect the current customer. The customer will be removed from the waiting list and the next one will become the first one.

18) *Exit button:* Click the button to exit from the system. If the representative has a waiting list that is not empty, the system will popup a dialog box with a warning message to let the user confirm before he actually exits. See **Figure A.13**.

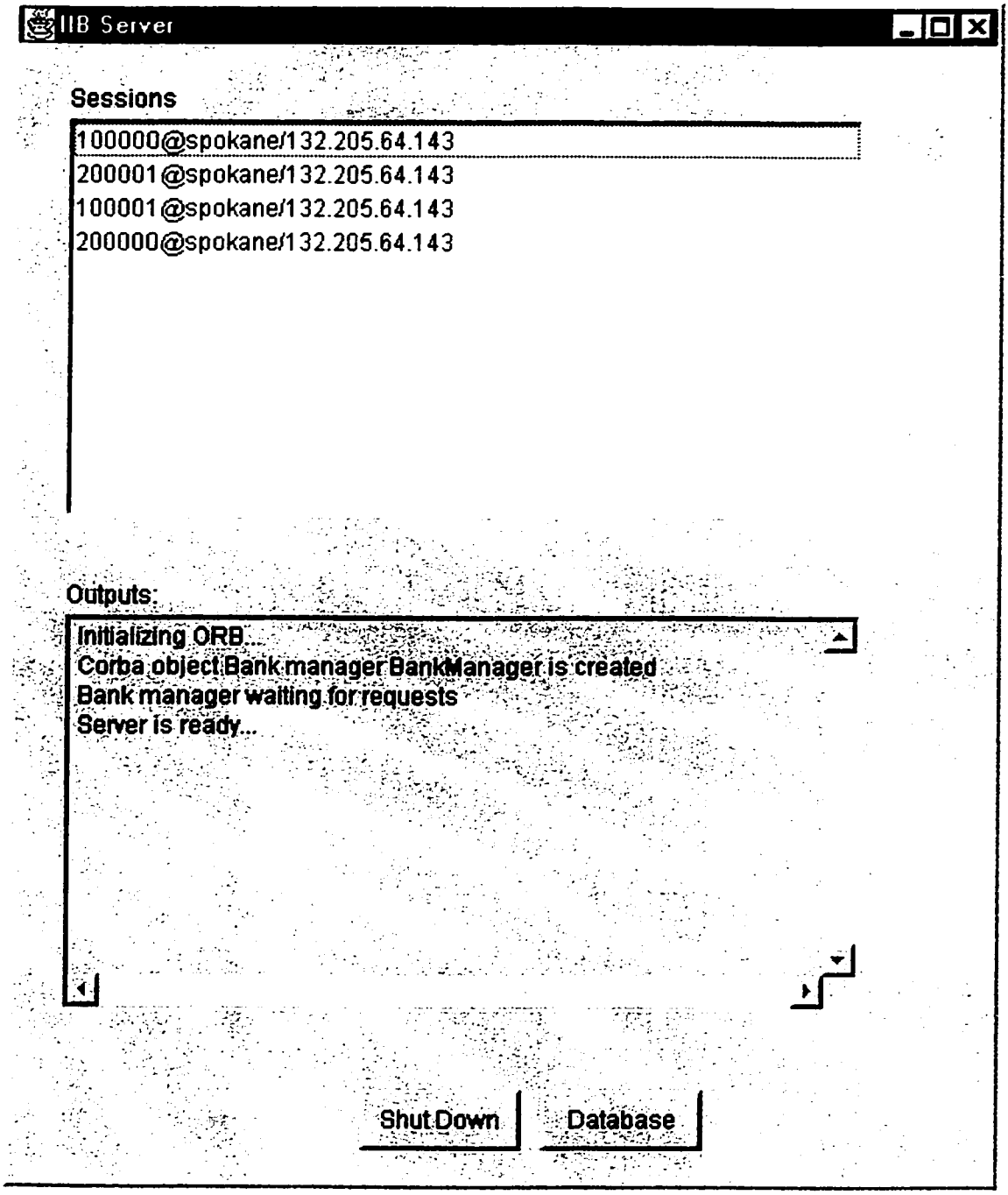

Figure A.1 Server UI

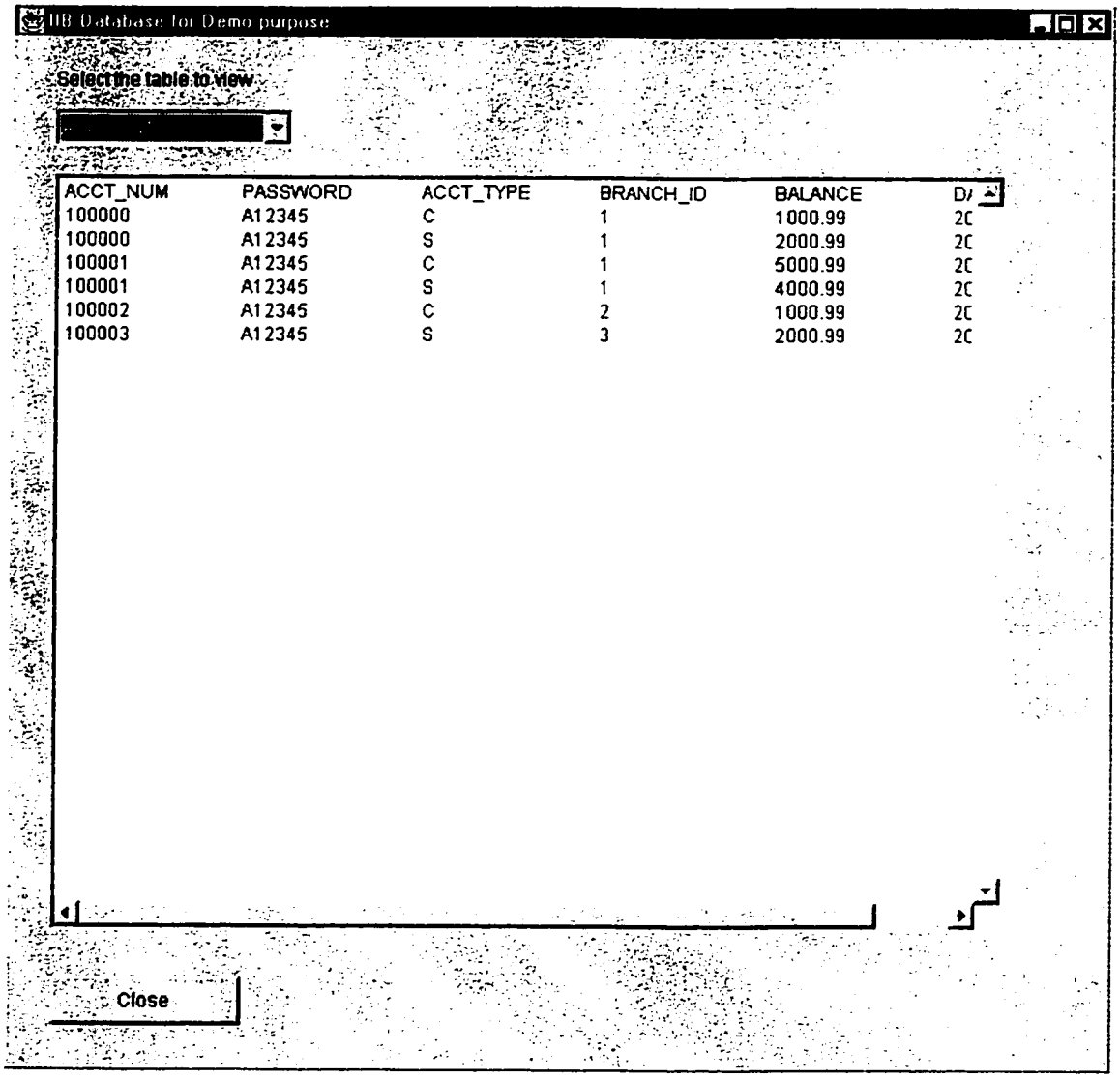

Figure A.2 User Interface for Database

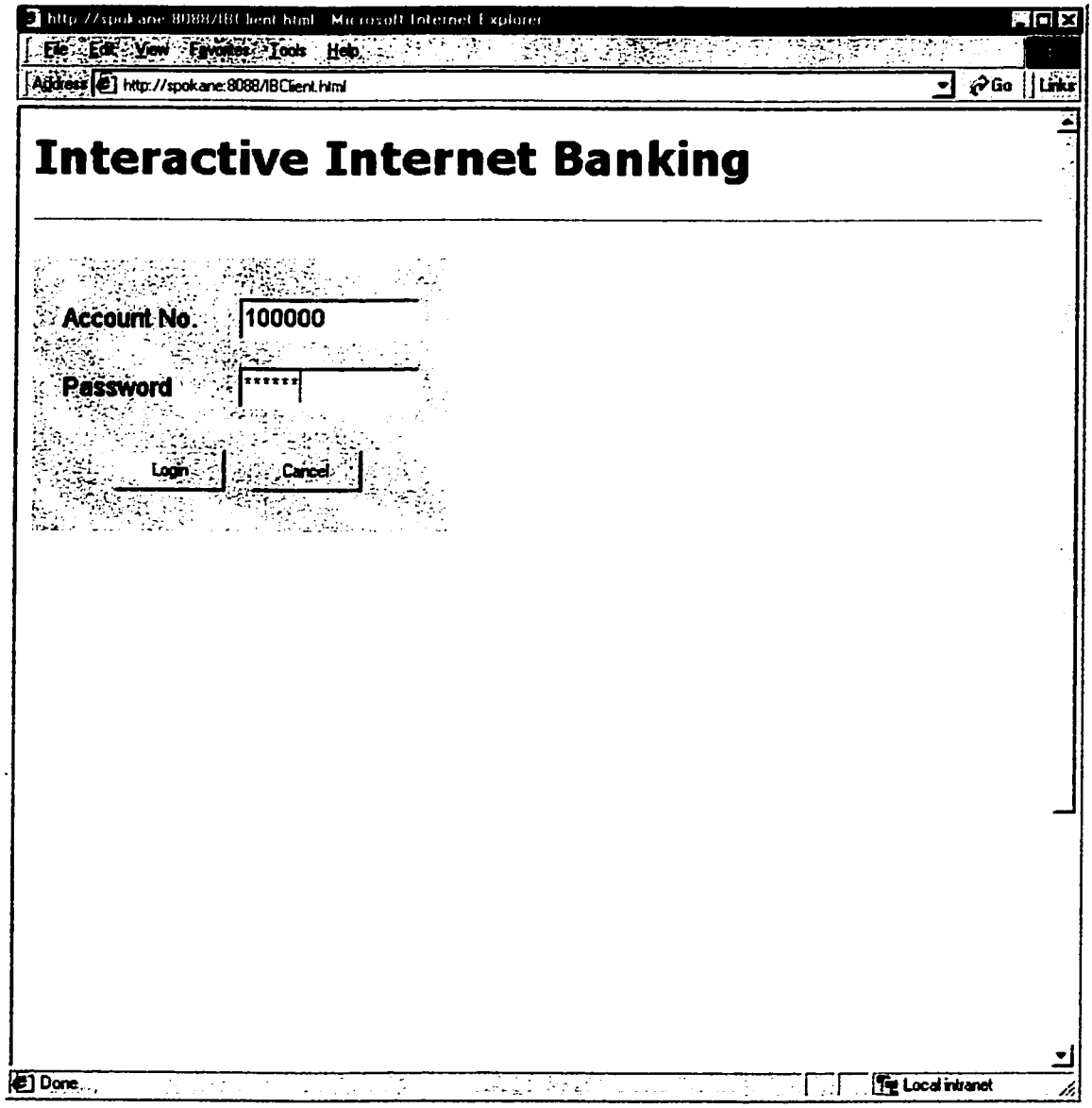

Figure A.3 Client Login

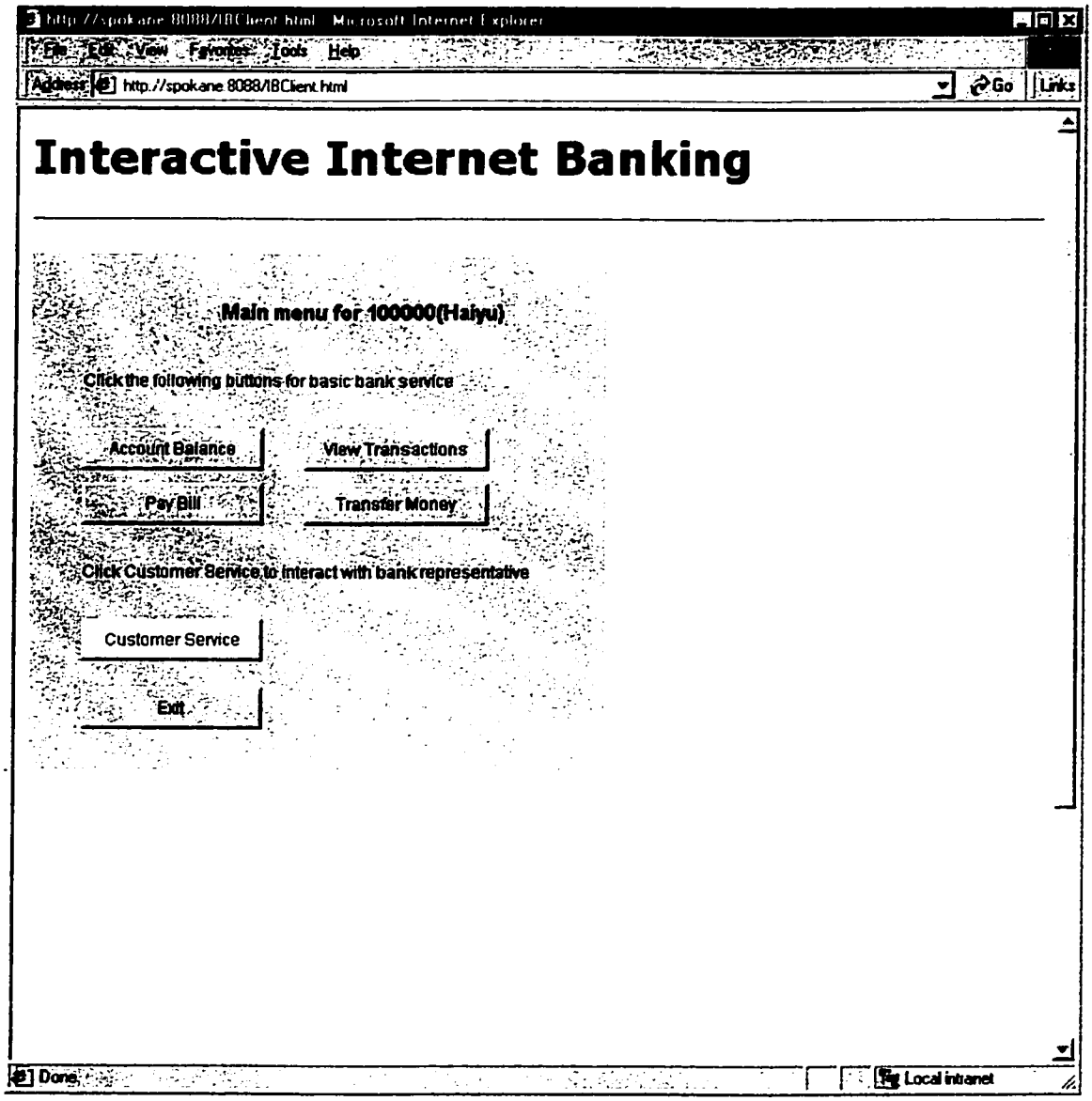

Figure A.4 Client Main Menu

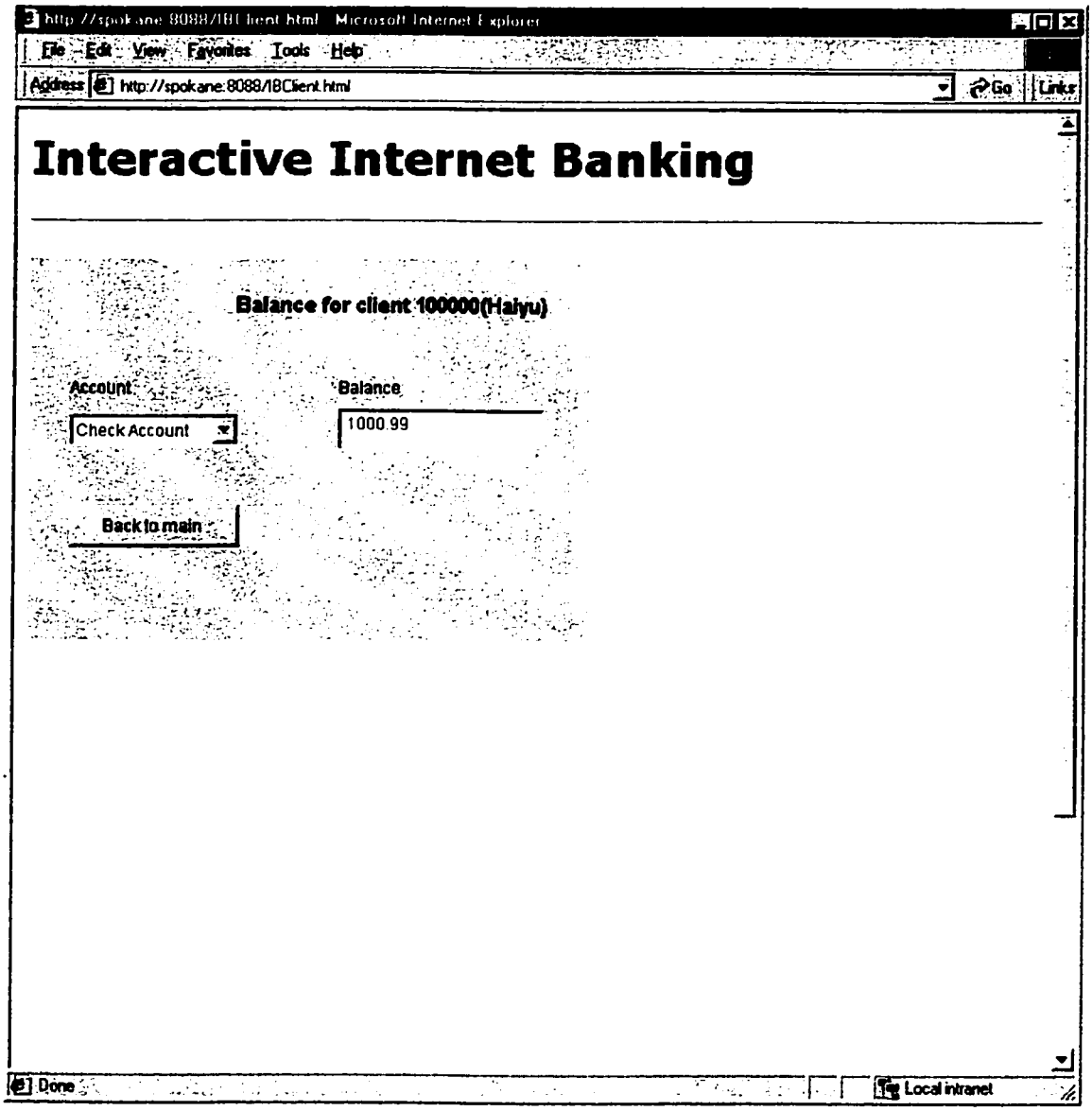

Figure A.5 Account Balance

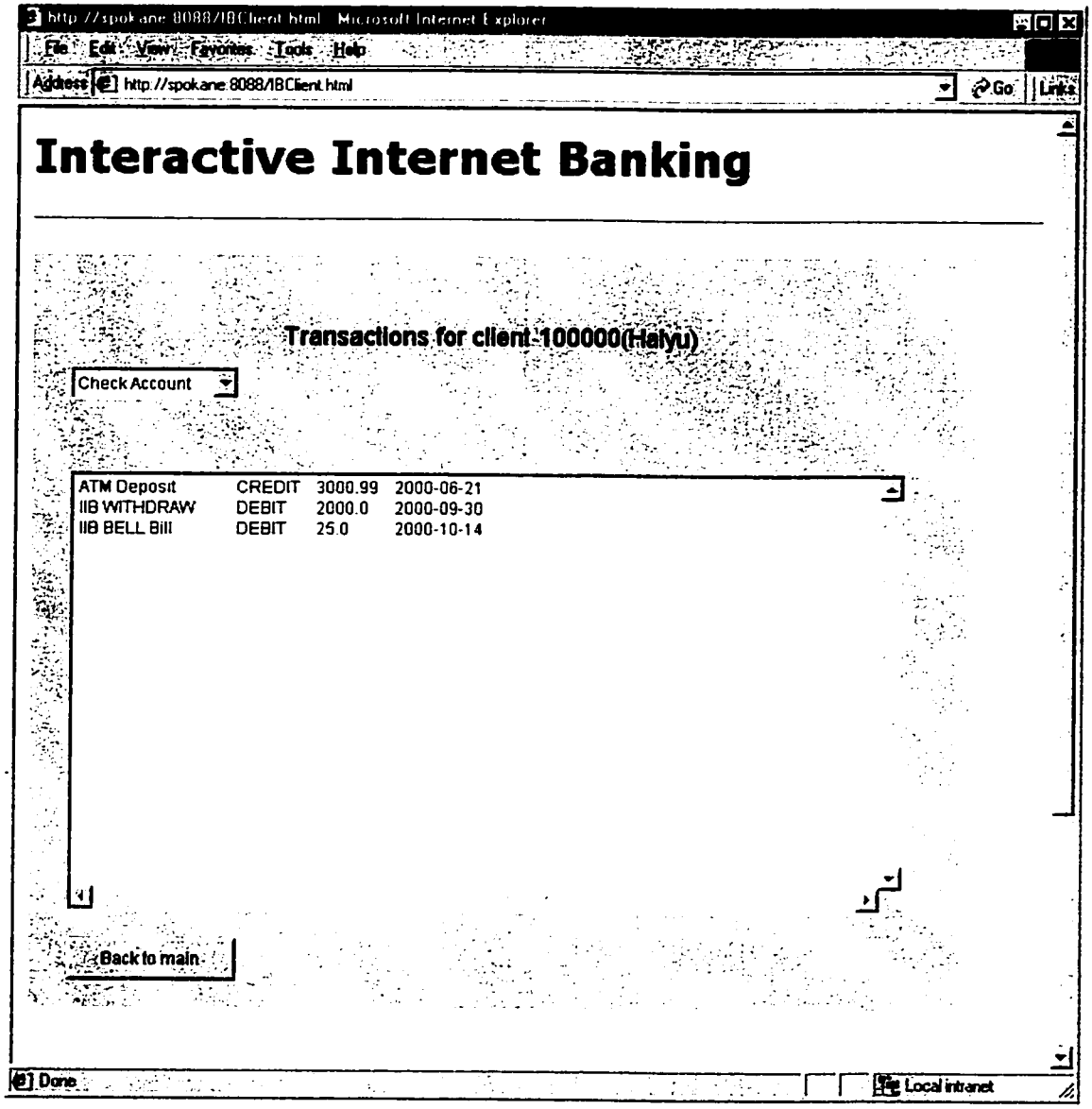

Figure A.6 Transactions

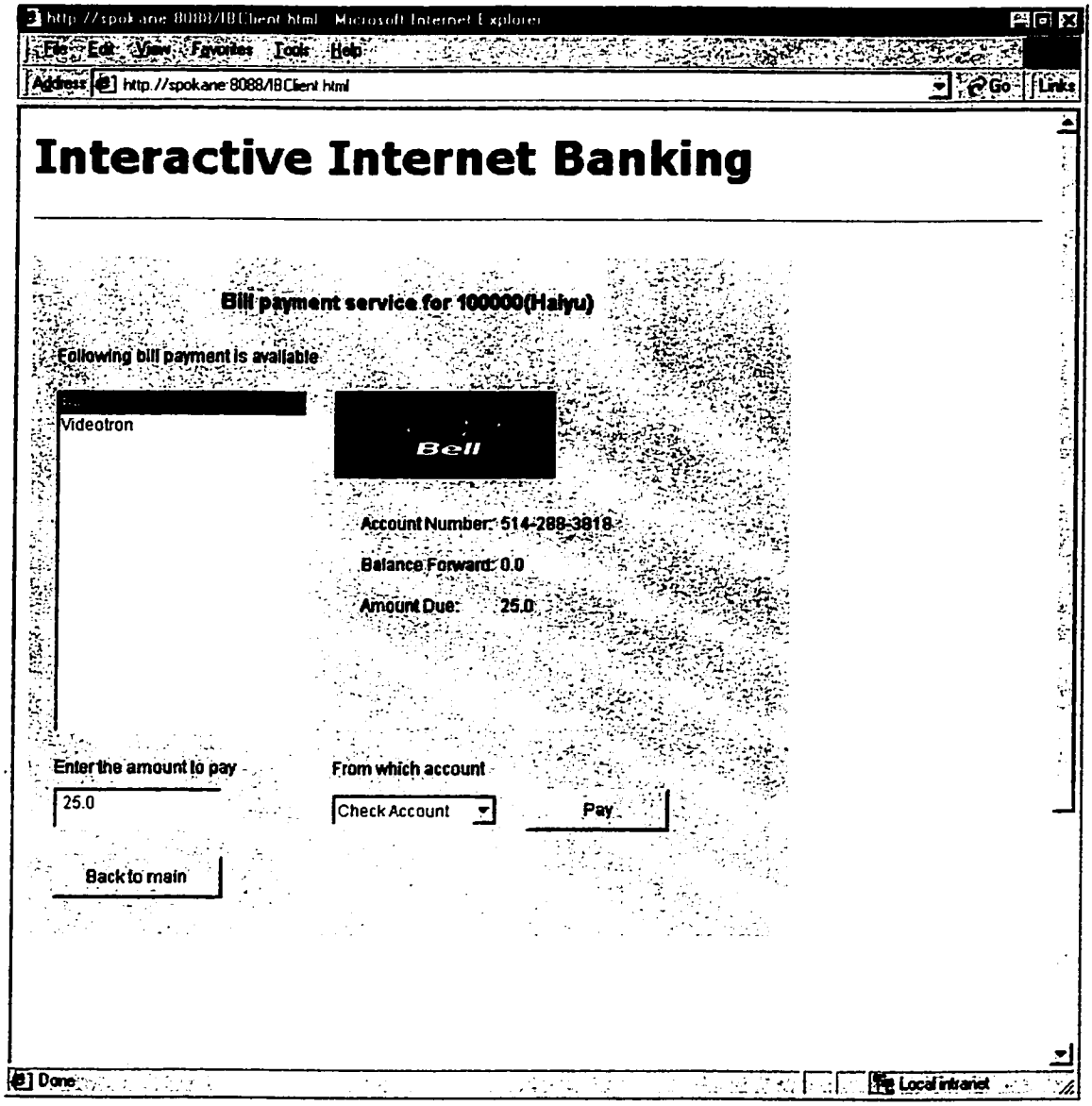

Figure A.7 Pay Bill

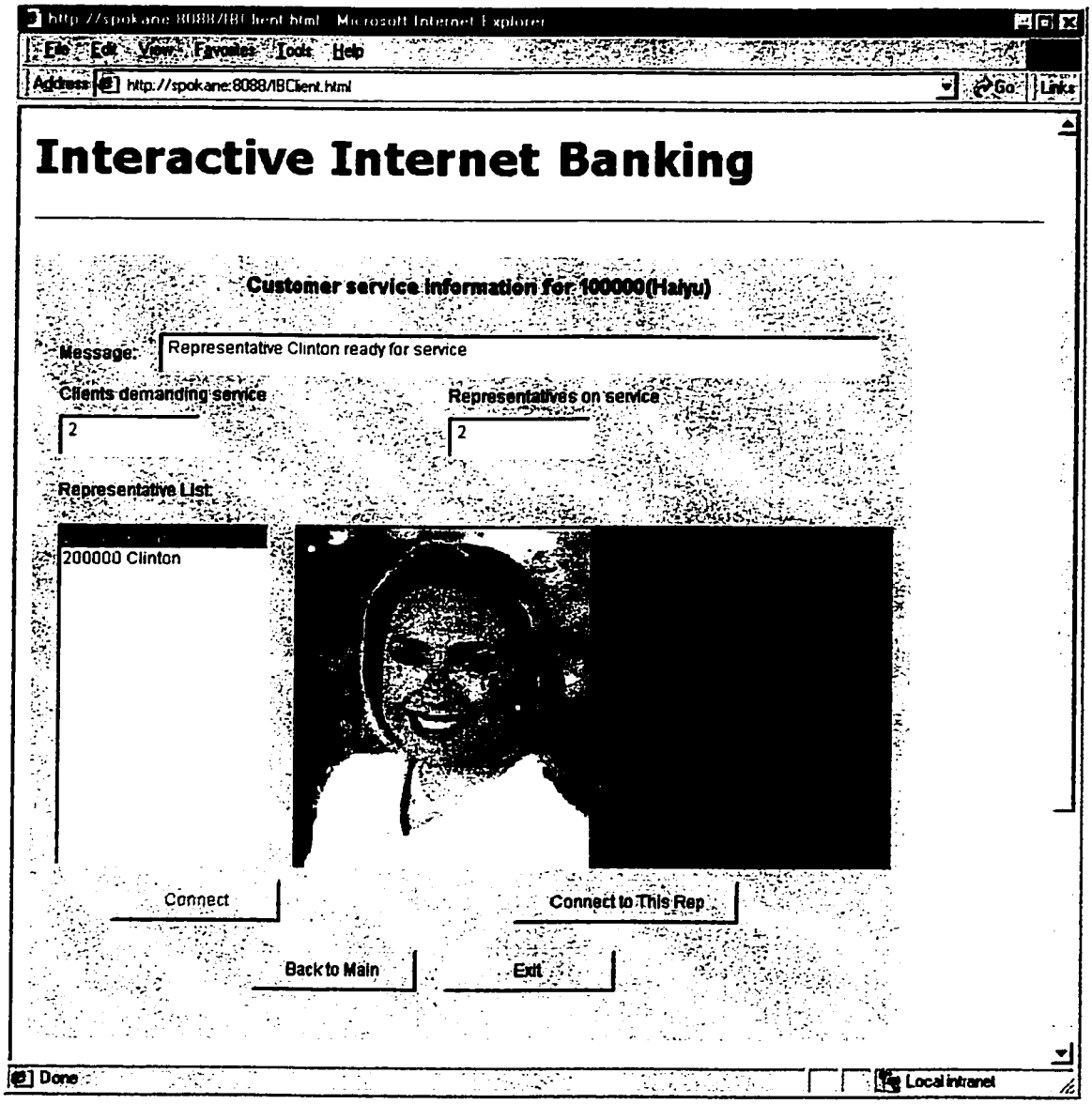

Figure A.8 Customer Service Information

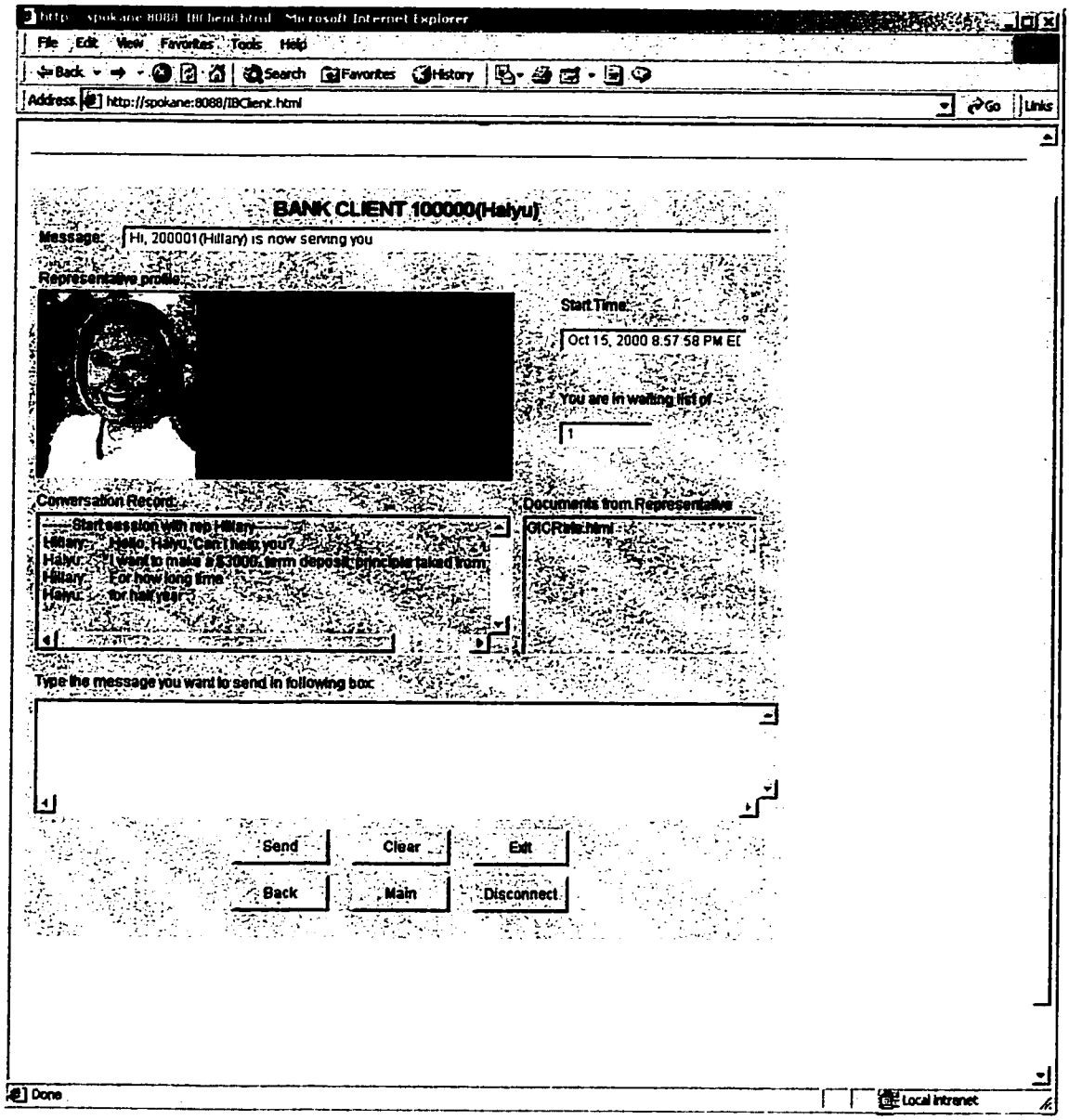

Figure A.9 Client Service

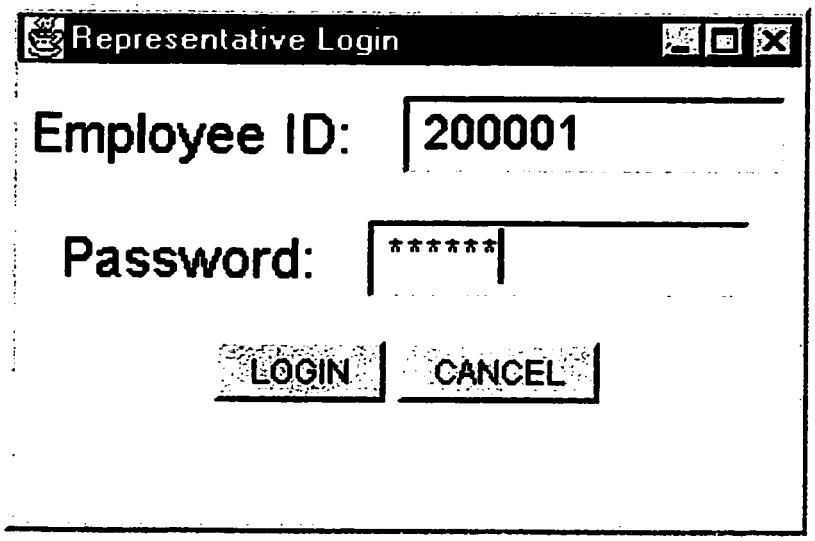

Figure A.10 Representative Login

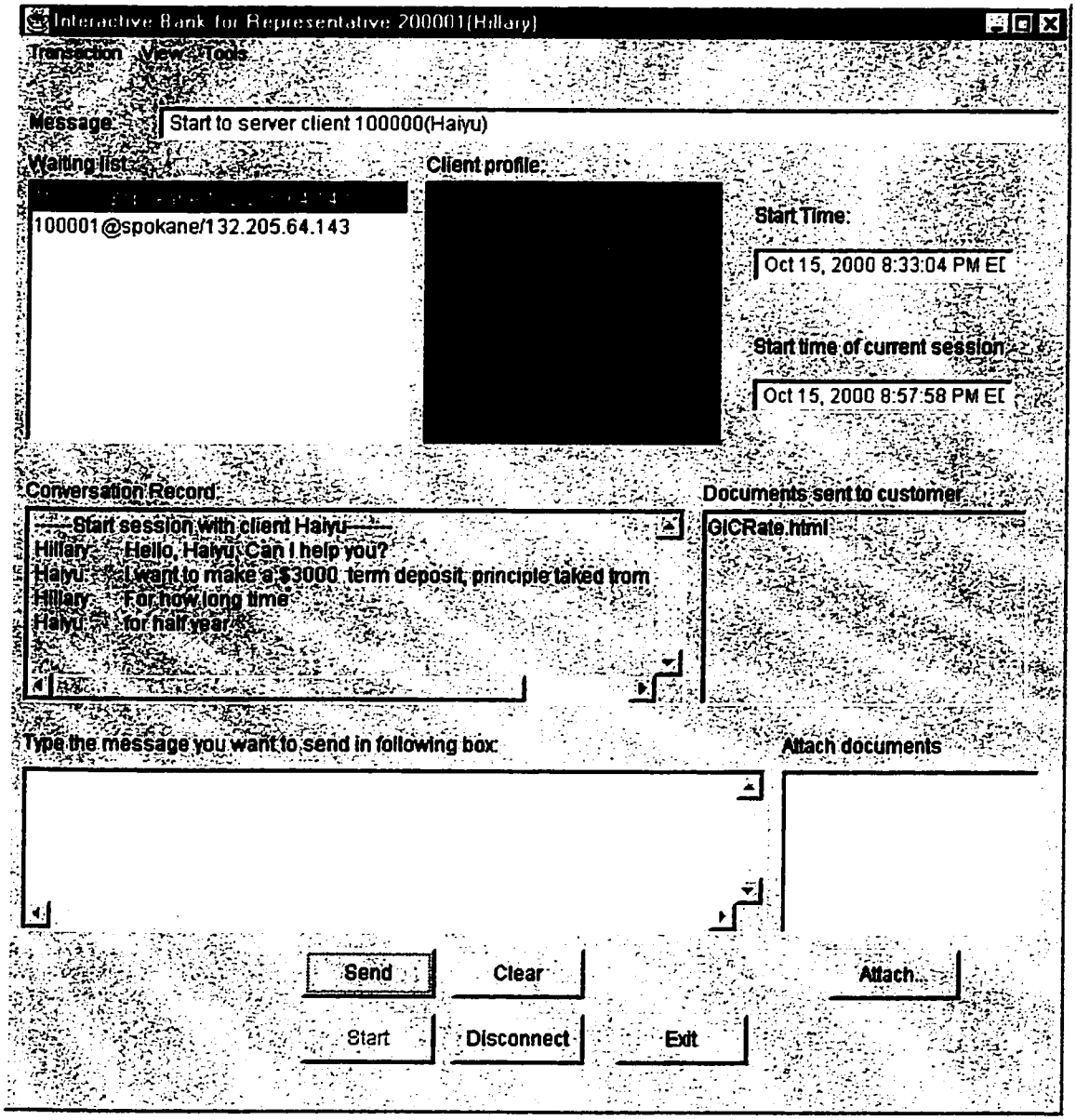

Figure A.11 Main Window for Representatives

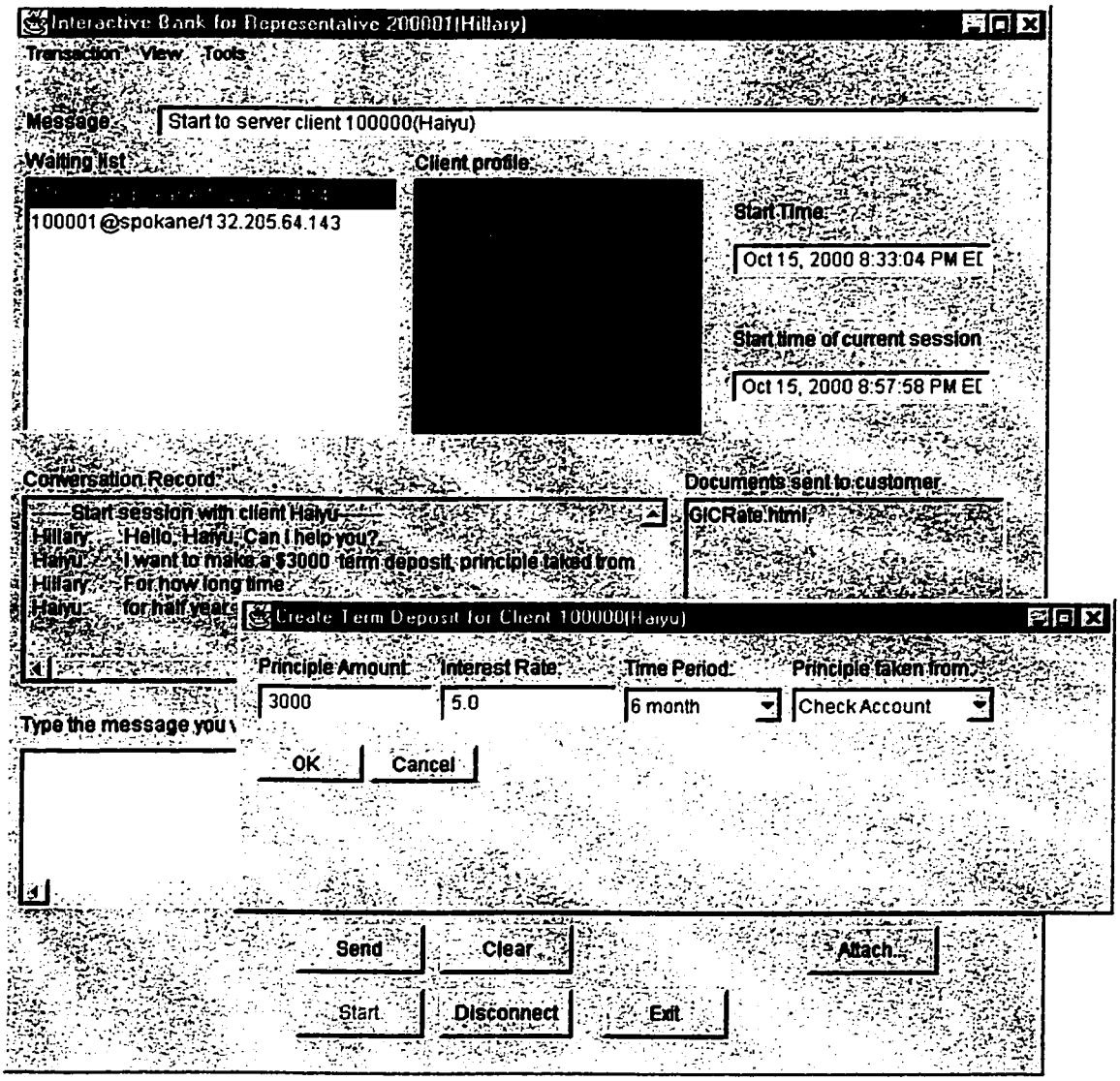

Figure A.12 Term Deposit

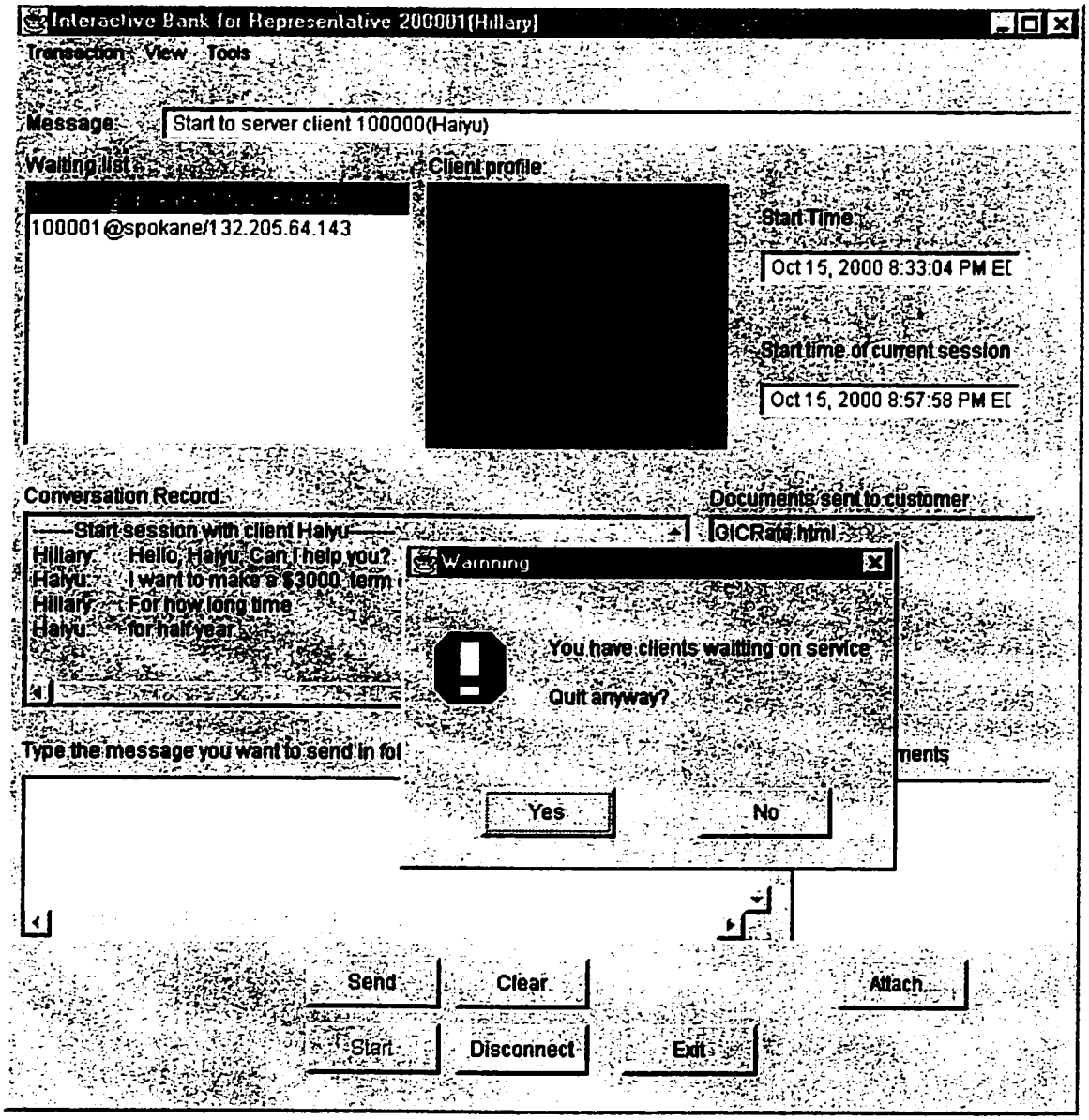

Figure A.13 Confirmation

Appendix B: Source Code

This appendix presents the Java source code for *Interactive Internet Banking System* (IIBS). The listed source code is not complete, it only contains the implementations of the major Java classes that are used to build the system, and the stub and skeleton code generated by IDL compiler is not included here. The whole system can be divided into three parts: *Client side application for customer* (CSAC), *Client side application for representative* (CSAR), and Server program. The source code will be presented in this order.

B.1 Source Code for CSAC

```
/*************************************************************************
 * File name:        IBClient.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB Client Applet(standalone program, if set fBrowser=False)
 * It dose the followings.
 * 1. Create and initialize UI widgets
 * 2. Initialize corba
 * 3. Locate server side Corba object Bank Manager, create
 *    client side callback object
 * 4. Implement client quit logic (tell server to remove it
 *    from its register list etc)
 *************************************************************************/

import java.applet.*;
import java.awt.*;
import java.net.*;

import org.omg.PortableServer.*;
import com.inprise.vbroker.PortableServerExt.BindSupportPolicyValue;
import com.inprise.vbroker.PortableServerExt.BindSupportPolicyValueHelper;

public class IBClient extends Applet
{  // Flag indicate if the IBClient is used in browser or a container frame
   public boolean fBrowser = false;
    // Client Login form
   public IBClientLogin ClientLogin;
   // Client Main Menu
   public IBClientMainMenu ClientMainMenu;
   // Servive information UI
   public IBClientServiceInfo ClientServiceInfo;
   // Client Service main user interface
   public IBClientService ClientService;
   // Client Bill payment
   public IBClientPayBill ClientPayBill;

   // Client account balance
   public IBClientBalance ClientBalance;
   // Client transactions
   public IBClientTransaction ClientTransaction;

   // Login fail messsage
   public MessagePanel Message;
   // Warning message bebore exit if client is on waiting list or being served
   public WarningPanel Warning;
   // Corba object reference of Server bank manager
   public Server.BankMgrCorba bankMgrCorba;
   // callback Corba object reference of client
   public IBClientCorbaImpl clientCorbaImpl;
   // callback object
   public Client.ClientCorba  clientCorba;
   // registered callback Corbar object index maintained by bank manager on server side
   public int ClientID;
   // more information of client
   public CClientInfo cClientInfo;
   // indicate client status
   public int status;
   // 0 mean client is not demanding service
   // 1 mean client is on representative waiting list
   // 2 mean client is being served by a rep

   public String lastUI;   // last active UI before exit warning dialog
                           // used to recover to origianl UI if user select
                           // No in exit warning message dialog
```

```
// indicate if the server is down
boolean fServerDown;

//private ClientCorbaThread clientCorbaThread = null;

// Initialize user interface
public void init()
{
    setLayout(null);  // absolute layout
    setBackground(Color.white);
    ClientMainMenu = new IBClientMainMenu(this);
    ClientLogin = new IBClientLogin(this);
    ClientServiceInfo = new IBClientServiceInfo(this);
    ClientService = new IBClientService(this);

    ClientPayBill = new IBClientPayBill();
    ClientPayBill.setIBClient(this);

    ClientBalance = new IBClientBalance();
    ClientBalance.setIBClient(this);
    ClientTransaction = new IBClientTransaction();
    ClientTransaction.setIBClient(this);

    Message = new MessagePanel(this);
    Warning = new WarningPanel(this);
    status = 0;
    lastUI = "";
    fServerDown = true;
    cClientInfo = null;
    ClientID = -1;

    // initialize corba
    initCorba();

    // Initialize the first page, which is login page
    ClientLogin.init();
}

public void destroy()
{
    System.out.println("destroy() is called");
    //System.out.println("status = " + status);
    // Warn client before quiting
    if((status == 1 || status == 2) && fBrowser == false ) {
        removeAll();
        setBackground(Color.white);
        if(status == 1) {
            Warning.setMessage("You are currently on waiting list");

        } else {
            Warning.setMessage("You are currently being served");
        }
        // bring up warning dialog waiting for user reponse
        Warning.init();
    }else {
        quit();
        System.exit(0);
    }
}

public void onQuit()
{
    //System.out.println("status = " + status);
    // Warn client before quiting
    if((status == 1 || status == 2)) {
        removeAll();
        setBackground(Color.white);
        if(status == 1) {
            Warning.setMessage("You are currently on waiting list");
        } else {
            Warning.setMessage("You are currently being served");
```

```
            }
            // bring up warning dialog waiting for user reponse
            Warning.init();
         }else {
            quit();
            if(fBrowser == true) {
               loadGoodbyePage();
            } else {
               System.exit(0);
            }
         }
      }

      public void quit()
      {
         if(bankMgrCorba != null && fServerDown == false && ClientID >= 0 ) {
            bankMgrCorba.clientQuit(ClientID);
         }
         // Clean up, set all references created on client side as null
         ClientLogin = null;
         ClientMainMenu = null;
         ClientServiceInfo = null;
         ClientService = null;
         ClientPayBill = null;
         clientCorba = null;

      }

      public void initCorba() {
         try
         {  // Initialize ORB

            // VERY IMPORTANT:
            // THE FIRST PARAMETER OF org.omg.CORBA.ORB.init()
            // MUST BE THE OBJECT REFERENCE OF APPLET, OTHERWISE, THE bind
            // METHOD WILL FAIL TO FIND THE SERVER CORBA OBJECT

            // BUT IF THIS APPLET IS USED IN CONTAINER FRAME, THE FIRST PARAMETER
            // MUST BE THE String[] args = null, OTHERWISE, org.omg.CORBA.ORB.init()
            // WILL FAIL.
            org.omg.CORBA.ORB orb;
            if(fBrowser == true)
            {
               orb = org.omg.CORBA.ORB.init(this, null);
            } else {
               String[] args = null;
               orb = org.omg.CORBA.ORB.init(args, null);
            }
            // resolve the bankMgrCorba object reference
            byte[] managerId = "BankManager".getBytes();
            if(fBrowser == true) {
               bankMgrCorba = Server.BankMgrCorbaHelper.bind(orb, "/bank_agent_poa",
managerId);
            } else {
               bankMgrCorba = Server.BankMgrCorbaHelper.bind(orb, "BankManager");
            }
            System.out.println("bank manager corba object BankManager is found ");

            try {
              // Activate it on the default POA which is root POA for this servant
               POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
               // Create policies for our persistent POA
               org.omg.CORBA.Any any = orb.create_any();
               BindSupportPolicyValueHelper.insert(any, BindSupportPolicyValue.BY_INSTANCE);
               org.omg.CORBA.Policy bsPolicy =
orb.create_policy(com.inprise.vbroker.PortableServerExt.BIND_SUPPORT_POLICY_TYPE.value,
any);
               org.omg.CORBA.Policy[] policies = {
                 rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT), bsPolicy
               };
               // Create myPOA with the right policies
```

```
            POA myPOA = rootPOA.create_POA( "bank_client_poa", rootPOA.the_POAManager(),
                                            policies );
            // Create the servant
            clientCorbaImpl = new IBClientCorbaImpl(this);

            // Decide on the ID for the servant
            byte[] BankClientId = "BankClient".getBytes();
            // Activate the servant with the ID on myPOA
            //myPOA.activate_object_with_id(BankClientId, clientCorbaImpl);
            myPOA.activate_object(clientCorbaImpl);
            rootPOA.the_POAManager().activate();
            System.out.println(myPOA.servant_to_reference(clientCorbaImpl) +
                        " is ready.");
            clientCorba =
Client.ClientCorbaHelper.narrow(myPOA.servant_to_reference(clientCorbaImpl));
         } catch (Exception e) {
           e.printStackTrace();
         }

         System.out.println("Create client side callback object clientCorba");
         fServerDown = false;
      } catch(Exception e)
      {
         System.out.println("Exception" + e);
      }
   }
   public void loadGoodbyePage(){
      try
      {
         // Clean up all object references
         ClientMainMenu = null;
         ClientLogin = null;
         ClientServiceInfo = null;
         ClientService = null;
         ClientPayBill = null;
         bankMgrCorba = null;
         clientCorba = null;

         removeAll();
         setBackground(Color.white);
         AppletContext context = getAppletContext();
         URL urlBase = getCodeBase();
         String urlStringBase = urlBase.toString();
         System.out.println(urlStringBase);
         urlStringBase = urlStringBase + "IBGoodBye.html";
         URL url = new URL(urlStringBase);
         context.showDocument(url);
      }
      catch(MalformedURLException ex) {
         System.out.println("Bad URL:");
      }
   }
}
```

```
/**************************************************************************
 * File name:        IBClientCorbaImpl.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     This is client side corba object implementation.
 * It serves as callback object when a client registers to the server,
 * afterwords the server communicates with the client through this callback object
 **************************************************************************/

public class IBClientCorbaImpl extends Client.ClientCorbaPOA
{
   public IBClient ibc;
   public IBClientCorbaImpl(Object client)
   {
      super();
      ibc = (IBClient)client;
   }
   // Client login
   public String getAcctNum()
   {
      return ibc.ClientLogin.getAcctNum();
   }

   public String getPassword()
   {
      return ibc.ClientLogin.getPassword();
   }

   public String getClientIP()
   {
      return ibc.ClientLogin.getIP();
   }
   // Client service info
   public void setNumClient(String s)
   {
      ibc.ClientServiceInfo.setNumClient(s);
   }
   public void setNumRep(String s)
   {
      ibc.ClientServiceInfo.setNumRep(s);
   }
   public void setMessage(String s){
      ibc.ClientServiceInfo.setMessage(s);
      ibc.ClientService.setMessage(s);
   }
   public void updateRepList(CRepInfo cRepInfo, int flag)
   {
      ibc.ClientServiceInfo.updateRepList(cRepInfo, flag);
   }
   public void setRepList(CRepInfo[] cRepInfoList)
   {
      ibc.ClientServiceInfo.setRepList(cRepInfoList);
   }

   // Client service
   public void removeConnectedRep(int repID) {
      ibc.ClientService.removeConnectedRep(repID);
   }
   public void addConversationRec(String s) {
      ibc.ClientService.addConversationRec(s);
   }
   public void addDocList(String s){
      ibc.ClientService.addDocList(s);
   }
   public void  updateStatus(int status){
      ibc.status = status;
      ibc.ClientService.updateStatus(status);
      ibc.ClientServiceInfo.updateStatus(status);
      ibc.ClientService.setStartTime(ibc.ClientService.getCurrentTime());
   }
   public int updateWaitListPos(int step){
```

```
        return ibc.ClientService.updateWaitListPos(step);
    }
    public void enterDisconnectState(){
        ibc.ClientService.enterDisconnectState();
    }
    public void  setServingRepCallback(Representative.RepCorba repObjRef){
        ibc.ClientService.setServingRepCallback(repObjRef);
    }
    public void serverDown(){
        ibc.fServerDown = true;
        if(ibc.fBrowser == true) {
            ibc.showStatus("Server is down");
        }
    }
}
```

```
/*********************************************************************
 * File name:        IBClientLogin.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     Implememenging client login
 * It dose the followings:
 * 1. Create and initialize UI widgets
 * 2. Register client callback with bank manager
 * 3. Load the client main menu if success, otherwise
 *    load the access deny warning message
 *********************************************************************/

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

public class IBClientLogin extends Panel
implements ActionListener, KeyListener
{
   // Login form variables
   private TextField tAcctNum;  // user name
   private TextField tPassword;  // password
   private Button bLogin;        // login
   private Button bCancel;        // back to last page
   private IBClient ibc;
   private String pwd;
   private String cfm;

   public IBClientLogin(IBClient ibc)
   {
      this.ibc = ibc;
      // Creat text fields and buttons
      tAcctNum = new TextField(12);
      tPassword = new TextField(12);
      tPassword.addKeyListener(this);
      bLogin = new Button("Login");
      bLogin.addActionListener(this);
      bCancel = new Button("Cancel");
      bCancel.addActionListener(this);
      pwd = "";
   }
   public void CreateComponent(){
   }
   public void init()
   {
      Color cBack = Color.lightGray;
      Color cFore = new Color(0,0,0);
      Font fInfo = new Font("SansSerif",Font.PLAIN,18);
      Font fTf = new Font("SansSerif",Font.PLAIN,18);

      // textfield for username and password
      tAcctNum.setFont(fTf);
      tPassword.setFont(fTf);
      tPassword.setEchoChar('*');

      // label for Account
      Label lUsr = new Label("Account No.");
      lUsr.setBackground(cBack);
      lUsr.setForeground(cFore);
      lUsr.setFont(fInfo);

      // label for password
      Label lPwd = new Label("Password");
      lPwd.setBackground(cBack);
      lPwd.setForeground(cFore);
      lPwd.setFont(fInfo);

      // Add panels to its parent applet
      ibc.add(this);
      setBackground(cBack);
      setLayout(null);
```

```
    //setBounds(150,160,300,200);
    setBounds(0,0,300,200);

    // add components to the control panel
    add(lUsr);
    add(lPwd);
    add(tAcctNum);
    add(tPassword);
    add(bLogin);
    add(bCancel);

    // relocate components
    lUsr.setBounds(20,30,120,30);
    tAcctNum.setBounds(150,30,130,30);
    lPwd.setBounds(20,80,120,30);
    tPassword.setBounds(150,80,130,30);
    bLogin.setBounds(60,140,80,30);
    bCancel.setBounds(160,140,80,30);
    tAcctNum.requestFocus();
    ibc.lastUI = "IBCLIENTLOGIN";

    // Force repaint
    if(ibc != null){
       ibc.setVisible(false);
       ibc.setVisible(true);
    }
}

public String getAcctNum()
{
   return tAcctNum.getText();
}

public String getPassword()
{
   return tPassword.getText();
}
public String getIP()
{
   String hostName = "Unknown";
   String hostName1 = "Unknown";
   try {
      //InetAddress inetAddress = InetAddress.getByName(null);
      InetAddress inetAddress = InetAddress.getLocalHost();
      hostName = inetAddress.toString();
      hostName1 = inetAddress.getHostName();

   }
   catch(UnknownHostException ex) {
      System.out.println("Unknown host");
   }
   return hostName;
}
public boolean register()
{
   boolean fSuccess = false;
   // Register clientCorba with bank magager
   int clientID = ibc.bankMgrCorba.registerClient(ibc.clientCorba);
   System.out.println(clientID);
   if(clientID < 0) //register not succeed
   {
      fSuccess  = false;
      System.out.println("Login Failed");

   } else if(clientID == 0) {
      fSuccess  = false;
      System.out.println("You already Login");
      ibc.removeAll();
      ibc.setBackground(Color.white);
      ibc.Message.setMessage("You already Login:");
```

```
            ibc.Message.init();
        } else {
            fSuccess  = true;
            // Setup client ID
            ibc.ClientID = clientID;
            // Setup more information
            ibc.cClientInfo = ibc.bankMgrCorba.getClientInfo(clientID);
            System.out.println("Login Success");
        }
        return fSuccess;
    }

    public void actionPerformed(ActionEvent e)
    {
        Object src = e.getSource();
        if(src == bLogin) {
            if(register()){
                ibc.removeAll();
                ibc.setBackground(Color.white);
                if(ibc.cClientInfo != null) {
                    ibc.ClientMainMenu.setTitle("Main menu for " + ibc.ClientID +
                                        "(" + ibc.cClientInfo.name + ")");
                }
                ibc.ClientMainMenu.init();
            } else {
                ibc.removeAll();
                ibc.setBackground(Color.white);
                ibc.Message.init();
            }
        }
        if(src == bCancel) {
            if(ibc.fBrowser == true) {
                ibc.loadGoodbyePage();
            }
        }
    }
    // handling Keystrokes
    public void keyTyped(KeyEvent e)
    {
        Object src = e.getSource();
        char key = e.getKeyChar();

        if (src == tPassword) {
            if(key>31 && key<127) {
                pwd = pwd + key;
            } else if(key==8) {
                int l = pwd.length();
                pwd = pwd.substring(0,l-1);
            }
        }
    }

    public void keyReleased(KeyEvent e) {
    }
    public void keyPressed(KeyEvent e) {
    }
}
```

```
/*********************************************************************
 * File name:        IBClientMainMenu.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB client main menu
 * It dose the followings
 * 1. Create and initialize UI widgets
 * 2. Load correspondent windows according to user selections
 *********************************************************************/

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

public class IBClientMainMenu extends Panel
implements ActionListener, MouseMotionListener
{
   // Client Main Menu variable
   private IBClient ibc;
   private Button bAccountBalance;
   private Button bViewTransaction;
   private Button bPayBill;
   private Button bTransferMoney;
   private Button bCustomerService;
   private Button bBack;
   private Button bExit;
   private Label lTitle;
   private Label lBasicBankService;
   private Label lCustomerService;

   private boolean fInitilized;

   public IBClientMainMenu(IBClient ibc){
      this.ibc = ibc;
      addMouseMotionListener(this);
      // Create menu buttons
      bAccountBalance = new Button("Account Balance");
      bAccountBalance.addActionListener(this);
      bAccountBalance.addMouseMotionListener(this);

      bViewTransaction = new Button("View Transactions");
      bViewTransaction.addActionListener(this);
      bViewTransaction.addMouseMotionListener(this);

      bPayBill = new Button("Pay Bill");
      bPayBill.addActionListener(this);
      bPayBill.addMouseMotionListener(this);

      bTransferMoney = new Button("Transfer Money");
      bTransferMoney.addActionListener(this);
      bTransferMoney.addMouseMotionListener(this);

      bCustomerService = new Button("Customer Service");
      bCustomerService.addActionListener(this);
      bCustomerService.addMouseMotionListener(this);

      bBack = new Button("Back");
      bBack.addActionListener(this);
      bBack.addMouseMotionListener(this);

      bExit = new Button("Exit");
      bExit.addMouseMotionListener(this);
      bExit.addActionListener(this);

      lTitle = new Label("Bank Client Main Menu",Label.CENTER);
      lBasicBankService = new Label("Click the following buttons for basic bank
service",Label.LEFT);
      lCustomerService = new Label("Click Customer Service to interact with bank
representative",Label.LEFT);
   }
   public void init()
   {
```

```
    int butHeight = 30;
    int butWidth = 130;
    int leftMargin = 40;
    int HorizDist = 20;
    int VertDist = 20;

    // Back ground color for labels and panel
    Color cBack = Color.lightGray;
    // Font for labels
    Font fTitle = new Font("SansSerif",Font.BOLD,15);
    Font fLabel = new Font("SansSerif",Font.PLAIN,12);

    // Set up title
    lTitle.setBackground(cBack);
    lTitle.setForeground(Color.black);
    lTitle.setFont(fTitle);
    // Set up label for basic bank service
    lBasicBankService.setBackground(cBack);
    lBasicBankService.setForeground(Color.black);
    lBasicBankService.setFont(fLabel);
    // Seu up lable for customer service
    lCustomerService.setBackground(cBack);
    lCustomerService.setForeground(Color.black);
    lCustomerService.setFont(fLabel);

    // Add panel to its parent applet
    ibc.add(this);
    setBackground(cBack);
    setLayout(null);
    //setBounds(150,60,420,400);
    setBounds(0,0,420,400);
    // add components
    add(lTitle);
    add(lBasicBankService);
    add(bAccountBalance);
    add(bViewTransaction);
    add(bPayBill);
    add(bTransferMoney);
    add(lCustomerService);
    add(bCustomerService);
    add(bBack);
    add(bExit);

    // relocate components
    lTitle.setBounds(leftMargin,30,400,30);
    lBasicBankService.setBounds(leftMargin,80,400,30);
    bAccountBalance.setBounds(leftMargin,130,butWidth,butHeight);
    bViewTransaction.setBounds(200,130,butWidth, butHeight);
    bPayBill.setBounds(leftMargin, 170, butWidth, butHeight);
    bTransferMoney.setBounds(200, 170, butWidth, butHeight);
    lCustomerService.setBounds(leftMargin,220,400,30);
    bCustomerService.setBounds(leftMargin, 270, butWidth, butHeight);
    bExit.setBounds(leftMargin, 320, butWidth, butHeight);
    ibc.lastUI = "IBCLIENTMAINMENU";
    // Force repainting
    if(ibc != null) {
       ibc.setVisible(false);
       ibc.setVisible(true);
    }
}
public void setTitle(String s){
    lTitle.setText(s);
}
public void mouseMoved(MouseEvent e) {
    Object src = e.getSource();
    if(src == bPayBill) {
       bPayBill.setBackground(Color.yellow);
    } else if(src == bAccountBalance){
       bAccountBalance.setBackground(Color.yellow);
       if(ibc.fBrowser == true) {
```

```
            ibc.showStatus("View your account balance");
         }
      } else if(src == bTransferMoney){
         bTransferMoney.setBackground(Color.yellow);
         if(ibc.fBrowser == true) {
            ibc.showStatus("Transfer money between your accounts");
         }
      } else if (src == bViewTransaction){
         bViewTransaction.setBackground(Color.yellow);
         if(ibc.fBrowser == true) {
            ibc.showStatus("View your recent tranactions");
         }
      } else if(src == bCustomerService) {
         bCustomerService.setBackground(Color.yellow);
         if(ibc.fBrowser == true) {
            ibc.showStatus("Interact to customer representative");
         }
      } else if(src == bExit) {
         bExit.setBackground(Color.yellow);
         if(ibc.fBrowser == true) {
            ibc.showStatus("Terminate the program");
         }
      } else if(src == bBack) {
         bBack.setBackground(Color.yellow);
         if(ibc.fBrowser == true) {
            ibc.showStatus("Back to login");
         }
      } else if(src == this) {
         bPayBill.setBackground(Color.lightGray);
         bAccountBalance.setBackground(Color.lightGray);
         bTransferMoney.setBackground(Color.lightGray);
         bViewTransaction.setBackground(Color.lightGray);
         bCustomerService.setBackground(Color.lightGray);
         bExit.setBackground(Color.lightGray);
         bBack.setBackground(Color.lightGray);
         if(ibc.fBrowser == true) {
            ibc.showStatus("Ready");
         }
      }
   }
public void mouseDragged(MouseEvent e) {
}
public void actionPerformed(ActionEvent e)
{
   Object src = e.getSource();
   if(src == bAccountBalance) {
      ibc.removeAll();
      ibc.setBackground(Color.white);
      if(ibc.cClientInfo != null) {
         ibc.ClientBalance.setTitle("Balance for client " +
                                    ibc.ClientID +
                                    "(" +
                                    ibc.cClientInfo.name +
                                    ")");
      }
      ibc.ClientBalance.init();
   }else if(src == bViewTransaction) {
      ibc.removeAll();
      ibc.setBackground(Color.white);
      if(ibc.cClientInfo != null) {
         ibc.ClientTransaction.setTitle("Transactions for client " +
                                    ibc.ClientID +
                                    "(" +
                                    ibc.cClientInfo.name +
                                    ")");
      }
      ibc.ClientTransaction.init();
   }else if(src == bPayBill) {
      ibc.removeAll();
      ibc.setBackground(Color.white);
      if(ibc.cClientInfo != null) {
```

```
            ibc.ClientPayBill.setTitle("Bill payment service for " +
                                       ibc.ClientID +
                                       "(" +
                                       ibc.cClientInfo.name +
                                       ")");
         }
         ibc.ClientPayBill.init();
      }else if(src == bTransferMoney) {

      }else if(src == bCustomerService) {
         ibc.removeAll();
         ibc.setBackground(Color.white);
         if(ibc.cClientInfo != null) {
            ibc.ClientServiceInfo.setTitle("Customer service information for " +
                                       ibc.ClientID +
                                       "(" +
                                       ibc.cClientInfo.name +
                                       ")");
         }
         ibc.ClientServiceInfo.init();
      }else if(src == bBack) {
         ibc.removeAll();
         ibc.setBackground(Color.white);
         ibc.add(ibc.ClientLogin);
      }else if (src == bExit) {
         ibc.onQuit();

      }
   }
}
```

```
/**************************************************************************
 * File name:        IBClientServiceInfo.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     Display customer service information
 * 1. Create and initialize UI widgets
 * 2. Display customer service information (number of representatives
 *    on service, number of clients demanding service etc, list of
 *    representatives etc)
 * 3. Load representative profile according to client selection
 * 4. Connect to the representative according to load balance or
 *    connect to specific representative according to client selection
 **************************************************************************/

import java.awt.*;
import java.awt.event.*;
import java.awt.Image;
import java.util.*;

public class IBClientServiceInfo extends IBClientServiceInfoUI
implements ActionListener,ItemListener
{
   // Information of registered representatives
   private Vector repInfoList;
   public String connectedRep;

   public IBClientServiceInfo(IBClient ibc)
   {
      super();
      this.ibc = ibc;
      repInfoList = new Vector();
      connectedRep = null;
   }
   public void init()
   {
      initUI();
      // register listener
      lstRepList.addItemListener(this);
      bConnect.addActionListener(this);
      bConnectToRep.addActionListener(this);
      bMainMenu.addActionListener(this);
      bExit.addActionListener(this);

      if(repInfoList.size()>0) {
         repInfoArea.setRepInfo((IBRepInfo)repInfoList.elementAt(0));
      } else {
         repInfoArea.setRepInfo(null);
         repInfoArea.repaint();
      }
      // By default the first one is selected
      lstRepList.select(0);
      updateStatus(ibc.status);
      // Force repainting
      if(ibc != null) {
         ibc.setVisible(false);
         ibc.setVisible(true);
      }
   }

   public Vector getRepInfoList()
   {
      return repInfoList;
   }

   public int getRepIndex(int RepID)
   {
      for(int i=0;i<repInfoList.size(); i++)
      {
         if(RepID == ((IBRepInfo)repInfoList.elementAt(i)).getID())// Found
         {
            return i;
```

```
        }
    }
    return -1;
}

public int getNumClientDemandService()
{
    int result = 0;
    for(int i=0; i<repInfoList.size(); i++)
    {
        IBRepInfo repInfo = (IBRepInfo)repInfoList.elementAt(i);
        if(repInfo != null )
        {
            result = result + repInfo.getNumClients();
        }
    }
    return result;
}

public int getNumRepOnService()
{
    return repInfoList.size();
}

// update representative information
public  void updateRepInfo(CRepInfo cRepInfo){
    if(cRepInfo != null) {
        int index = getRepIndex(cRepInfo.ID);
        // get it
        IBRepInfo repInfo = (IBRepInfo)repInfoList.elementAt(index);
        // changed to the latest one
        repInfo.setCRepInfo(cRepInfo);
        //put it back
        repInfoList.setElementAt(repInfo, index);
    } else {
        return;
    }
}

// Initialize the rep list
public void setRepList(CRepInfo[] cRepInfoList)
{
    int numberReps = 0;
    int numberClients = 0;

    for(int k=0; k<cRepInfoList.length; k++)
    {
        numberReps++;
        //numberClient
        IBRepInfo repInfo = new IBRepInfo();
        repInfo.setCRepInfo(cRepInfoList[k]);
        Image i = null;
        if(ibc.fBrowser) {
            i = ibc.getImage(ibc.getCodeBase(), "./images/" + repInfo.getPhotoFile());
        } else {
            Toolkit toolkit = Toolkit.getDefaultToolkit();
            i = toolkit.getImage("./images/" + repInfo.getPhotoFile());
        }
        repInfo.setPhoto(i);
        // update rep info list
        repInfoList.addElement(repInfo);
        // update list UI
        lstRepList.add(repInfo.getID() + " " + repInfo.getName());
    }

    // set the first rep as the current rep
    lstRepList.select(0);
    repInfoArea.setRepInfo((IBRepInfo)repInfoList.elementAt(0));
    repInfoArea.repaint();
    // number of representatives
    setNumRep(Integer.toString(numberReps));
```

```
   // number of clients
   numberClients = getNumClientDemandService();
   setNumClient(Integer.toString(numberClients));
}

public void updateRepList(CRepInfo cRepInfo, int flag)
{
   // first check if this rep exist
   if(flag == 1) { // Add this rep
      if(getRepIndex(cRepInfo.ID)== -1) // not exist
      {
         IBRepInfo repInfo = new IBRepInfo();
         repInfo.setCRepInfo(cRepInfo);
         Image i = null;
         if(ibc.fBrowser) {
            i = ibc.getImage(ibc.getCodeBase(), "./images/" + repInfo.getPhotoFile());
         } else {
            Toolkit toolkit = Toolkit.getDefaultToolkit();
            i = toolkit.getImage("./images/" + repInfo.getPhotoFile());
         }
         repInfo.setPhoto(i);
         // update rep info list
         repInfoList.addElement(repInfo);
         // update UI
         lstRepList.add(repInfo.getID() + " " + repInfo.getName());
         if(repInfoList.size()==1){
            // set the first rep as the current rep
            lstRepList.select(0);
            repInfoArea.setRepInfo(repInfo);
            repInfoArea.repaint();
            bConnect.setEnabled(true);
            bConnectToRep.setEnabled(true);
         }
      }else {//do nothing
         return;
      }
   } else if(flag == 0){ // remove this rep
      int index = getRepIndex(cRepInfo.ID);
      if(index >= 0)
      {
         // update rep info list
         repInfoList.removeElementAt(index);
         // update UI
         lstRepList.remove(cRepInfo.ID + " " + cRepInfo.name);
         // if this is current selected rep, select the first rep as current one
         if(repInfoArea.getRepInfo().getID() == cRepInfo.ID)
         {
            if(repInfoList.size()>0) {
               lstRepList.select(0);
               repInfoArea.setRepInfo((IBRepInfo)repInfoList.elementAt(0));
            } else {
               repInfoArea.setRepInfo(null);
            }
            repInfoArea.repaint();
         }
      }else {// do nothing
         return;
      }
   } else if(flag == 2) { // modified this rep
      updateRepInfo(cRepInfo);
      //check if this is the current selected rep and update IBServiceInfo UI
      String currItem = lstRepList.getSelectedItem();
      IBRepInfo repInfo = getIBRepInfoFromListItem(currItem);
      if(repInfo.getID() == cRepInfo.ID) {
         repInfo.setCRepInfo(cRepInfo);
         repInfoArea.setRepInfo(repInfo);
         repInfoArea.repaint();
      }
      //check if this is the currently connected rep and update IBService UI
      if(connectedRep != null) {
         repInfo = getIBRepInfoFromListItem(connectedRep);
```

```
            if(repInfo.getID() == cRepInfo.ID) {
               ibc.ClientService.updateRepProfile(cRepInfo);
            }
         }
      }
      // number of representatives
      setNumRep(Integer.toString(getNumRepOnService()));
      // number of clients
      setNumClient(Integer.toString(getNumClientDemandService()));
   }

   public IBRepInfo getIBRepInfoFromListItem(String item)
   {
      if(repInfoList.size() <= 0)
      {
         return null;
      }
      // compute the length of ID
      String strID = Integer.toString(((IBRepInfo)repInfoList.elementAt(0)).getID());
      int length = strID.length();
      // get the ID substring of item (Format= ID + space + name)
      String strRepID = item.substring(0, length);
      int repID = Integer.parseInt(strRepID);
      // now find the rep info
      for(int i=0; i< repInfoList.size(); i++)
      {
         IBRepInfo repInfo = (IBRepInfo)repInfoList.elementAt(i);
         if(repInfo.getID() == repID) {
            return repInfo;
         }
      }
      return null;
   }
   public void updateStatus(int status){

      if(status == 1 || status == 2) { //entering waiting list/being served
         bConnect.setEnabled(false);
         String item = lstRepList.getSelectedItem();
         if(connectedRep != null) {
            if(!item.equals(connectedRep)){
               bConnectToRep.setEnabled(false);
            } else {
               bConnectToRep.setEnabled(true);
            }
         }
      } else { // free to choose connection
         if(repInfoList.size()>0) {
            bConnect.setEnabled(true);
            bConnectToRep.setEnabled(true);
         } else {
            bConnect.setEnabled(false);
            bConnectToRep.setEnabled(false);
         }
      }
   }
   // handling for Item event from list
   public void itemStateChanged(ItemEvent e)
   {
      Object src = e.getSource();
      if(src == lstRepList) {
         String item = lstRepList.getSelectedItem();
         IBRepInfo repInfo = getIBRepInfoFromListItem(item);
         if(repInfo != null) {
            repInfoArea.setRepInfo(repInfo);
            repInfoArea.repaint();
         }
         if(connectedRep != null) {
            if(!item.equals(connectedRep)){
               bConnectToRep.setEnabled(false);
            } else {
               bConnectToRep.setEnabled(true);
```

```
            }
         }
      }
   }

   public void actionPerformed(ActionEvent e)
   {
      Object src = e.getSource();
      if(src == bConnect) {
         ibc.removeAll();
         ibc.setBackground(Color.white);
         ibc.ClientService.init();
      } else if(src == bConnectToRep) {

         if(connectedRep == null) {// not connected yet
            String currItem = lstRepList.getSelectedItem();
            connectedRep = currItem;
            IBRepInfo repInfo = getIBRepInfoFromListItem(currItem);
            // rep profile
            ibc.ClientService.setRepProfile(repInfo);
            ibc.ClientService.setMessage("You are connected to rep " +
repInfo.getName());
            // Inform server
            ibc.bankMgrCorba.connectToRep(ibc.ClientID, repInfo.getID());

            // now this client is on waiting status
            ibc.status = 1;
            // disable connect buttons
            //bConnectToRep.setEnabled(false);
            bConnect.setEnabled(false);
            // set IBService UI waiting list order
            repInfo = getIBRepInfoFromListItem(currItem);
            ibc.removeAll();
            ibc.setBackground(Color.white);

   ibc.ClientService.setNumber(Integer.toString(repInfo.getCRepInfo().numClients));
            ibc.ClientService.setTitle("BANK CLIENT " + ibc.ClientID + "(" +
ibc.cClientInfo.name + ")");
            ibc.ClientService.init();
         } else {
            ibc.removeAll();
            ibc.setBackground(Color.white);
            ibc.setVisible(false);
            ibc.ClientService.setTitle("BANK CLIENT " + ibc.ClientID + "(" +
ibc.cClientInfo.name + ")");
            ibc.ClientService.init();
         }
      } else if(src == bMainMenu) {
         ibc.removeAll();
         ibc.setBackground(Color.white);
         ibc.ClientMainMenu.init();
      } else if(src == bExit) {
         ibc.onQuit();
      }
   }

}

class IBClientServiceInfoUI extends Panel
{
   public java.awt.List lstRepList;
   // grand Parent
   protected IBClient ibc;

   protected TextField tMessage;
   protected RepInfoArea repInfoArea;
   protected TextField tNumClient;
   protected TextField tNumRep;

   protected Button bConnect;
   protected Button bConnectToRep;
```

```
protected Button bMainMenu;
protected Button bExit;
protected Label lTitle;

public IBClientServiceInfoUI(){

   // Create List, TextField and Canvas
   lstRepList = new java.awt.List();

   tMessage = new TextField(70);
   repInfoArea = new RepInfoArea();

   tNumClient = new TextField(20);
   tNumRep = new TextField(20);

   bConnect = new  Button("Connect");
   bConnectToRep = new Button("Connect to This Rep");
   bMainMenu = new Button("Back to Main");
   bExit = new Button("Exit");

   lTitle = new Label("Customer Service Information ", Label.CENTER);
}

public void initUI()
{
   //Color cBack = new Color(200,200,200);
   Color cBack = Color.lightGray;
   Color cFore = new Color(0,0,0);
   Font fTitle = new Font("SansSerif",Font.BOLD,15);
   Font fInfo = new Font("SansSerif",Font.PLAIN,12);

   // Add panel to its parent applet
   ibc.add(this);
   setBackground(cBack);
   setLayout(null);
   setBounds(0,0,650,650);

   // lable for title
   lTitle.setBackground(cBack);
   lTitle.setForeground(cFore);
   lTitle.setFont(fTitle);
   // label for message
   Label lMessage = new Label("Message:");
   lMessage.setBackground(cBack);
   lMessage.setForeground(cFore);
   lMessage.setFont(fInfo);

   // Lable for num of client
   Label lNumClient = new Label("Clients demanding service");
   lNumClient.setBackground(cBack);
   lNumClient.setForeground(cFore);
   lNumClient.setFont(fInfo);

   // Lable for num of rep
   Label lNumRep = new Label("Representatives on service");
   lNumRep.setBackground(cBack);
   lNumRep.setForeground(cFore);
   lNumRep.setFont(fInfo);
   // Label for list
   Label lSLst = new Label("Representative List:");
   lSLst.setBackground(cBack);
   lSLst.setForeground(cFore);
   lSLst.setFont(fInfo);
   repInfoArea.setBackground(Color.gray);

   // add component to panel
   add(lSLst);
   add(lstRepList);
   add(lMessage);
   add(tMessage);
   add(repInfoArea);
```

```
        add(lTitle);
        add(lNumClient);
        add(lNumRep);
        add(tNumClient);
        add(tNumRep);
        add(bConnect);
        add(bConnectToRep);
        add(bMainMenu);
        add(bExit);

        // Relocate components
        lTitle.setBounds(20,10,600,30);
        lMessage.setBounds(20, 60, 60, 30);
        tMessage.setBounds(90, 60, 520, 30);
        lNumClient.setBounds(20,90,200,30);
        lNumRep.setBounds(300,90,200,30);
        tNumClient.setBounds(20,120,100,30);
        tNumRep.setBounds(300,120,100,30);
        lSLst.setBounds(20,160,150,30);
        lstRepList.setBounds(20, 200, 150, 250);
        bConnect.setBounds(60, 460, 120, 30);
        repInfoArea.setBounds(190, 200, 430, 250);
        bConnectToRep.setBounds(350, 460, 160, 30);

        bMainMenu.setBounds(160, 510, 120, 30);
        bExit.setBounds(300, 510, 120, 30);

        ibc.lastUI = "IBCLIENTSERVICEINFO";
    }

    public void setTitle(String s){
        lTitle.setText(s);
    }

    public void setNumClient(String s)
    {
        tNumClient.setText(s);
    }

    public void setNumRep(String s)
    {
        tNumRep.setText(s);
    }

    public void setMessage(String s)
    {
        tMessage.setText(s);
    }
    public void enterConnectState(){

    }
    public void enterDisconnectState(){

    }
}

class RepInfoArea extends Canvas
{
    private IBRepInfo repInfo; // Current selected representative
    private Font fRep1;
    private Font fRep2;

    public RepInfoArea()
    {
        super();
        repInfo = null;
        fRep1 = new  Font("SansSerif",Font.ITALIC,12);
        fRep2 = new  Font("SansSerif",Font.PLAIN,12);
    }
    public IBRepInfo getRepInfo()
    {
```

```
        return repInfo;
    }
    public void setRepInfo(IBRepInfo repInfo)
    {
        this.repInfo = repInfo;
    }
    public void paint(Graphics g) {
        if(repInfo == null )
        {
            g.drawString("Representative profile is not available", 20, 20);
        } else {
            int photoWidth = 250;
            Image photo = repInfo.getPhoto();
            if(photo != null) {
                int w = photo.getWidth(this);
                int h = photo.getHeight(this);
                float ratio =  (float) w/(float)h;
                photoWidth = (int) (ratio * getSize().height);
                g.drawImage(photo, 0, 0, photoWidth, getSize().height, this);
            }
            g.setFont(fRep1);
            g.drawString("Name", photoWidth + 10, 20);
            g.setFont(fRep2);
            g.drawString(repInfo.getName(), photoWidth + 110, 20);
            g.setFont(fRep1);
            g.drawString("Branch ID", photoWidth + 10, 50);
            g.setFont(fRep2);
            g.drawString(Integer.toString(repInfo.getBranchID()), photoWidth + 110, 50);

            g.setFont(fRep1);
            g.drawString("Telephone", photoWidth + 10, 80);
            g.setFont(fRep2);
            g.drawString(repInfo.getTelephone(), photoWidth + 110, 80);

            g.setFont(fRep1);
            g.drawString("Number of Clients", photoWidth + 10, 110);
            g.setFont(fRep2);
            g.drawString(Integer.toString(repInfo.getNumClients()), photoWidth + 160, 110);
        }

        Rectangle rect = this.getBounds();
        g.draw3DRect(0, 0, rect.width, rect.height, true);
    }
}
```

```
/*************************************************************************
 * File name:        IBClientService.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB client side main interface for customer service
 *                   (client/representative interaction)
 * It dose the followings
 * 1. Create and initialize UI widgets
 * 2. Load the representative profile that the client is waitting for
 * 3. Implement client/representative interaction logic, connect,
 *    disconnect, send message, exit etc
 *************************************************************************/

import java.awt.*;
import java.awt.event.*;
import java.text.*;
import java.util.*;
import java.applet.*;
import java.net.*;

public class IBClientService extends IBClientServiceUI
implements ActionListener, ItemListener
{
   IBRepInfo   lastConnectedRep;
   boolean fInitCalled;
   private  Representative.RepCorba repCorba;//currently serving rep callback

   public IBClientService(IBClient ibc){
      super();
      this.ibc = ibc;
      lastConnectedRep = null;
      repCorba = null;
      fInitCalled = false;
   }
   public void init(){

      initUI();
      // listeners
      if(fInitCalled==false) {
         bSend.addActionListener(this);
         bClear.addActionListener(this);
         bExit.addActionListener(this);
         bBack.addActionListener(this);
         bMain.addActionListener(this);
         bConnect.addActionListener(this);
         bDisconnect.addActionListener(this);
         lstDocList.addItemListener(this);
      }
      ibc.lastUI = "IBCLIENTSERVICE";
      updateStatus(ibc.status);
      fInitCalled = true;
      // Force repainting
      if(ibc != null) {
         ibc.setVisible(false);
         ibc.setVisible(true);
      }
   }
   public void setRepProfile(IBRepInfo repInfo)
   {
      repProfile.setRepInfo(repInfo);
   }
   public void setServingRepCallback(Representative.RepCorba repCorba){
      this.repCorba = repCorba;
   }
   // Called when this client's servering rep exit
   public void removeConnectedRep(int repID){
      if(repID == repProfile.getRepInfo().getID()) {
         lastConnectedRep = null;
         repCorba = null;
         ibc.ClientServiceInfo.connectedRep = null;
         ibc.status = 0;
```

```
            setNumber("");
            setMessage("rep " + repID + " " + repProfile.getRepInfo().getName() + " has
quit");
            repProfile.setRepInfo(null);
            repProfile.repaint();
         }
      }
      public String getCurrentTime(){
         String currTime;
         String language = "en";
         String country = "CANADA";
         String timezone = "EST";
         //set locale and timezone
         GregorianCalendar cal = new GregorianCalendar();
         Locale locale = new Locale (language, country);
         TimeZone tz = TimeZone.getTimeZone(timezone);
         /*set display format in specified style, locale, and timezone*/
         DateFormat myFormat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
DateFormat.LONG,locale);
         myFormat.setTimeZone(tz);
         currTime  =  myFormat.format(cal.getTime());
         return currTime;
      }
      public void updateStatus(int status){
         if(status == 0) {
            enterDisconnectState();
         } else if(status == 1) {
            enterConnectState();
         } else if(status == 2) {
            enterServeState();
         }
      }
      public void enterDisconnectState() {
         // clear start time and waiting list order
         ibc.status = 0;
         setStartTime("");
         setNumber("");
         bDisconnect.setEnabled(false);

         bSend.setEnabled(false);
         bClear.setEnabled(false);
         lConversationRec.setEnabled(false);
         aConversationRec.setEnabled(false);
         lDocList.setEnabled(false);
         lstDocList.setEnabled(false);
         lMsgToSend.setEnabled(false);
         aMsgToSend.setEnabled(false);
         ibc.ClientServiceInfo.connectedRep = null;
         repCorba = null;
         if(repProfile != null) {
            repProfile.setRepInfo(null);
            repProfile.repaint();
         }
      }
      public void enterConnectState() {
         // now this client is on waiting status
         ibc.status = 1;
         // disable connect button and enable disconnect button
         bDisconnect.setEnabled(true);
         bConnect.setEnabled(false);
         bSend.setEnabled(false);
         bClear.setEnabled(false);
         lConversationRec.setEnabled(false);
         aConversationRec.setEnabled(false);
         lDocList.setEnabled(false);
         lstDocList.setEnabled(false);
         lMsgToSend.setEnabled(false);
         aMsgToSend.setEnabled(false);
      }
      public void enterServeState(){
         // now this client is being served by rep
```

```
        ibc.status = 2;
        // disable connect button and enable disconnect button
        bDisconnect.setEnabled(true);
        bConnect.setEnabled(false);
        //enable all other UI element
        bSend.setEnabled(true);
        bClear.setEnabled(true);
        lConversationRec.setEnabled(true);
        aConversationRec.setEnabled(true);
        lDocList.setEnabled(true);
        lstDocList.setEnabled(true);
        lMsgToSend.setEnabled(true);
        aMsgToSend.setEnabled(true);
        // set focus
        aMsgToSend.requestFocus();
    }
    public void actionPerformed(ActionEvent e)
    {
        Object src = e.getSource();
        if(src == bConnect) {
            if(lastConnectedRep != null) {
                repProfile.setRepInfo(lastConnectedRep);
                repProfile.repaint();
                setMessage("You are reconnected to rep " + lastConnectedRep.getName());

                // Inform server
                ibc.bankMgrCorba.connectToRep(ibc.ClientID, lastConnectedRep.getID());
                enterConnectState();
            }
        } else if(src == bDisconnect) {
            // Inform server
            if(repCorba != null){
                repCorba.addConversationRec("------End session with " +
                                ibc.cClientInfo.name + "-------\n");
            }
            if(repProfile != null) {
                aConversationRec.append("------End session with " +
                                repProfile.getRepInfo().getName()+ "-------\n\n");
                setMessage("You are disconnected from rep " +
repProfile.getRepInfo().getName());
                ibc.bankMgrCorba.disconnectToRep(ibc.ClientID,
repProfile.getRepInfo().getID());
            }
            enterDisconnectState();
        } else if(src == bMain) {
            ibc.removeAll();
            ibc.setBackground(Color.white);
            ibc.ClientMainMenu.init();
        } else if(src == bExit) {
            ibc.onQuit();
        } else if(src == bBack) {
            ibc.removeAll();
            ibc.setBackground(Color.white);
            ibc.ClientServiceInfo.init();
        } else if(src == bSend) {
            String msg = aMsgToSend.getText();
            if(msg == null ||msg.equals("")) {
                return; //nothing to send
            }
            msg = ibc.cClientInfo.name + ":\t" + msg + "\n";
            if(repCorba != null) {
                repCorba.addConversationRec(msg);
            }
            addConversationRec(msg);
            aMsgToSend.setText("");
        } else if(src == bClear) {
            aMsgToSend.setText("");
        }
    }
    public void itemStateChanged(ItemEvent e)
    {
```

```
      Object src = e.getSource();
      if(src == lstDocList) {
         String item = lstDocList.getSelectedItem();
         if(ibc.fBrowser == true) {
            AppletContext context = ibc.getAppletContext();
            if(context != null) {
               try {
                  // get rep IP address
                  String IP = repProfile.getRepInfo().getIP();
                  StringTokenizer strToker = new StringTokenizer(IP, "/");
                  String host = strToker.nextToken();
                  String urlString = "http://" + host +":15000/attach/" + item;

                  URL url = new URL(urlString);
                  context.showDocument(url, "NEW WINDOW");
                  //context.showDocument(url);
               }catch(MalformedURLException ex) {
                  System.out.println("Bad URL:");
               }
            }
         }
      }
   }
}
class IBClientServiceUI extends Panel

{
   // Conference form variables
   protected TextField tStartTime;      // start time
   protected TextField tNumber;      // the number
   protected TextField tMessage;     // message send by server
   protected RepInfoArea repProfile;   // representative profile
   protected TextArea aConversationRec;// conversation record
   protected java.awt.List lstDocList; // list of document sent from rep
   protected TextArea aMsgToSend;       // message to send
   protected Button bSend;           // send mesage
   protected Button bClear;          // clear message to send
   protected Button bExit;           // Exit
   protected Button bBack;           // back to last menu
   protected Button bMain;           // back to main menu
   protected Button bConnect;        // connnect to rep
   protected Button bDisconnect;     // disconnect from rep

   protected Label lTitle;
   protected Label lStart;
   protected Label lEnd;

   protected Label lNumber;
   protected Label lMessage;
   protected Label lRepProfile;

   protected Label lConversationRec;
   protected Label lDocList;
   protected Label lMsgToSend;
   protected Color cBack;
   protected Color cFore;
   protected Font fTitle;
   protected Font fLabel;

   public  IBClient ibc;
   public IBClientServiceUI(){
      // Creat text fields, areas and buttons
      tStartTime = new TextField(5);
      tNumber = new TextField(5);
      tMessage = new TextField(70);
      repProfile = new RepInfoArea();
      aConversationRec = new TextArea(10,80);
      lstDocList = new java.awt.List();
      aMsgToSend = new TextArea(10,80);
      bSend = new Button("Send");
      bClear = new Button("Clear");
```

```
    bExit = new Button("Exit");
    bBack = new Button("Back");
    bMain = new Button("Main");
    bConnect = new Button("Connect");
    bDisconnect = new Button("Disconnect");

    //cBack = new Color(200,200,200);
    cBack = Color.lightGray;
    cFore = new Color(0,0,0);
    fLabel = new Font("SansSerif",Font.PLAIN,12);
    fTitle = new Font("SansSerif",Font.BOLD,16);
    lTitle = new Label("BANK CLIENT" ,Label.CENTER);
    lStart = new Label("Start Time:");
    lEnd = new Label("  Stop Time:");
    lNumber = new Label("You are in waiting list of ");
    lMessage = new Label("Message: ");
    lRepProfile = new Label("Representative profile:");
    lConversationRec = new Label("Conversation Record:");
    lDocList = new Label("Documents from Representative");
    lMsgToSend = new Label("Type the message you want to send in following box:");
}
public void initUI()
{
    // label for title
    lTitle.setBackground(cBack);
    lTitle.setForeground(cFore);
    lTitle.setFont(fTitle);

    // label for start time, left time and participant#
    lStart.setBackground(cBack);
    lStart.setForeground(cFore);
    lStart.setFont(fLabel);

    lNumber.setBackground(cBack);
    lNumber.setForeground(cFore);
    lNumber.setFont(fLabel);

    // label for message
    lMessage.setBackground(cBack);
    lMessage.setForeground(cFore);
    lMessage.setFont(fLabel);

    // label for rep profile
    lRepProfile.setBackground(cBack);
    lRepProfile.setForeground(cFore);
    lRepProfile.setFont(fLabel);

    // label for conversation record
    lConversationRec.setBackground(cBack);
    lConversationRec.setForeground(cFore);
    lConversationRec.setFont(fLabel);

    // label for document list
    lDocList.setBackground(cBack);
    lDocList.setForeground(cFore);
    lDocList.setFont(fLabel);

    lstDocList.setBackground(Color.lightGray);

    // conversation record
    aConversationRec.setBackground(Color.lightGray);
    aConversationRec.setEditable(false);

    // label for message to send
    lMsgToSend.setBackground(cBack);
    lMsgToSend.setForeground(cFore);
    lMsgToSend.setFont(fLabel);

    // message to send
    aMsgToSend.setBackground(Color.white);
    aMsgToSend.setEditable(true);
```

```
     // Add panels in the container

    ibc.add(this);
    setBackground(cBack);
    setLayout(null);
    setBounds(0,0,650,650);

    // add components to the control panel
    add(lTitle);
    add(lStart);
    add(tStartTime);
    add(lEnd);
    add(lNumber);
    add(tNumber);
    add(lMessage);
    add(tMessage);
    add(lRepProfile);
    add(repProfile);
    add(lConversationRec);
    add(aConversationRec);
    add(lDocList);
    add(lstDocList);
    add(lMsgToSend);
    add(aMsgToSend);
    add(bSend);
    add(bClear);
    add(bExit);
    add(bBack);
    add(bMain);
    add(bConnect);
    add(bDisconnect);

    // relocate components
    lTitle.setBounds(170,5,300,30);
    lMessage.setBounds(10,35,60,20);
    tMessage.setBounds(80,35,550,20);
    lRepProfile.setBounds(10,70,200,20);
    repProfile.setBounds(10,90,400,160);
    repProfile.setBackground(Color.gray);

    lStart.setBounds(450,90,70,20);
    tStartTime.setBounds(450,120,160,20);
    lNumber.setBounds(450,170,200,20);
    tNumber.setBounds(450,200,80,20);

    lConversationRec.setBounds(10,260,200,20);
    aConversationRec.setBounds(10,280,400,120);
    lDocList.setBounds(420,260,200,20);
    lstDocList.setBounds(420,280,200,120);
    lMsgToSend.setBounds(10,415,400,20);
    aMsgToSend.setBounds(10,440,630,100);
    bSend.setBounds(180,550,80,30);
    bClear.setBounds(280,550,80,30);
    bExit.setBounds(380,550,80,30);
    bBack.setBounds(180,590,80,30);
    bMain.setBounds(280,590,80,30);
    //bConnect.setBounds(190,590,80,30);
    bDisconnect.setBounds(380,590,80,30);
    aMsgToSend.requestFocus();

    bSend.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    bClear.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    bExit.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    bBack.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    bMain.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    bDisconnect.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
}
public void setStartTime(String s){
    tStartTime.setText(s);
}
public void setMessage(String s){
```

```
        tMessage.setText(s);
    }
    public void setTitle(String s)
    {
        lTitle.setText(s);
    }
    public void setNumber(String s){
        tNumber.setText(s);
    }
    public void addConversationRec(String s){
        aConversationRec.append(s);
    }
    public void addDocList(String s) {
        lstDocList.add(s);
    }
    public void updateRepProfile(CRepInfo cRepInfo) {
        IBRepInfo repInfo = repProfile.getRepInfo();
        if(repInfo != null) {
            repInfo.setCRepInfo(cRepInfo);
            repProfile.setRepInfo(repInfo);
            repProfile.repaint();
        }
    }
    public int updateWaitListPos(int step){
        int oldPos = Integer.parseInt(tNumber.getText());
        int newPos = oldPos - step;
        tNumber.setText(Integer.toString(newPos));
        return newPos;
    }
}
```

```
/**************************************************************************
 * File name:      IBClientPayBill.java
 * Author:         Haiyu Huang
 * Date:           Oct 29, 2000
 * Descriptions:   IIB Client bill payment implementation
 * 1. Create and initialize UI widgets
 * 2. Load all available bill payment services (BELL and Videotron)
 * 3. Display client's bill information according to client selection
 * 4. Perform bill payment operation if pay button is clicked
 *************************************************************************/

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

public class IBClientPayBill extends Panel
implements ItemListener,ActionListener
{
   private IBClient ibc;
   private IBRep   ibr;

   private Font fTitle;
   private Font fLabel;

   private Label lTitle;
   private Label lBillList;
   private java.awt.List lstBillList;
   private Label lAmountToPay;
   private TextField tAmount;
   private Label lWhichAccount;
   private Choice cWhichAccount;

   private MessageArea caMessage;
   private TextField tAccountNo;

   private Button bOK;
   private Button bCancel;
   private Button bBack;
   private Button bClose;
   private Button bPay;

   BillInfoArea bellBillInfo;
   BillInfoArea videotronBillInfo;

   IBClientPayBill(){
      ibc = null;
      ibr = null;
      fTitle = new Font("SansSerif",Font.BOLD,15);
      fLabel = new Font("SansSerif",Font.PLAIN,12);
      lTitle = new Label("Bill Payment Service", Label.CENTER);
      lBillList = new Label("Following bill payment is available");
      lstBillList = new List();
      lstBillList.addItemListener(this);
      lstBillList.addItem("Bell");
      lstBillList.addItem("Videotron");
      bellBillInfo = new BillInfoArea();
      videotronBillInfo = new BillInfoArea();
      lAmountToPay = new Label("Enter the amount to pay");
      tAmount = new TextField(20);
      lWhichAccount = new Label("From which account");
      cWhichAccount = new Choice();
      cWhichAccount.addItemListener(this);

      caMessage = new MessageArea();
      tAccountNo = new TextField();
      bBack = new Button("Back to main");
      bBack.addActionListener(this);
      bClose = new Button("Close");
      bClose.addActionListener(this);
      bPay = new Button("Pay");
      bPay.addActionListener(this);
```

```
    bOK = new Button("OK");
    bOK.addActionListener(this);
    bCancel = new Button("Cancel");
    bCancel.addActionListener(this);
 }

 public void setIBClient(IBClient ibc) {
    this.ibc = ibc;
 }
 public void setIBRep(IBRep ibr) {
    this.ibr = ibr;
 }
 public void init(){

    Color cBack = Color.lightGray;
    if(ibc != null){
       ibc.add(this);
    }
    setBackground(cBack);
    setLayout(null);
    //setBounds(150,20,550, 500);
    setBounds(0,0,550, 500);

    lTitle.setForeground(Color.black);
    lTitle.setFont(fTitle);

    add(lTitle);
    add(lBillList);
    add(lstBillList);
    lstBillList.select(0);
    loadBell();
    add(lAmountToPay);
    add(tAmount);
    add(lWhichAccount);
    add(cWhichAccount);
    if(ibc != null) {
       add(bBack);
    }else {
       add(bClose);
    }
    add(bPay);

    lTitle.setBounds(20,20,500,30);
    lBillList.setBounds(20,60,200,30);
    lstBillList.setBounds(20, 100, 180, 250);
    lAmountToPay.setBounds(20, 360, 180, 30);
    tAmount.setBounds(20, 390, 120, 30);
    lWhichAccount.setBounds(220, 360, 180, 30);
    cWhichAccount.setBounds(220, 390, 120, 30);
    bPay.setBounds(360, 390, 100, 30);
    bBack.setBounds(20, 440,120, 30);
    bClose.setBounds(20, 440,120, 30);
    initAccount();
    initAmountToPay();
    if(ibc != null){
       ibc.lastUI = "IBCLIENTPAYBILL";
       // Force repainting
       ibc.setVisible(false);
       ibc.setVisible(true);
    }
 }
 public void setTitle(String s){
    lTitle.setText(s);
 }
 public void initAccount(){
    CAccount[]  cAccounts = null;
    if(ibc != null) {
       cAccounts = ibc.bankMgrCorba.getAccounts(ibc.ClientID);
    } else if(ibr != null) {
       IBClientInfo clientInfo = ibr.repService.getCurClientInfo();
       if(clientInfo!= null){
```

```
            cAccounts = ibr.bankMgrCorba.getAccounts(clientInfo.getAcctNum());
         }
      } else {
         return;
      }
      cWhichAccount.removeAll();
      for(int i =0; i<cAccounts.length; i++) {
         if(i>=1 && cAccounts[i].type == cAccounts[0].type){
            break;//following data is not valid (same as the first one)
         }
         if(cAccounts[i].type == 'S') { // Saving account
            cWhichAccount.add("Saving Account");
         }
         if(cAccounts[i].type == 'C') {// Check account
            cWhichAccount.add("Check Account");
         }
         // more account can be added.....
      }
   }
   public void initAmountToPay(){
      String item = lstBillList.getSelectedItem();
      float amount = 0;
      if(item.equals("Bell")) {
         amount = (Float.valueOf(bellBillInfo.getBalanceForward())).floatValue() +
                (Float.valueOf(bellBillInfo.getAmountDue())).floatValue();

      } else if(item.equals("Videotron")){
         amount = (Float.valueOf(videotronBillInfo.getBalanceForward())).floatValue() +
                (Float.valueOf(videotronBillInfo.getAmountDue())).floatValue();
      }
      // more bill pay can be added
      if(amount <= 0) {
            amount = 0;
      }
      tAmount.setText(Float.toString(amount));
   }

   private boolean IsFloat(String s)
   {
      for(int i=0; i<s.length(); i++) {
         if((s.charAt(i) != 46 && s.charAt(i)< 48) || s.charAt(i) > 57) {
            System.out.println(s + " is not a float number");
            return false;
         }
      }
      return true;
   }

   public void pay(){
      String acctNum = null;
      float amountToPay;
      String bankAcctType = null;

      String amount = tAmount.getText();

      if(!IsFloat(amount)) {
         return;
      } else {
         String item = lstBillList.getSelectedItem();
         if(item.equals("Bell")) {
            acctNum = bellBillInfo.getAccountNum();
         } else if(item.equals("Videotron")){
            acctNum = videotronBillInfo.getAccountNum();
         }
         item = cWhichAccount.getSelectedItem();
         if(item.equals("Check Account")){
            bankAcctType = "C";
         }else if (item.equals("Saving Account")){
            bankAcctType = "S";
         }
         // more account can be added
```

```
            amountToPay = (Float.valueOf(amount)).floatValue();
            if(ibc != null) {
               ibc.bankMgrCorba.payBill(acctNum, amountToPay, ibc.ClientID, bankAcctType);
            } else if(ibr != null){
               IBClientInfo clientInfo = ibr.repService.getCurClientInfo();
               if(clientInfo != null) {
                  ibr.bankMgrCorba.payBill(acctNum, amountToPay, clientInfo.getAcctNum(),
bankAcctType);
               }
            }
         }
         tAmount.setText("");
      }

      public void showBillRegisterUI(String s1, String s2, String s3, String s4) {
         caMessage.setMessage(s1,s2,s3,s4);
         add(caMessage);
         caMessage.setBounds(220, 100, 300, 100);
         add(tAccountNo);
         tAccountNo.setBounds(240, 200, 200, 25);
         add(bOK);
         bOK.setBounds(240, 270,80, 30);
         add(bCancel);
         bCancel.setBounds(330, 270,80, 30);
      }
      public void loadBell(){
         remove(caMessage);
         remove(tAccountNo);
         remove(bOK);
         remove(bCancel);
         remove(videotronBillInfo);
         // Connnect to db to get bell bill account information
         String billNum = "";
         if(ibc != null) {
            billNum = ibc.bankMgrCorba.findBillAcctNum(ibc.cClientInfo.acctNum, "BELL");
         }else if(ibr != null){
            IBClientInfo clientInfo = ibr.repService.getCurClientInfo();
            if(clientInfo != null) {
            billNum = ibr.bankMgrCorba.findBillAcctNum(clientInfo.getAcctNum(), "BELL");
            }
         }else {
            return;
         }
         if(!billNum.equals("")) {
            CBillInfo cBillInfo = null;
            if(ibc != null) {
               cBillInfo = ibc.bankMgrCorba.getBillInfo(billNum);
            }else if(ibr != null){
               cBillInfo = ibr.bankMgrCorba.getBillInfo(billNum);
            }else {
               return;
            }
            bellBillInfo.setAccountNum(cBillInfo.acctNum);
            bellBillInfo.setBalanceForward(Float.toString(cBillInfo.balanceForward));
            bellBillInfo.setAmountDue(Float.toString(cBillInfo.amountDue));
            add(bellBillInfo);
            bellBillInfo.setBounds(220, 100, 250, 250);
            bellBillInfo.setBackground(Color.pink);
            Image i = null;

            if(ibc != null && ibc.fBrowser) {
               i = ibc.getImage(ibc.getCodeBase(), "./images/bell_logo.gif");
            } else {
               Toolkit toolkit = Toolkit.getDefaultToolkit();
               i = toolkit.getImage("./images/bell_logo.gif");
            }
            bellBillInfo.setLogo(i);
         } else {
            showBillRegisterUI("The system can not find your Bell account.",
                               "If you are Bell customer, please register",
                               "the bill payment service by entering your Bell",
```

```
                        "account number in the following box, and click OK");
        }

    }

    public void loadVideotron(){
        remove(caMessage);
        remove(tAccountNo);
        remove(bOK);
        remove(bCancel);
        remove(bellBillInfo);
        // Connnect to db to get bell bill account information
        String billNum = "";
        if(ibc != null) {
            billNum = ibc.bankMgrCorba.findBillAcctNum(ibc.cClientInfo.acctNum,
"Videotron");
        }
        else if(ibr != null){
            IBClientInfo clientInfo = ibr.repService.getCurClientInfo();
            if(clientInfo != null) {
                ibr.bankMgrCorba.findBillAcctNum(clientInfo.getAcctNum(), "Videotron");
            }
        }else {
            return;
        }
        if(!billNum.equals("")) {
            CBillInfo cBillInfo = null;
            if(ibc != null){
                cBillInfo = ibc.bankMgrCorba.getBillInfo(billNum);
            } else if(ibr != null) {
                cBillInfo = ibr.bankMgrCorba.getBillInfo(billNum);
            }
            videotronBillInfo.setAccountNum(cBillInfo.acctNum);
            videotronBillInfo.setAmountDue(Float.toString(cBillInfo.amountDue));
            add(videotronBillInfo);
            videotronBillInfo.setBounds(220, 100, 250, 250);
            videotronBillInfo.setBackground(Color.pink);
            Image i = null;
            if(ibc != null && ibc.fBrowser) {
                i = ibc.getImage(ibc.getCodeBase(), "./images/videoTron_logo.gif");
            } else {
                Toolkit toolkit = Toolkit.getDefaultToolkit();
                i = toolkit.getImage("./images/videoTron_logo.gif");
            }
            videotronBillInfo.setLogo(i);
        } else {
            showBillRegisterUI("The system can not find your Videotron account.",
                               "If you are Videotron customer, please register",
                               "the bill payment service by entering your Videotron",
                               "account number in the following box, and click OK");

        }
    }
    public void itemStateChanged(ItemEvent e)
    {
        Object src = e.getSource();
        if(src == lstBillList) {
            String item = lstBillList.getSelectedItem();
            //repInfo.setName(item);

            if(item.equals("Bell")){
                loadBell();
            } else if(item.equals("Videotron")) {
                loadVideotron();
            }

        }
        if(src == cWhichAccount) {
            String account = (String)e.getItem();
        }
    }
```

```
    public void actionPerformed(ActionEvent e)
    {
       Object src = e.getSource();
       if(src == bBack) {
          ibc.removeAll();
          ibc.setBackground(Color.white);
          ibc.ClientMainMenu.init();
       } else if(src == bClose){
          ibr.repService.frame_ibcpb.setVisible(false);
       }else if(src == bPay) {
          pay();
       } else if(src == bOK) {

       } else if(src == bCancel) {

       }
    }
}

class BillInfoArea extends Canvas
{
   private String accountNum;
   private String amountDue;
   private String balanceForward;

   private Image logo;
   private Font fLabel;

   public BillInfoArea()
   {
      super();
      accountNum = "";
      amountDue = "";
      balanceForward = "";
      logo = null;
      fLabel = new Font("SansSerif",Font.PLAIN,12);
   }

   public BillInfoArea(String accountNum, String amountDue,
                  String dateDue,Image logo)
   {
      super();
      this.accountNum = accountNum;
      this.amountDue = amountDue;
      this.balanceForward = balanceForward;
      this.logo = logo;
   }
   // inquery
   public String getAccountNum(){
      return accountNum;
   }
   public String getAmountDue(){
      return amountDue;
   }
   public String getBalanceForward(){
      return balanceForward;
   }

   public Image getLogo(){
      return logo;
   }
   // mutator
   public void setAccountNum(String s){
      accountNum = s;
      repaint();
   }
   public void setAmountDue(String s){
      amountDue = s;
      repaint();
   }
   public void setBalanceForward(String s){
```

```
        balanceForward = s;
        repaint();
    }
    public void setLogo(Image i) {
        logo = i;
        repaint();
    }
    public void paint(Graphics g) {
        g.drawString("Account Number:", 20, 100);
        g.drawString(accountNum, 120, 100);

        g.drawString("Balance Forward:", 20, 130);
        g.drawString(balanceForward, 120, 130);

        g.drawString("Amount Due:", 20, 160);
        g.drawString(amountDue, 120, 160);

        if(logo != null) {
            int w = logo.getWidth(this);
            int h = logo.getHeight(this);
            float ratio =  (float) w/(float)h;
            int logoWidth = (int) (ratio * getSize().height/4);
            g.drawImage(logo, 0, 0, logoWidth, getSize().height/4, this);
        }
    }
}

class MessageArea extends Canvas
{
    private String message1 = "";
    private String message2 = "";
    private String message3 = "";
    private String message4 = "";

    public void setMessage(String s1,String s2,String s3,String s4){
        message1 = s1;
        message2 = s2;
        message3 = s3;
        message4 = s4;
        repaint();
    }
    public void paint(Graphics g) {
        g.drawString(message1, 10,10);
        g.drawString(message2, 10,30);
        g.drawString(message3, 10,50);
        g.drawString(message4, 10,70);
    }
}
```

```
/*************************************************************
 * File name:        IBClientBalance.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB Client Balance
 * 1. Create and initialize UI widgets
 * 2. Display client accounts balances
 *************************************************************/
import java.awt.*;
import java.awt.event.*;

public class IBClientBalance extends Panel
implements ItemListener,ActionListener
{
   private IBClient ibc;
   private IBRep ibr;
   private Font fTitle;
   private Font fLabel;

   private Label     lTitle;
   private Label     lBalance;
   private TextField tBalance;
   private Label     lWhichAccount;
   private Choice    cWhichAccount;

   private Button    bBack;
   private Button    bClose;

   private float saveBalance;
   private float checkBalance;

   IBClientBalance(){
      ibc = null;
      ibr = null;
      fTitle = new Font("SansSerif",Font.BOLD,15);
      fLabel = new Font("SansSerif",Font.PLAIN,12);
      lTitle = new Label("Account Balance ", Label.CENTER);

      lBalance = new Label("Balance");
      tBalance = new TextField(20);

      lWhichAccount = new Label("Account");
      cWhichAccount = new Choice();
      cWhichAccount.addItemListener(this);

      bBack = new Button("Back to main");
      bBack.addActionListener(this);
      bClose = new Button("Close");
      bClose.addActionListener(this);
      saveBalance = 0;
      checkBalance = 0;
   }
   public void setIBClient(IBClient ibc) {
      this.ibc = ibc;
   }
   public void setIBRep(IBRep ibr) {
      this.ibr = ibr;
   }
   public void setTitle(String s) {
      lTitle.setText(s);
   }
   public void init(){

      Color cBack = Color.lightGray;
      if(ibc != null){
         ibc.add(this);
      }
      setBackground(cBack);
      setLayout(null);
      //setBounds(150,20,400, 300);
      setBounds(0,0,400, 300);
```

```
    lTitle.setForeground(Color.black);
    lTitle.setFont(fTitle);

    add(lTitle);
    add(lBalance);
    add(tBalance);
    add(lWhichAccount);
    add(cWhichAccount);
    if(ibc == null) {
       add(bClose);
    } else {
       add(bBack);
    }
    lTitle.setBounds(10,20,500,30);

    lWhichAccount.setBounds(30, 80, 180, 30);
    cWhichAccount.setBounds(30, 110, 120, 30);

    lBalance.setBounds(220, 80, 180, 30);
    tBalance.setBounds(220, 110, 150, 30);

    bBack.setBounds(30, 180, 120, 30);
    bClose.setBounds(30, 180, 120, 30);

    loadBalance();

    // Force repainting
    if(ibc != null) {
       ibc.setVisible(false);
       ibc.setVisible(true);
    }
 }
 public void loadBalance(){
    CAccount[]  cAccounts = null;
    cWhichAccount.removeAll();
    if(ibc != null) {
       cAccounts = ibc.bankMgrCorba.getAccounts(ibc.ClientID);
    } else if(ibr != null) {
       IBClientInfo clientInfo = ibr.repService.getCurClientInfo();
       if(clientInfo.= null){
          cAccounts = ibr.bankMgrCorba.getAccounts(clientInfo.getAcctNum());
       }
    } else {
       return;
    }
    for(int i =0; i<cAccounts.length; i++) {
       if(i>=1 && cAccounts[i].type == cAccounts[0].type){
          break;//following data is not valid (same as the first one)
       }
       if(cAccounts[i].type == 'S') { // Saving account
          cWhichAccount.add("Saving Account");
          saveBalance = cAccounts[i].balance;
       }
       if(cAccounts[i].type == 'C') {// Check account
          cWhichAccount.add("Check Account");
          checkBalance = cAccounts[i].balance;
       }
       // more account can be added.....
    }
    String item = cWhichAccount.getSelectedItem();
    if(item.equals("Saving Account")) {
       setBalance(Float.toString(saveBalance));
    } else if(item.equals("Check Account")){
       setBalance(Float.toString(checkBalance));
    }
    // more
 }
 public void itemStateChanged(ItemEvent e)
 {
    Object src = e.getSource();
```

```
        if(src == cWhichAccount) {
            String account = (String)e.getItem();
            if(account.equals("Saving Account")){
                setBalance(Float.toString(saveBalance));
            }else if(account.equals("Check Account")){
                setBalance(Float.toString(checkBalance));
            }
        }
    }
    public void setBalance(String s) {
        tBalance.setText(s);
    }
    public void actionPerformed(ActionEvent e)
    {
        Object src = e.getSource();
        if(src == bBack) {
            ibc.removeAll();
            ibc.setBackground(Color.white);
            ibc.ClientMainMenu.init();
        } else if(src == bClose) {
            if(ibr != null) {
                ibr.repService.frame_ibcb.setVisible(false);
            }
        }
    }
}
```

```
/*************************************************************
 * File name:        IBClientTransaction.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB Client transaction
 * 1. Create and initialize UI widgets
 * 2. Display the client account transactions
 *************************************************************/
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.util.*;

public class IBClientTransaction extends Panel
implements ActionListener, ItemListener
{
   // Transaction output
   private Choice    cWhichAccount;
   private TextArea  aTransaction;
   private Button    bClose;        // close
   private Button    bClear;        // clear out put
   private Button    bBack;         // bact to main menu if client
   private Label     lTitle;

   private Color cBack;
   private Color cFore;
   private Font fTitle;
   private Font fLabel;

   private IBClient ibc;
   private IBRep ibr;

   private String saveTransaction;
   private String checkTransaction;

   public IBClientTransaction(){
      ibc = null;
      ibr = null;
      cWhichAccount = new Choice();
      cWhichAccount.addItemListener(this);
      aTransaction = new TextArea(10,80);
      bClose = new Button("Close");
      bBack = new Button("Back to main");
      bClose.addActionListener(this);
      bBack.addActionListener(this);
      bClear = new Button("Clear");
      bClear.addActionListener(this);

      //cBack = new Color(200,200,200);
      cBack = Color.lightGray;
      cFore = Color.black;
      fLabel = new Font("SansSerif",Font.PLAIN,18);
      lTitle = new Label("Transactions", Label.CENTER);

      saveTransaction = "";
      checkTransaction ="";
   }

   public void setIBClient(IBClient ibc) {
      this.ibc = ibc;
   }

   public void setIBRep(IBRep ibr) {
      this.ibr= ibr;
   }
   public void setTitle(String s) {
      lTitle.setText(s);
   }
   public void init()
   {
      // Title
```

```
    lTitle.setBackground(cBack);
    lTitle.setForeground(cFore);
    lTitle.setFont(fLabel);

    // Transactions
    aTransaction.setBackground(Color.white);
    aTransaction.setEditable(false);
    setBackground(cBack);
    //setBounds(20, 20, 680, 550);
    setBounds(0, 0, 680, 550);
    if(ibc != null) {
       ibc.add(this);
    }
    setLayout(null);

    // add components to the control panel
    add(lTitle);
     add(cWhichAccount);
    add(aTransaction);
    if(ibc == null) {
       add(bClose);
    } else {
       add(bBack);
    }
    add(bClear);
    // relocate components
    lTitle.setBounds(30,50,600,20);
    cWhichAccount.setBounds(30, 80, 120, 30);
    aTransaction.setBounds(30,160,600,320);
    bClose.setBounds(30,500,120,30);
    bBack.setBounds(30,500,120,30);
    //bClear.setBounds(270,500,80,30);

     // Register listerner
    bClose.addActionListener(this);
    bBack.addActionListener(this);
    bClear.addActionListener(this);

    getTransaction();

    // Force repainting
    if(ibc != null) {
       ibc.setVisible(false);
       ibc.setVisible(true);
    }
    bClear.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    bClose.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    bBack.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    cWhichAccount.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
}

public void getTransaction(){
    CAccount[]  cAccounts = null;
    if(ibc != null) {
       cAccounts = ibc.bankMgrCorba.getAccounts(ibc.ClientID);
    }
     else if(ibr != null) {
       IBClientInfo clientInfo = ibr.repService.getCurClientInfo();
       if(clientInfo!= null){
          cAccounts = ibr.bankMgrCorba.getAccounts(clientInfo.getAcctNum());
       }
    } else {
       return;
    }
    cWhichAccount.removeAll();
    for(int i =0; i<cAccounts.length; i++) {
       if(i>=1 && cAccounts[i].type == cAccounts[0].type){
          break;//following data is not valid (same as the first one)
       }
       if(cAccounts[i].type == 'S') { // Saving account
          cWhichAccount.add("Saving Account");
```

```
            if(ibc != null) {
               saveTransaction = ibc.bankMgrCorba.getTransaction(ibc.ClientID, "S");
            }else if(ibr != null) {
               IBClientInfo clientInfo = ibr.repService.getCurClientInfo();
               if(clientInfo!= null){
                  saveTransaction  =
ibr.bankMgrCorba.getTransaction(clientInfo.getAcctNum(),"S");
               }
            } else {
               return;
            }
         }
         if(cAccounts[i].type == 'C') {// Check account
            cWhichAccount.add("Check Account");
            if(ibc != null) {
               checkTransaction = ibc.bankMgrCorba.getTransaction(ibc.ClientID, "C");
            } else if(ibr != null) {
               IBClientInfo clientInfo = ibr.repService.getCurClientInfo();
               if(clientInfo!= null){
                  checkTransaction  =
ibr.bankMgrCorba.getTransaction(clientInfo.getAcctNum(),"C");
               }
            } else {
               return;
            }
         }
         // more account can be added......
      }
      // By default, transaction of the first account is loaded
      cWhichAccount.select(0);
      String item = cWhichAccount.getSelectedItem();
      clearTransaction();
      if(item.equals("Saving Account")) {
         setTransaction("Saving Account");
      } else if(item.equals("Check Account")){
         setTransaction("Check Account");
      }
   }

   public void actionPerformed(ActionEvent e)
   {
      Object src = e.getSource();
      if(src == bClear) {
         } else if(src == bClose) {
         if(ibr != null) {
            ibr.repService.frame_ibct.setVisible(false);
         }
      } else if(src == bBack) {
         ibc.removeAll();
         ibc.setBackground(Color.white);
         ibc.ClientMainMenu.init();
      }
   }
   public void itemStateChanged(ItemEvent e)
   {
      Object src = e.getSource();
      if(src == cWhichAccount) {
         String account = (String)e.getItem();
         if(account.equals("Saving Account")){
            setTransaction("Saving Account");
         }else if(account.equals("Check Account")){
            setTransaction("Check Account");
         }
      }
   }
   public void addTransaction(String s)
   {
      aTransaction.append(s);
   }
   public void clearTransaction(){
      aTransaction.setText("");
```

```
    }
    public void setTransaction(String acctType) {
       StringTokenizer theToks = null;
       if(acctType.equals("Saving Account")) {
          theToks = new StringTokenizer(saveTransaction, "\n");
       } else if(acctType.equals("Check Account")){
          theToks = new StringTokenizer(checkTransaction, "\n");
       }
       if(theToks != null) {
          // clear previous one
          clearTransaction();
          while(theToks.hasMoreTokens()) {
             addTransaction(theToks.nextToken()+"\n");
          }
       }
    }
    public static void main(String[] s)
    {
       IBClientTransaction ibt = new IBClientTransaction();
       ibt.addTransaction("T1");
       ibt.init();
       Frame f = new Frame();
       f.setLayout(null);
       f.add(ibt);
       f.setSize(700,600);
       f.setVisible(true);
       f.setTitle("IIB Transaction");
       f.addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e)
       {
          System.exit(0);}
       });
    }
}
```

B.2 Source Code for CSAR

```
/*************************************************************************
 * File name:        IBRep.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB Representative main program
 * It dose the followings.
 * 1. Create and initialize UI widgets
 * 2. Initialize corba
 * 3. Locate server side Corba object Bank Manager, create
 *    representative side callback object
 *************************************************************************/

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

import org.omg.PortableServer.*;
import com.inprise.vbroker.PortableServerExt.BindSupportPolicyValue;
import com.inprise.vbroker.PortableServerExt.BindSupportPolicyValueHelper;

public class IBRep
{
   // Representative Login form
   public IBRepLogin repLogin;
   // Representative Service main user interface
   public IBRepService repService;
   // Corba object reference of Server bank manager
   public Server.BankMgrCorba bankMgrCorba;
   // callback Corba object reference of representative
   public IBRepCorbaImpl repCorbaImpl;
   public Representative.RepCorba repCorba;
   // login fail message dialog
   public MessageDialog msgDlg;

   // registered callback Corbar object index maintained by bank manager on server side
   public int RepID;
   // more information about the rep
   public CRepInfo cRepInfo;
   // indicate if the server is down
   boolean fServerDown;

   public IBRep()
   {
      repLogin = new IBRepLogin(this);
      repService = new IBRepService(this);
      msgDlg = new MessageDialog(repLogin, "Login Fail", true);
      RepID = 0;
      cRepInfo = null;
      fServerDown = true;
   }

   public void initCorba()
   {
      String[] args = null;
      try
      {  // Initialize ORB
         org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
         // resolve the bankMgrCorba object reference
         bankMgrCorba = Server.BankMgrCorbaHelper.bind(orb,"BankManager");
         System.out.println("bank manager corba object BankManager is found ");

         try {
            // Activate it on the default POA which is root POA for this servant
            POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            // Create policies for our persistent POA
```

```
            org.omg.CORBA.Any any = orb.create_any();
            BindSupportPolicyValueHelper.insert(any, BindSupportPolicyValue.BY_INSTANCE);
            org.omg.CORBA.Policy bsPolicy =
orb.create_policy(com.inprise.vbroker.PortableServerExt.BIND_SUPPORT_POLICY_TYPE.value,
any);

            org.omg.CORBA.Policy[] policies = {
              rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT), bsPolicy
            };
            // Create myPOA with the right policies
            POA myPOA = rootPOA.create_POA( "bank_rep_poa", rootPOA.the_POAManager(),
                                            policies );
            // Create the servant
            repCorbaImpl = new IBRepCorbaImpl(this);

            // Decide on the ID for the servant
            byte[] BankRepId = "BankRep".getBytes();
            // Activate the servant with the ID on myPOA
            myPOA.activate_object_with_id(BankRepId, repCorbaImpl);
            rootPOA.the_POAManager().activate();
            System.out.println(myPOA.servant_to_reference(repCorbaImpl) +
                        " is ready.");
            repCorba =
Representative.RepCorbaHelper.narrow(myPOA.servant_to_reference(repCorbaImpl));
         } catch (Exception e) {
           e.printStackTrace();
         }
         System.out.println("Create rep side callback object repCorba");
         fServerDown = false;
      } catch(Exception e)
      {
         System.out.println("Exception" + e);
      }
   }

   public static void main(String[] s)
   {
      IBRep rep = new IBRep();
      rep.initCorba();
      rep.repLogin.inits();
      rep.repLogin.setSize(300, 200);
      rep.repLogin.setVisible(true);
      rep.repLogin.setTitle("Representative Login");

      rep.repLogin.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
        });
   }
}

/**************************************************************************
 * File name:        IBRepCorbaImpl.java
 * Author:           Haiyu Huang
 * Date:          Nov 3, 2000
 * Descriptions:  This is rep side corba object implementation.
 * it serves as callback object when a representative registers to server.
 * afterwords the server communicates with rep through this callback object
 *************************************************************************/

public class IBRepCorbaImpl extends Representative.RepCorbaPOA
{
   public IBRep ibr;
   public IBRepCorbaImpl(Object rep)
   {
      super();
      ibr = (IBRep)rep;
   }

   public String getRepID()
   {
      return ibr.repLogin.getRepID();
```

```
    }

    public String getPassword()
    {
       return ibr.repLogin.getPassword();
    }
    public String getRepIP()
    {
       return ibr.repLogin.getIP();
    }
    public void updateClientWaitList(CClientInfo cClientInfo,int flag)
    {
       ibr.repService.updateClientWaitList(cClientInfo, flag);
    }
    public void setMessage(String s){
       ibr.repService.setMessage(s);
    }
    public void updateStatus(int status) {
       ibr.repService.updateStatus(status);
    }
    public void addConversationRec(String s){
       ibr.repService.addConversationRec(s);
    }
    public void serverDown(){
       ibr.fServerDown = true;
    }
}
```

```
/*********************************************************************
 * File name:        IBRepLogin.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     Implememenging representative login
 * It dose the followings:
 * 1. Create and initialize UI widgets
 * 2. Register representative callback with bank manager
 * 3. Load representative main windows if success, otherwise
 *    load access deny warning dialog
 *********************************************************************/
import java.awt.*;
import java.awt.event.*;
import java.net.*;

public class IBRepLogin extends Frame
implements ActionListener, KeyListener
{
   // Login form variables
   private TextField tEmpID;     // user name
   private TextField tPassword;  // password
   private Button bLogin;        // login
   private Button bCancel;       // back to last page
   private String pwd;
   private String cfm;
   private IBRep ibr;

   public IBRepLogin(IBRep ibr)
   {
      super();
      this.ibr = ibr;
      // Creat text fields and buttons
      tEmpID = new TextField(12);
      tPassword = new TextField(12);
      tPassword.addKeyListener(this);
      bLogin = new Button("  LOGIN  ");
      bLogin.addActionListener(this);
      bCancel = new Button("  CANCEL  ");
      bCancel.addActionListener(this);
      pwd = "";
   }
   public String getRepID()
   {
      return tEmpID.getText();
   }
   public String getPassword()
   {
      return tPassword.getText();
   }
   public String getIP()
   {
      String hostName = "Unknown";
      String hostName1 = "Unknown";
      try {
         //InetAddress inetAddress = InetAddress.getByName(null);
         InetAddress inetAddress = InetAddress.getLocalHost();
         hostName = inetAddress.toString();
         hostName1 = inetAddress.getHostName();

      }
      catch(UnknownHostException ex) {
         System.out.println("Unknown host");
      }
      return hostName;
   }
   public void inits()
   {
      //Color cBack = new Color(200,200,200);
      Color cBack = Color.white;
      Color cFore = new Color(0,0,0);
      Font fInfo = new Font("SansSerif",Font.PLAIN,20);
```

```
    Font fTf = new Font("SansSerif",Font.PLAIN,18);

    // textfield for username and password
    tEmpID.setFont(fTf);
    tPassword.setFont(fTf);
    tPassword.setEchoChar('*');

    // label for username
    Label lUsr = new Label("Employee ID:");
    lUsr.setBackground(cBack);
    lUsr.setForeground(cFore);
    lUsr.setFont(fInfo);

    // label for password
    Label lPwd = new Label("Password:");
    lPwd.setBackground(cBack);
    lPwd.setForeground(cFore);
    lPwd.setFont(fInfo);

    // Panel for user name
    Panel pUsername = new Panel();
    pUsername.setLayout(new FlowLayout(FlowLayout.CENTER));
    pUsername.add(lUsr);
    pUsername.add(tEmpID);
    // Panel for password
    Panel pPassword = new Panel();
    pPassword.setLayout(new FlowLayout(FlowLayout.CENTER));
    pPassword.add(lPwd);
    pPassword.add(tPassword);
    // Panel for buttons
    Panel pButtons = new Panel();
    pButtons.setLayout(new FlowLayout());
    pButtons.add(bLogin);
    pButtons.add(bCancel);
    setLayout(new FlowLayout(FlowLayout.CENTER));
    setBackground(cBack);
    add(pUsername);
    add(pPassword);
    add(pButtons);
    tEmpID.requestFocus();
}

public boolean register()
{
    boolean fSuccess = false;
    // Register repCorba with bank magager
    int repID = ibr.bankMgrCorba.registerRep((Representative.RepCorba)ibr.repCorba);
    if(repID < 0) //register not succeed
    {
        fSuccess  = false;
        System.out.println("represenative Login Failed");
    } else {
        fSuccess  = true;
        // Setup rep ID
        ibr.RepID = repID;
        // Setup rep information
        ibr.cRepInfo = ibr.bankMgrCorba.getRepInfo(repID);
        System.out.println("Representative Login Success");
    }
    return fSuccess;
}

public void actionPerformed(ActionEvent e)
{
    Object src = e.getSource();
    if(src == bLogin) {

        if(register()) {
            //IBRepService repService = new IBRepService();
            ibr.repService.init();
            ibr.repService.setSize(650, 700);
```

```
            ibr.repService.setVisible(true);
            ibr.repService.setTitle("Interactive Bank for Representative " +
                                    getRepID() +
                                    "(" + ibr.cRepInfo.name + ")");

            ibr.repService.addWindowListener(new WindowAdapter() {
               public void windowClosing(WindowEvent e)
               {
                  ibr.repService.quit();
                  System.exit(0);
               }
               }
            );
            setVisible(false);
            System.out.println("Login Success");
         } else {
            ibr.msgDlg.showDialog();
         }
      }
      if(src == bCancel) {
         System.exit(0);
      }
   }
   // handling Keystrokes
   public void keyTyped(KeyEvent e)
   {
      Object src = e.getSource();
      char key = e.getKeyChar();
      if (src == tPassword) {
         if(key>31 && key<127) {
            pwd = pwd + key;
         } else if(key==8) {
            int l = pwd.length();
            pwd = pwd.substring(0,l-1);
         }
      }
   }
   public void keyReleased(KeyEvent e) {
   }
   public void keyPressed(KeyEvent e) {
   }
}
```

```
/**************************************************************************
 * File name:        IBRepService.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB representative side main windows for
 *                   client/representative interaction
 * It dose the followings
 * 1. Create and initialize UI widgets
 * 2. Load/manage client waiting list and display the first client's profile
 * 3. Implement client/representative interaction logic, start,
 *    disconnect, send message/attach file, exit etc
 *************************************************************************/
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.text.*;

import com.inprise.vbroker.PortableServerExt.BindSupportPolicyValue;
import com.inprise.vbroker.PortableServerExt.BindSupportPolicyValueHelper;

public class IBRepService extends IBRepServiceUI
implements ActionListener
{
   // Information of clients on waiting list
   private Vector clientInfoList;
   // list of file send to clients, element is of type FileInfo
   private Vector fileInfoList;
   // current servering client callback corba object
   private Client.ClientCorba currClientObj;
   protected WarningDialog warnDlg;
   protected IBTermDeposit termDeposit;

   protected IBClientBalance ibcb;
   protected Frame   frame_ibcb;

   protected IBClientTransaction ibct;
   protected Frame   frame_ibct;

   protected IBClientPayBill ibcpb;
   protected Frame   frame_ibcpb;

   public IBRepService(IBRep ibr) {
      super();
      this.ibr = ibr;
      this.parent = this;
      clientInfoList = new Vector();
      fileInfoList = new Vector();
      warnDlg = new WarningDialog(this, "Warnning", true);
      currClientObj = null;
      termDeposit = new IBTermDeposit(this);
      ibcb = new IBClientBalance();
      ibcb.setIBRep(ibr);
      frame_ibcb = new Frame();

      ibct = new IBClientTransaction();
      ibct.setIBRep(ibr);
      frame_ibct = new Frame();

      ibcpb = new IBClientPayBill();
      ibcpb.setIBRep(ibr);
      frame_ibcpb = new Frame();
   }
   public void setCurrServeringClient(Client.ClientCorba currClientObj) {
      this.currClientObj = currClientObj;
   }
   public IBClientInfo getCurClientInfo(){
      return (IBClientInfo)clientInfoList.elementAt(0);
   }
   public void init(){
      initUI();
      // Register listerner
```

```
        bSend.addActionListener(this);
        bClear.addActionListener(this);
        bExit.addActionListener(this);
        bAttach.addActionListener(this);
        bPause.addActionListener(this);
        bStart.addActionListener(this);
        bNext.addActionListener(this);
        bDisconnect.addActionListener(this);
        miBalance.addActionListener(this);
        miTransaction.addActionListener(this);
        miInterestRate.addActionListener(this);
        miPayBill.addActionListener(this);
        miTransMoney.addActionListener(this);
        miTermDeposit.addActionListener(this);
        miUpdateClient.addActionListener(this);

        aMsgToSend.requestFocus();
        setStartTime(getCurrentTime());
        enterIdleState();
        if(clientInfoList.size()<=0) {
           bStart.setEnabled(false);
        }
     }
     public int getNumClient(){
        return clientInfoList.size();
     }
     public int getClientIndex(int ClientID)
     {
        for(int i=0;i<clientInfoList.size(); i++)
        {
           if(ClientID == ((IBClientInfo)clientInfoList.elementAt(i)).getAcctNum())// Found
           {
              return i;
           }
        }
        return -1;
     }
     public void updateClientWaitList(CClientInfo cClientInfo,int flag)
     {
        // first check if this client exist

        if(flag == 1) { // Add this client
           if(getClientIndex(cClientInfo.acctNum)== -1) // not exist
           {
              IBClientInfo clientInfo = new IBClientInfo();
              clientInfo.setCClientInfo(cClientInfo);
              // update client info list
              clientInfoList.addElement(clientInfo);
              // update UI
              lstWaitingList.add(clientInfo.getAcctNum() + "@" + clientInfo.getIP());
              if(clientInfoList.size()==1){
                 // set the first client as the current client
                 lstWaitingList.select(0);
                 clientProfile.setClientInfo(clientInfo);
                 clientProfile.repaint();
              }
              setMessage("client "+ clientInfo.getName() + " is placed in your waiting
list");
              if(getNumClient()==1 && bStart.isEnabled()== false) {
                 bStart.setEnabled(true);
              }
           }else {//do nothing
              return;
           }
        } else if(flag == 0){ // remove this client
           int index = getClientIndex(cClientInfo.acctNum);
           if(index >= 0)
           {
              // update client info list
              clientInfoList.removeElementAt(index);
              // update UI
```

```
            lstWaitingList.remove(cClientInfo.acctNum + "@" + cClientInfo.IP);
            setMessage("client "+ cClientInfo.name + " is remove your waiting list");
            // if this is current selected client, select the first client as current one
            if(clientProfile.getClientInfo().getAcctNum() == cClientInfo.acctNum)
            {
               if(clientInfoList.size()>0) {
                  lstWaitingList.select(0);
                  clientProfile.setClientInfo((IBClientInfo)clientInfoList.elementAt(0));
               } else {
                  clientProfile.setClientInfo(null);
               }
               clientProfile.repaint();
            }
            // enable start button
            bStart.setEnabled(true);
         }else {// do nothing
            return;
         }
      }
   }
   public String getCurrentTime(){
      String currTime;
      String language = "en";
      String country = "CANADA";
      String timezone = "EST";
      //set locale and timezone
      GregorianCalendar cal = new GregorianCalendar();
      Locale locale = new Locale (language, country);
      TimeZone tz = TimeZone.getTimeZone(timezone);
      /*set display format in specified style, locale, and timezone*/
      DateFormat myFormat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
DateFormat.LONG,locale);
      myFormat.setTimeZone(tz);
      currTime  =  myFormat.format(cal.getTime());
      return currTime;
   }

   public void quit()
   {
      if(ibr.bankMgrCorba != null && ibr.fServerDown == false) {
         ibr.bankMgrCorba.repQuit(ibr.RepID);
      }
   }

   // start to server the first customer in waiting list
   public void start(){
      if( currClientObj != null) {
         return; //do nothing
      }
      if(clientInfoList.size()<=0) {
         return;
      }
      //Send client message and enable client UI
      /*Client.ClientCorba clientObjRef = ibr.bankMgrCorba.getClientCallbackObj(
                              clientProfile.getClientInfo().getAcctNum());*/
      IBClientInfo firstClient = (IBClientInfo)clientInfoList.elementAt(0);
      Client.ClientCorba clientObjRef = ibr.bankMgrCorba.getClientCallbackObj(
                              firstClient.getAcctNum());
      if(clientObjRef != null) {
         setCurrServeringClient(clientObjRef);
         //send callback to this client
         clientObjRef.setServingRepCallback((Representative.RepCorba)ibr.repCorba);
         String msg1 = "Hi, " + ibr.RepID + "(" +
                     ibr.cRepInfo.name +
                     ") is now serving you";
         clientObjRef.setMessage(msg1);
         String msg2 = "------Start session with client " +
                  firstClient.getName()+ "-------\n";
         String msg3 = "------Start session with rep " +
                  ibr.cRepInfo.name + "-------\n";
         String msg4 = ibr.cRepInfo.name + ":" + "\t"+
```

```
                                "Hello, "  +
                                firstClient.getName()+
                                ", Can I help you?\n";
        clientObjRef.addConversationRec(msg3 + msg4);
        // update own UI
        setMessage("Start to server client " + firstClient.getAcctNum() +
                "(" + firstClient.getName()+ ")");
        addConversationRec(msg2);
        addConversationRec(msg4);
        // now the client is being served
        clientObjRef.updateStatus(2);
        // set current session start time
        setCurrStartTime(getCurrentTime());
     }
     enterServeState();
  }
  public void updateStatus(int status) {
     if(status == 0) {
        enterIdleState();
     } else if(status == 1) {
        enterServeState();
     }
  }
  public void enterIdleState(){
     // no customer is being served
     currClientObj = null;
     // clear start time
     setCurrStartTime("");
     if(parent.getNumClient()>=1) {
        bStart.setEnabled(true);
     } else {
        bStart.setEnabled(false);
     }
     bDisconnect.setEnabled(false);
     bSend.setEnabled(false);
     bClear.setEnabled(false);
     bAttach.setEnabled(false);

     lCurStartTime.setEnabled(false);
     tCurrStartTime.setEnabled(false);
     lConversationRec.setEnabled(false);
     aConversationRec.setEnabled(false);
     lDocList.setEnabled(false);
     lstDocList.setEnabled(false);
     lMsgToSend.setEnabled(false);
     aMsgToSend.setEnabled(false);
     lAttachFile.setEnabled(false);
     lstAttachFile.setEnabled(false);

     miBalance.setEnabled(false);
     miTransaction.setEnabled(false);
     miInterestRate.setEnabled(false);
     miPayBill.setEnabled(false);
     miTransMoney.setEnabled(false);
     miTermDeposit.setEnabled(false);
     miUpdateClient.setEnabled(false);
  }
  public void enterServeState(){
     lCurStartTime.setEnabled(true);
     tCurrStartTime.setEnabled(true);
     lConversationRec.setEnabled(true);
     aConversationRec.setEnabled(true);
     lDocList.setEnabled(true);
     lstDocList.setEnabled(true);
     lMsgToSend.setEnabled(true);
     aMsgToSend.setEnabled(true);
     lAttachFile.setEnabled(true);
     lstAttachFile.setEnabled(true);

     miBalance.setEnabled(true);
     miTransaction.setEnabled(true);
```

```
    miInterestRate.setEnabled(true);
    miPayBill.setEnabled(true);
    miTransMoney.setEnabled(true);
    miTermDeposit.setEnabled(true);
    miUpdateClient.setEnabled(true);

    //disable start
    bStart.setEnabled(false);
    // enable disconnect
    bDisconnect.setEnabled(true);
    bSend.setEnabled(true);
    bClear.setEnabled(true);
    bAttach.setEnabled(true);
    //set focus
    aMsgToSend.requestFocus();
}
public void disconnect(){
    // Inform server
    int clientID = clientProfile.getClientInfo().getAcctNum();
    IBClientInfo firstClient = (IBClientInfo)clientInfoList.elementAt(0);
    // inform remote client to entering disconnect state
    currClientObj.enterDisconnectState();
    String msg = "--------End session with rep " +
             ibr.cRepInfo.name+ "--------\n\n";
    currClientObj.addConversationRec(msg);
    ibr.bankMgrCorba.disconnectToRep(clientID, ibr.RepID);
    // enter idle state
    enterIdleState();
    if(parent. getNumClient()<=0) {
       bStart.setEnabled(false);
    }
    msg = "-------End session with client " +
             firstClient.getName()+ "--------\n\n";
    addConversationRec(msg);
}
public void send(){
    // send mesaage
    String msg = aMsgToSend.getText();
    if(msg != null && msg.equals("")) {
       msg = ibr.cRepInfo.name + ":\t" + msg + "\n";
       if(currClientObj != null) {
          currClientObj.addConversationRec(msg);
       }
       addConversationRec(msg);
       clear();
    }
    // send file
    for(int i=0;i<lstAttachFile.getItemCount(); i++) {
       lstDocList.add(lstAttachFile.getItem(i));
       currClientObj.addDocList(lstAttachFile.getItem(i));
    }
    lstAttachFile.removeAll();
}
public void clear() {
    aMsgToSend.setText("");
}
public void exit(){
    if(getNumClient()>0) {
       warnDlg.showDialog();
    } else {
       quit();
       System.exit(0);
    }
}
public void attach(){
    fileDlg.setMode(FileDialog.LOAD);
    //fileDlg.setDirectory(fileDlg.getDirectory() + "/attach");
    fileDlg.show();
    if(fileDlg.getFile() != null) {
       // add to fileInfo list
       /*fileInfoList.addElement(new FileInfo(fileDlg.getFile(),
```

```
                                    fileDlg.getDirectory(),
                                    false));*/
        lstAttachFile.add(fileDlg.getFile());
    }
}
public void actionPerformed(ActionEvent e)
{
    Object src = e.getSource();
    // Buttons
    if(src == bSend) {
        send();
    } else if(src == bClear) {
        clear();
    } else if(src == bDisconnect) {
        disconnect();
    } else if(src == bStart) {
        start();
    } else if(src == bExit) {
        exit();
    } else if(src == bPause) {

    } else if(src == bAttach) {
        attach();
    }
    // Menu items
    // Transaction
    else if(src == miTermDeposit) {
        termDeposit.inits();
        termDeposit.setSize(550, 200);
        termDeposit.setVisible(true);
        IBClientInfo clientInfo = (IBClientInfo)clientInfoList.elementAt(0);
        if(clientInfo != null) {
            termDeposit.setTitle("Create Term Deposit for Client " +
                            clientInfo.getAcctNum() + "(" +
                            clientInfo.getName() +")");
        }
        termDeposit.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
        });

    }else if(src == miPayBill){
        ibcpb.init();
        int h = ibcpb.getSize().height;
        int w = ibcpb.getSize().width;
        ibcpb.setBounds(0,0,w,h);
        frame_ibcpb.setLayout(null);
        frame_ibcpb.add(ibcpb);
        frame_ibcpb.setSize(w,h);
        frame_ibcpb.setVisible(true);
        IBClientInfo clientInfo = this.getCurClientInfo();

        if(clientInfo != null) {
            frame_ibcpb.setTitle("Pay bill for client " + clientInfo.getAcctNum()
                    + "(" + clientInfo.getName() + ")");
        }

        frame_ibcpb.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e)
        {
            frame_ibcpb.setVisible(false);}
        });
    }else if(src == miTransMoney){
    }else if(src ==  miUpdateClient){
    }
    // View
    else if(src == miBalance) {
        ibcb.init();
        int h = ibcb.getSize().height;
        int w = ibcb.getSize().width;
        ibcb.setBounds(0,0,w,h);
        frame_ibcb.setLayout(null);
```

```
            frame_ibcb.add(ibcb);
            frame_ibcb.setSize(w,h);
            frame_ibcb.setVisible(true);
            IBClientInfo clientInfo = this.getCurClientInfo();

            if(clientInfo != null) {
               frame_ibcb.setTitle("Account Balance for client " + clientInfo.getAcctNum()
                        + "(" + clientInfo.getName() + ")");
            }

            frame_ibcb.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
               frame_ibcb.setVisible(false);}
            });
         }else if(src == miTransaction) {
            ibct.init();
            int h = ibct.getSize().height;
            int w = ibct.getSize().width;
            ibct.setBounds(0,0,w,h);
            frame_ibct.setLayout(null);
            frame_ibct.add(ibct);
            frame_ibct.setSize(w,h);
            frame_ibct.setVisible(true);
            IBClientInfo clientInfo = this.getCurClientInfo();
            if(clientInfo != null) {
               frame_ibct.setTitle("Transactions for client " + clientInfo.getAcctNum()
                        + "(" + clientInfo.getName() + ")");
            }
            frame_ibct.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
               frame_ibct.setVisible(false);}
            });
         }
         // Tools
         else if(src ==  miInterestRate) {
         }

      }
   }

   class IBRepServiceUI extends Frame
   {
      protected TextField tStartTime;        // start time
      protected TextField tCurrStartTime;    // the time start to server current customer
      protected TextField tMessage;          // message send by server
      protected java.awt.List lstWaitingList;
      protected ClientInfoArea clientProfile;  // representative profile
      protected TextArea aConversationRec;     // conversation record
      protected java.awt.List lstDocList;
      protected TextArea aMsgToSend;            // message to send
      protected java.awt.List lstAttachFile;    //list of attached files to send
      protected Button bSend;           // send mesage
      protected Button bClear;          // clear message
      protected Button bExit;           // Exit
      protected Button bAttach;
      protected Button bPause;          //pause the service
      protected Button bStart;          //start to serve customer
      protected Button bNext;           //start to serve next customer in waiting list
      //disconnect the current customer without starting to serve next
      protected Button bDisconnect;

      protected MenuBar mb;

      protected Menu mView;
      protected MenuItem miBalance;
      protected MenuItem miTransaction;

      protected Menu mTools;
      protected MenuItem   miInterestRate;
```

```
protected Menu mTransaction;
protected MenuItem miPayBill;
protected MenuItem miTransMoney;
protected MenuItem miTermDeposit;
protected MenuItem miUpdateClient;


protected Label lStart;
protected Label lEnd;

protected Label lCurStartTime;
protected Label lMessage;
protected Label lWaitingList;
protected Label lClientProfile;

protected Label lConversationRec;
protected Label lDocList;
protected Label lMsgToSend;
protected Label lAttachFile;
protected Color cBack;
protected Color cFore;
protected Font fTitle;
protected Font fLabel;

protected FileDialog fileDlg;
public   IBRep ibr;
protected IBRepService parent;

public IBRepServiceUI(){
   // Creat text fields, areas and buttons
   tStartTime = new TextField(5);
   tCurrStartTime = new TextField(5);
   tMessage = new TextField(70);
   lstWaitingList = new java.awt.List();
   clientProfile = new ClientInfoArea();

   aConversationRec = new TextArea(10,80);
   lstDocList = new java.awt.List();
   aMsgToSend = new TextArea(10,80);
   lstAttachFile = new java.awt.List();
   bSend = new Button("Send");
   bClear = new Button("Clear");
   bExit = new Button("Exit");
   bAttach = new Button("Attach...");
   bPause = new Button("Pause");
   bStart = new Button("Start");
   bDisconnect = new Button("Disconnect");
   bNext = new Button("Next");

   mb = new MenuBar();
   mTransaction = new Menu("Transaction");
   miPayBill = new MenuItem ("Pay Bill");
   miTransMoney = new MenuItem("Transfer Money");
   miTermDeposit = new MenuItem("Term Deposit");
   miUpdateClient = new MenuItem("Update Clinet Info");

   mView = new Menu("View");
   miBalance = new MenuItem("Balance");
   miTransaction = new MenuItem("Transactions");

   mTools = new Menu("Tools");
   miInterestRate = new MenuItem("Interest Rates");

   //cBack = new Color(200,200,200);
   cBack = Color.lightGray;
   cFore = new Color(0,0,0);
   fLabel = new Font("SansSerif",Font.PLAIN,12);

   lStart = new Label("Start Time:");
   lEnd = new Label("  Stop Time:");
```

```
    lCurStartTime = new Label("Start time of current session");
    lMessage = new Label("Message: ");
    lWaitingList = new Label("Waiting list");
    lClientProfile = new Label("Client profile:");
    lConversationRec = new Label("Conversation Record:");
    lDocList = new Label("Documents sent to customer");
    lMsgToSend = new Label("Type the message you want to send in following box:");
    lAttachFile = new Label("Attach documents");
    fileDlg = new FileDialog(this);
}
public void initUI()
{
    // Menus
    setMenuBar(mb);
    mb.add(mTransaction);
    mTransaction.add(miPayBill);
    mTransaction.add(miTransMoney);
    mTransaction.add(miTermDeposit);
    mTransaction.add(miUpdateClient);

    mb.add(mView);
    mView.add(miBalance);
    mView.add(miTransaction);

    mb.add(mTools);
    mTools.add(miInterestRate);

    // label for start time and curr session start time
    lStart.setBackground(cBack);
    lStart.setForeground(cFore);
    lStart.setFont(fLabel);

    lCurStartTime.setBackground(cBack);
    lCurStartTime.setForeground(cFore);
    lCurStartTime.setFont(fLabel);

    // Message label
    lMessage.setBackground(cBack);
    lMessage.setForeground(cFore);
    lMessage.setFont(fLabel);
    // waiting list label
    lWaitingList.setBackground(cBack);
    lWaitingList.setForeground(cFore);
    lWaitingList.setFont(fLabel);

    // client profile label
    lClientProfile.setBackground(cBack);
    lClientProfile.setForeground(cFore);
    lClientProfile.setFont(fLabel);

    // conversation record label
    lConversationRec.setBackground(cBack);
    lConversationRec.setForeground(cFore);
    lConversationRec.setFont(fLabel);

    // doc list label
    lDocList.setBackground(cBack);
    lDocList.setForeground(cFore);
    lDocList.setFont(fLabel);

    lstDocList.setBackground(Color.lightGray);

    // conversation record text area
    aConversationRec.setBackground(Color.lightGray);
    aConversationRec.setEditable(false);

    // message to send label
    lMsgToSend.setBackground(cBack);
    lMsgToSend.setForeground(cFore);
    lMsgToSend.setFont(fLabel);
    // attach file label
```

```
lAttachFile.setBackground(cBack);
lAttachFile.setForeground(cFore);
lAttachFile.setFont(fLabel);
// message to send text area
aMsgToSend.setBackground(Color.white);

// Add panels in the container
setBackground(cBack);
setLayout(null);
//setBounds(0,0,650,650);

// add components to the control panel
add(lStart);
add(tStartTime);
add(lEnd);
add(lCurStartTime);
add(tCurrStartTime);
add(lMessage);
add(tMessage);
add(lWaitingList);
add(lstWaitingList);

add(lClientProfile);
add(clientProfile);
add(lConversationRec);
add(lDocList);
add(lstDocList);
add(aConversationRec);
add(lMsgToSend);
add(aMsgToSend);
add(lAttachFile);
add(lstAttachFile);
add(bSend);
add(bClear);
add(bExit);
add(bAttach);
add(bPause);
add(bStart);
add(bNext);
add(bDisconnect);

 // relocate components
lMessage.setBounds(10,65,80,20);
tMessage.setBounds(90,65,550,20);
lWaitingList.setBounds(10,90,200,20);
lstWaitingList.setBounds(10,110,230,160);
lClientProfile.setBounds(250,90,180,20);
clientProfile.setBounds(250,110,180,160);
clientProfile.setBackground(Color.gray);

lStart.setBounds(450,120,70,20);
tStartTime.setBounds(450,150,160,20);
lCurStartTime.setBounds(450,200,200,20);
tCurrStartTime.setBounds(450,230,160,20);

lConversationRec.setBounds(10,290,200,20);
aConversationRec.setBounds(10,310,400,120);
lDocList.setBounds(420,290,200,20);
lstDocList.setBounds(420,310,200,120);

lMsgToSend.setBounds(10,445,400,20);
aMsgToSend.setBounds(10,470,450,100);
lAttachFile.setBounds(470,445,160,20);
lstAttachFile.setBounds(470,470,160,100);
bSend.setBounds(180,580,80,30);
bClear.setBounds(270,580,80,30);
bAttach.setBounds(500,580, 80, 30);
//bPause.setBounds(10,620,80,30);
bStart.setBounds(180,620,80,30);
//bNext.setBounds(190,620,80,30);
bDisconnect.setBounds(270,620,80,30);
```

```
        bExit.setBounds(370,620,80,30);
    }
    public void setMessage(String s) {
        tMessage.setText(s);
    }

    public void setStartTime(String s) {
        tStartTime.setText(s);
    }
    public void setCurrStartTime(String s) {
        tCurrStartTime.setText(s);
    }
    public void addConversationRec(String s) {
        aConversationRec.append(s);
    }
}

class ClientInfoArea extends Canvas
{
    private IBClientInfo clientInfo;
    private Font font1;
    private Font font2;
    public ClientInfoArea()
    {
        super();
        clientInfo = null;
        font1 = new  Font("SansSerif",Font.ITALIC,12);
        font2 = new  Font("SansSerif",Font.PLAIN,12);
    }
    public IBClientInfo getClientInfo()
    {
        return clientInfo;
    }

    // mutator
    public void setClientInfo(IBClientInfo clientInfo)
    {
        this.clientInfo = clientInfo;
    }
    public void paint(Graphics g) {

        if(clientInfo == null ){
            g.setFont(font1);
            g.drawString("No customer is on ", 20, 20);
            g.drawString("your waiting list", 20, 40);
        } else {
            g.setFont(font1);
            g.drawString("Name:", 10, 20);
            g.setFont(font2);
            g.drawString(clientInfo.getName(), 90, 20);

            g.setFont(font1);
            g.drawString("Account No:", 10, 50);
            g.setFont(font2);
            g.drawString(Integer.toString(clientInfo.getAcctNum()), 90, 50);

            g.setFont(font1);
            g.drawString("Address:", 10, 80);
            g.setFont(font2);
            g.drawString(clientInfo.getAddress(), 10, 110);

            g.setFont(font1);
            g.drawString("Telephone:", 10, 140);
            g.setFont(font2);
            g.drawString(clientInfo.getTelephone(), 90, 140);
        }
        Rectangle rect = this.getBounds();
        g.draw3DRect(0, 0, rect.width, rect.height, true);

    }
}
```

```
class FileInfo
{
   public String name;
   public String path;
   public boolean fSend;
   //public int ID;
   public FileInfo(String name, String path, boolean fSend){
      this.name = name;
      this.path = path;
      this.fSend = fSend;
      //this.ID = ID;
   }
}
```

```
/*************************************************************
 * File name:         IBTermDeposit.java
 * Author:            Haiyu Huang
 * Date:              Oct 29, 2000
 * Descriptions:      IIB Term deposit implementation
 * It does the followings
 * 1. Create and initialize UI widgets
 * 2. Perform term deposit transaction, a tuple
 *    will be created in Table TERM_ACCOUNT if success
 *************************************************************/
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.text.*;

public class IBTermDeposit extends Frame
implements ActionListener, ItemListener
{
   // Login form variables
   private TextField tPrinciple;     // principle amount
   private TextField tInterestRate; // interest rates
   private Choice cPeriod;           // time period
   private Choice cWhichAccount;     // which account
   private Button bOK;               // OK
   private Button bCancel;           // Cancel
   private String account;
   private String period;
   private IBRepService parent;
   private float saveBalance;
   private float checkBalance;

   public IBTermDeposit(IBRepService parent)
   {
      super();
      this.parent = parent;
      // Creat text fields and buttons
      tPrinciple = new TextField(12);
      tInterestRate = new TextField(12);
      cPeriod = new Choice();
      cPeriod.setSize(120, 30);
      cPeriod.add("3 month");
      cPeriod.add("6 month");
      cPeriod.add("1 year");
      cPeriod.add("1.5 year");
      cPeriod.add("2 year");
      cPeriod.add("3 year");
      cPeriod.add("4 year");
      cPeriod.add("5 year");
      cPeriod.add("6 year");
      cPeriod.add("7 year");
      cPeriod.add("8 year");
      cPeriod.add("9 year");
      cPeriod.add("10 year");
      cPeriod.add("15 year");
      cPeriod.add("20 year");

      cPeriod.addItemListener(this);
      cWhichAccount = new Choice();
      cWhichAccount.setSize(120, 30);
      cWhichAccount.addItemListener(this);

      bOK = new Button("     OK       ");
      bOK.setSize(120, 30);
      bOK.addActionListener(this);
      bCancel = new Button("  Cancel  ");
      bCancel.setSize(120, 30);
      bCancel.addActionListener(this);
      period = "";
      account = "";
      checkBalance = 0;
      saveBalance = 0;
```

```
    }
    private boolean isFloat(String s)
    {
       for(int i=0; i<s.length(); i++) {
          if((s.charAt(i) := 46 && s.charAt(i)< 48) || s.charAt(i) > 57) {
             System.out.println(s + " is not a float number");
             return false;
          }
       }
       return true;
    }
    public String getCurrentTime(){


       String currTime;
       String language = "en";
       String country = "CANADA";
       String timezone = "EST";
       //set locale and timezone

       Locale locale = new Locale (language, country);
       TimeZone tz = TimeZone.getTimeZone(timezone);
       Calendar cal = Calendar.getInstance(tz, locale);
       //date = cal.getTime();

       return cal.get(Calendar.YEAR)+"/" + cal.get(Calendar.MONTH) + "/" +
cal.get(Calendar.DAY_OF_MONTH);
    }
    public String getNewTime(String rollValue){

       Date date = null;

       String currTime;
       String language = "en";
       String country = "CANADA";
       String timezone = "EST";
       //set locale and timezone

       Locale locale = new Locale (language, country);
       TimeZone tz = TimeZone.getTimeZone(timezone);
       Calendar cal = Calendar.getInstance(tz, locale),
       if(rollValue.equals("3 month")) {
          for(int i=0;i<3; i++) {
             cal.roll(Calendar.MONTH, true);
          }
          date = cal.getTime();
       }else if(rollValue.equals("6 month")){
          for(int i=0;i<6; i++) {
             cal.roll(Calendar.MONTH, true);
          }
          date = cal.getTime();
       }else if(rollValue.equals("1 year")){
          cal.roll(Calendar.YEAR, true);
          date = cal.getTime();
       }else if(rollValue.equals("1.5 year")){
          for(int i=0;i<18; i++) {
             cal.roll(Calendar.MONTH, true);
          }
          date = cal.getTime();
       }else if(rollValue.equals("2 year")){
          for(int i=0;i<2; i++) {
             cal.roll(Calendar.YEAR, true);
          }
          date = cal.getTime();
       }else if(rollValue.equals("3 year")){
          for(int i=0;i<3; i++) {
             cal.roll(Calendar.YEAR, true);
          }
          date = cal.getTime();
       }else if(rollValue.equals("4 year")){
          for(int i=0;i<4; i++) {
```

```
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }else if(rollValue.equals("5 year")){
         for(int i=0;i<5; i++) {
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }else if(rollValue.equals("6 year")){
         for(int i=0;i<6; i++) {
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }else if(rollValue.equals("7 year")){
         for(int i=0;i<7; i++) {
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }else if(rollValue.equals("8 year")){
         for(int i=0;i<8; i++) {
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }else if(rollValue.equals("9 year")){
         for(int i=0;i<9; i++) {
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }else if(rollValue.equals("10 year")){
         for(int i=0;i<10; i++) {
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }else if(rollValue.equals("15 year")){
         for(int i=0;i<15; i++) {
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }else if(rollValue.equals("20 year")){
         for(int i=0;i<20; i++) {
            cal.roll(Calendar.YEAR, true);
         }
         date = cal.getTime();
      }

      return cal.get(Calendar.YEAR)+"/" + cal.get(Calendar.MONTH) + "/" +
cal.get(Calendar.DAY_OF_MONTH);
   }

   public void initAccount(){
      CAccount[]  cAccounts = null;
      IBClientInfo clientInfo = parent.getCurClientInfo();
      if(clientInfo != null) {
         cAccounts = parent.ibr.bankMgrCorba.getAccounts(clientInfo.getAcctNum());
      }
      cWhichAccount.removeAll();
      if(cAccounts != null) {
         for(int i =0; i<cAccounts.length; i++) {
            if(i>=1 && cAccounts[i].type == cAccounts[0].type){
               break;//following data is not valid (same as the first one)
            }
            if(cAccounts[i].type == 'S') { // Saving account
               cWhichAccount.add("Saving Account");
               saveBalance = cAccounts[i].balance;
            }
            if(cAccounts[i].type == 'C') {// Check account
               cWhichAccount.add("Check Account");
               checkBalance = cAccounts[i].balance;
            }
            // more account can be added......
         }
```

```
    }
}

public void inits()
{
    //Color cBack = new Color(200,200,200);
    Color cBack = Color.lightGray;
    Color cFore = new Color(0,0,0);
    Font fInfo = new Font("SansSerif",Font.PLAIN,12);
    Font fTf = new Font("SansSerif",Font.PLAIN,12);

    // textfield for username and password
    tPrinciple.setFont(fTf);
    tInterestRate.setFont(fTf);

    // label for principle amount
    Label lPrinciple = new Label("Principle Amount:");
    lPrinciple.setBackground(cBack);
    lPrinciple.setForeground(cFore);
    lPrinciple.setFont(fInfo);

    // label for interest rate
    Label lInterestRate = new Label("Interest Rate:  ");
    lInterestRate.setBackground(cBack);
    lInterestRate.setForeground(cFore);
    lInterestRate.setFont(fInfo);

    // label for time period
    Label lPeriod = new Label("Time Period:      ");
    lPeriod.setBackground(cBack);
    lPeriod.setForeground(cFore);
    lPeriod.setFont(fInfo);

    // label for which account
    Label lWhichAccount = new Label("Principle taken from:");
    lWhichAccount.setBackground(cBack);
    lWhichAccount.setForeground(cFore);
    lWhichAccount.setFont(fInfo);

    // Panel for principle amount
    Panel pPrinciple = new Panel();
    pPrinciple.setLayout(new GridLayout(2,1));
    pPrinciple.add(lPrinciple);
    pPrinciple.add(tPrinciple);

    // Panel for interest rate
    Panel pInterestRate = new Panel();
    pInterestRate.setLayout(new GridLayout(2,1));
    pInterestRate.add(lInterestRate);
    pInterestRate.add(tInterestRate);

    // Panel for time period
    Panel pPeriod = new Panel();
    pPeriod.setLayout(new GridLayout(2,1));
    pPeriod.add(lPeriod);
    pPeriod.add(cPeriod);

    // Panel for account
    Panel pWhichAccount = new Panel();
    pWhichAccount.setLayout(new GridLayout(2,1));
    pWhichAccount.add(lWhichAccount);
    pWhichAccount.add(cWhichAccount);

    // Panel for panels
    Panel pPanel = new Panel();
    pPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
    pPanel.add(pPrinciple);
    pPanel.add(pInterestRate);
    pPanel.add(pPeriod);
    pPanel.add(pWhichAccount);
```

```
    // Panel for buttons
    Panel pButtons = new Panel();
    pButtons.setLayout(new FlowLayout(FlowLayout.LEFT));
    pButtons.add(bOK);
    pButtons.add(bCancel);


    setLayout(new FlowLayout(FlowLayout.LEFT));
    setBackground(cBack);
    add(pPanel);
    add(pButtons);

    initAccount();
    tPrinciple.requestFocus();
}

public boolean isOverDrafted(String acctType){

    float principle  = (Float.valueOf(tPrinciple.getText())).floatValue();

    if(acctType.equals("Check Account")) {
       return principle > checkBalance;
    }else if(acctType.equals("Saving Account")) {
       return principle > saveBalance;
    }
    return true;
}
public boolean doTermDeposit(String acctType, float principle, float interestRate,
                        String startDate, String matureDate){
    boolean fSuccess = false;
    IBClientInfo clientInfo = parent.getCurClientInfo();
    if(clientInfo != null) {
       int acctNum = clientInfo.getAcctNum();
       if(acctType.equals("Check Account")) {
          acctType = "C";
       } else if(acctType.equals("Saving Account")){
          acctType = "S";
       }
       parent.ibr.bankMgrCorba.termDeposit(acctNum, acctType, principle, interestRate,
                                           startDate, matureDate);
       fSuccess = true;
    }

    return fSuccess;
}
public void actionPerformed(ActionEvent e)
{
    Object src = e.getSource();
    if(src == bOK) {
       if(!isFloat(tPrinciple.getText()) || !isFloat(tInterestRate.getText())){
          tPrinciple.setText("");
          tInterestRate.setText("");
          cPeriod.select(0);
          parent.ibr.msgDlg.setTitle("Error");
          parent.ibr.msgDlg.setMessage("Invalid Input!");
          parent.ibr.msgDlg.showDialog();
          return;
       }
       String acctType = cWhichAccount.getSelectedItem();
       if( isOverDrafted(acctType) ){
          parent.ibr.msgDlg.setTitle("Error");
          parent.ibr.msgDlg.setMessage("Not enough balance!");
          parent.ibr.msgDlg.showDialog();
          return;
       }

       period = cPeriod.getSelectedItem();
       String startDate = getCurrentTime();
       String matureDate= getNewTime(period);
       System.out.println(startDate);
       System.out.println(matureDate);
```

```
            float principle = (Float.valueOf(tPrinciple.getText())).floatValue();
            float interestRate = (Float.valueOf(tInterestRate.getText())).floatValue();
            if(doTermDeposit(acctType, principle, interestRate, startDate, matureDate)) {
               parent.ibr.msgDlg.setTitle("Success");
               parent.ibr.msgDlg.setMessage("Transaction completed:");
               parent.ibr.msgDlg.showDialog();
            } else {
               parent.ibr.msgDlg.setTitle("Error");
               parent.ibr.msgDlg.setMessage("Error in transaction:");
               parent.ibr.msgDlg.showDialog();
            }

            setVisible(false);
         }
         if(src == bCancel) {
            setVisible(false);
         }
      }

      public void itemStateChanged(ItemEvent e)
      {
         Object src = e.getSource();
         if(src == cPeriod) {
            period = (String)e.getItem();
         }else if(src == cWhichAccount) {
            account = (String)e.getItem();
         }
      }
   }
```

B.3 Source Code for Server Program

```
/*************************************************************
 * File name:        IBServer.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB server main program
 * It dose the followings:
 * 1. Create and initialize server UI widgets
 * 2. Initialize corba
 * 3. Create Database manager and load system database
 * 4. Create server side corba object Bank Manager and
 *    export it to ORB bus
 *************************************************************/
//import org.omg.CosNaming.*;
import java.util.*;
import java.text.*;
import java.awt.*;
import java.awt.event.*;

import org.omg.PortableServer.*;
import com.inprise.vbroker.PortableServerExt.BindSupportPolicyValue;
import com.inprise.vbroker.PortableServerExt.BindSupportPolicyValueHelper;

public class IBServer
{
   private IBDBManager dbManager;
   private IBBankMgrCorbaImpl bankMgrCorbaImpl;
   private IBServerUI serverUI;

   public IBServer()
   {
      dbManager = null;
      bankMgrCorbaImpl = null;
      serverUI = new IBServerUI(this);
   }
   IBDBManager getDBManager() {
      return dbManager;
   }
   public void setServerUIDBManager() {
      serverUI.setDBManager(dbManager);
   }
   public void initDatabase() throws Exception
   {
      dbManager = new IBDBManager();
      dbManager.connect();
      dbManager.loadInitBankDatabase();
      dbManager.closeConnection();
   }
   public void updateSession(String item, int flag)
   {
      serverUI.updateSession(item, flag);
   }
   public void initCorba()
   {
      try
      {
         boolean fBrowser = false;
         // Initialize the ORB.
         serverUI.addOutput("Initializing ORB...\n");
         String[] args = null;
         org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

         // Initialize the BOA
         // get a reference to the root POA
         POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
         // Create policies for our persistent POA
```

```
         org.omg.CORBA.Any any = orb.create_any();
         BindSupportPolicyValueHelper.insert(any, BindSupportPolicyValue.BY_INSTANCE);
         org.omg.CORBA.Policy bsPolicy =
orb.create_policy(com.inprise.vbroker.PortableServerExt.BIND_SUPPORT_POLICY_TYPE.value,
any);
         org.omg.CORBA.Policy[] policies = null;
         if(fBrowser == false) {
            policies = new org.omg.CORBA.Policy[]
                     {rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT),
bsPolicy };
         } else {
            policies = new org.omg.CORBA.Policy[] {
                     rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)};
         }
         // Create myPOA with the right policies
         POA myPOA = rootPOA.create_POA( "bank_agent_poa", rootPOA.the_POAManager(),
                                         policies );
         // Create the servant
         bankMgrCorbaImpl = new IBBankMgrCorbaImpl(dbManager, this);
         // Decide on the ID for the servant
         byte[] managerId = "BankManager".getBytes();
         // Activate the servant with the ID on myPOA
         myPOA.activate_object_with_id(managerId, bankMgrCorbaImpl);

         rootPOA.the_POAManager().activate();

         System.out.println(myPOA.servant_to_reference(bankMgrCorbaImpl) +
                            " is ready.");
         serverUI.addOutput("Corba object Bank manager BankManager is created\n");
         // Export the bank manager corba object
         //boa.obj_is_ready(bankMgrCorba);
         System.out.println("Bank manager waiting for requests");
         serverUI.addOutput("Bank manager waiting for requests\n");
         serverUI.addOutput("Server is ready...\n\n");
         // Ready to service requests
         //boa.impl_is_ready();
         // Wait for incoming requests
         orb.run();
      } catch(Exception e) {
         System.err.println(e);
      }

   }
   public void initUI()
   {
      serverUI.init();
      serverUI.setSize(500,600);
      serverUI.setVisible(true);
      serverUI.setTitle("IIB Server");

   }
   public void serverDown(){
      bankMgrCorbaImpl.serverDown();
   }
   static public void main(String[] args)
   {
      IBServer server = new IBServer();
      try {
         server.initUI();
         server.initDatabase();
         server.setServerUIDBManager();
         server.initCorba();
      }catch(Exception e) {
         System.err.println(e);
      }
   }
}
```

```
/*************************************************************
 * File name:        IBServerUI.java
 * Author:           Haiyu Huang
 * Date:             Oct 29, 2000
 * Descriptions:     IIB server ui implementation
 *                   It creates and initializes server UI widgets
 *************************************************************/
import java.awt.*;
import java.awt.event.*;
import java.net.*;

public class IBServerUI extends Frame
implements ActionListener, WindowListener
{
   // list of all regitsered clients and reps sesssions
   private  List  lstSessionList;
   // Server output
   private TextArea aOutput;

   private Button bShutDown;      // shut down server
   private Button bDatabase;      // clear out put

   private Label lSessionList;
   private Label lOutput;

   private Color cBack;
   private Color cFore;
   private Font fTitle;
   private Font fLabel;

   private IBDatabase database;

   private IBServer server;

   public IBServer getServer() {
      return server;
   }
   public void setDBManager(IBDBManager dbManager) {
      database.setDBManager(dbManager);
   }
   public IBServerUI(IBServer server){

      lstSessionList = new List();
      aOutput = new TextArea(10,80);
      bShutDown = new Button("Shut Down");
      bShutDown.addActionListener(this);
      bDatabase = new Button("Database");
      bDatabase.addActionListener(this);

      //cBack = new Color(200,200,200);
      cBack = Color.lightGray;
      cFore = Color.black;
      fLabel = new Font("SansSerif",Font.PLAIN,12);
      lSessionList = new Label("Sessions");
      lOutput = new Label("Outputs:");
      this.server = server;
      database = new IBDatabase();
   }
   public void init()
   {
      // Session label
      lSessionList.setBackground(cBack);
      lSessionList.setForeground(cFore);
      lSessionList.setFont(fLabel);

      // Outputs label
      lOutput.setBackground(cBack);
      lOutput.setForeground(cFore);
      lOutput.setFont(fLabel);

      // Output
```

```
    aOutput.setBackground(Color.lightGray);
    aOutput.setEditable(false);

    setBackground(cBack);
    setLayout(null);
    // add components to the control panel
    add(lSessionList);
    add(lstSessionList);
    add(lOutput);
     add(aOutput);
    add(bShutDown);
    add(bDatabase);

    // relocate components
    lSessionList.setBounds(30,40,400,20);
    lstSessionList.setBounds(30,60,400,200);
    lOutput.setBounds(30,290,400,20);
    aOutput.setBounds(30,310,400,200);
    bShutDown.setBounds(180,550,80,30);
    bDatabase.setBounds(270,550,80,30);

     // Register listerner
    bShutDown.addActionListener(this);
    bDatabase.addActionListener(this);
    addWindowListener(this);
}
public void actionPerformed(ActionEvent e)
{
    Object src = e.getSource();
    if(src == bDatabase) {
       database.init();
       database.setSize(700,680);
       database.setVisible(true);
       database.setTitle("IIB Database for Demo purpose");
       database.addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e)
       {
          database.setVisible(false);;}
       });
    } else if(src == bShutDown) {
       server.serverDown();
       System.exit(0);
    }
}

public void windowClosed(WindowEvent event) { }
public void windowDeiconified(WindowEvent event) { }
public void windowIconified(WindowEvent event) { }
public void windowActivated(WindowEvent event) { }
public void windowDeactivated(WindowEvent event) { }
public void windowOpened(WindowEvent event) { }
public void windowClosing(WindowEvent event)
{
    server.serverDown();
    System.exit(0);
}
public boolean sessionListContains(String item) {
    String[] allItems = lstSessionList.getItems();
    boolean fContain = false;
    for(int i=0; i<allItems.length; i++) {
       if(allItems[i].equals(item)) {

          fContain = true;
          break;
       }
    }
    return fContain;
}
public void updateSession(String session, int flag){
    if(flag == 1) { //add
       if(!sessionListContains(session))
```

```
            {
               lstSessionList.add(session);
            }
         }else if(flag == 0) {//remove
            if(sessionListContains(session))
            {
               lstSessionList.remove(session);
            }
         }else if(flag == 2) {//modify
         }else {//default, do nothing
         }

      }
      public void addOutput(String s)
      {
         aOutput.append(s);
      }

}
```

```
/*************************************************************
 * File name:        IBBankMgrCorbaImpl.java
 * Author:           Haiyu Huang
 * Date:             Nov 3, 2000
 * Descriptions:     IIB server side corba object Bank manager imlementation,
 * this class implements the server main logic
 * It dose the followings:
 * 1. Manage two lists of callback object reference (clients and reps)
 * 2. Register/quit clients and representatives
 * 3. Connect/disconnect clients to specific representative (put in waiting list)
 * 4. Access system database on hehalf of clients and representatives
 *************************************************************/
import java.util.*;

public class IBBankMgrCorbaImpl extends Server.BankMgrCorbaPOA
{
   // Parent
   private IBServer  server;
   private IBDBManager dbManager = null;
   // List of all registered clients
   private Vector clientList;
   // List of all registered representatives
   private Vector repList;

   public IBBankMgrCorbaImpl(IBDBManager dbManager, IBServer parent)
   {
      this.dbManager = dbManager;
      this.server = parent;
      System.out.println("Bank manager corba object created");
      clientList = new Vector();
      repList = new Vector();
   }
   // private util functions for local usages
   private boolean IsInteger(String s)
   {
      for(int i=0; i<s.length(); i++) {
         if(s.charAt(i)< 48 || s.charAt(i) > 57) {
            System.out.println(s + " is not a integer");
            return false;
         }
      }
      return true;
   }
   private int getRegisteredRepIndex(int repID){
      int index = -1;
      for(int i= 0; i<repList.size(); i++) {
         RegisteredRep regRep = (RegisteredRep) repList.elementAt(i);
         if(regRep.repInfo.getID() == repID)// found
         {
            index = i;
            return index;
         }
      }
      return index;
   }
   private int getRegisteredClientIndex(int clientID){
      return 0;
   }
   private RegisteredRep getRegisteredRepFromID(int repID)
   {
      RegisteredRep regRep = null;
      for(int i= 0; i<repList.size(); i++) {
         regRep = (RegisteredRep) repList.elementAt(i);
         if(regRep.repInfo.getID() == repID)// found
         {
            return regRep;
         }
      }
      return null;
   }
   private RegisteredClient getRegisteredClientFromID(int clientID)
```

```
    {
        RegisteredClient regClient = null;
        for(int i= 0; i<clientList.size(); i++) {
            regClient = (RegisteredClient) clientList.elementAt(i);
            if(regClient.clientInfo.getAcctNum() == clientID)// found
            {
                return regClient;
            }
        }
        return null;
    }
    private int getNumClientInWaitList(){
        int result = 0;
        for(int i=0; i<repList.size(); i++) {
            RegisteredRep regRep = (RegisteredRep) repList.elementAt(i);
            if(regRep != null) {
                result += regRep.getNumClients();
            }
        }
        return result;
    }
    // End of util funtions

    // Remote utility for client and rep
    public CClientInfo  getClientInfo(int clientID){
        CClientInfo clientInfo = null;
        RegisteredClient regClient = getRegisteredClientFromID(clientID);
        if(regClient != null) {
            clientInfo = regClient.clientInfo.getCClientInfo();
            return clientInfo;
        } else {
            return null;
        }
    }

    public CRepInfo getRepInfo(int repID){
        RegisteredRep regRep = getRegisteredRepFromID(repID);
        if(regRep != null) {
            return regRep.repInfo.getCRepInfo();
        } else {
            return null;
        }
    }

    // update the waiting list pos of all clients behind the clientID if this client is
quit/disconnected
    private void updateWaitListPos(RegisteredRep regRep, int clientID, int step) {
        int pos = 0; //the pos of this client
        for(int i = 0; i<regRep.clientList.size(); i--){
            RegisteredClient regClient = (RegisteredClient)regRep.clientList.elementAt(i);
            if(regClient.clientInfo.getAcctNum() == clientID) { // found
                pos = i;
                break;
            }
        }
        // change the pos of all clients behind
        for(int i = pos; i<regRep.clientList.size(); i++){
            RegisteredClient regClient = (RegisteredClient)regRep.clientList.elementAt(i);
            if(regClient != null) {
                if(regClient.clientObjRef != null) {
                    regClient.clientObjRef.updateWaitListPos(step);
                }
            }
        }
    }
    public int registerClient(Client.ClientCorba clientObjRef)
    {
        int clientIndex = -1;
        String acctNum = clientObjRef.getAcctNum();
        String password = clientObjRef.getPassword();
        // Check if acctNum is a integer
```

```
      if(!IsInteger(acctNum)){
         return -2;
      }
      // Check if this client is already registered
      if( getRegisteredClientFromID(Integer.parseInt(acctNum))!= null ){
         return 0;
      }
      try {
         CAccount[] cAccounts = null;
         dbManager.connect();
         cAccounts = dbManager.lookupAccount(acctNum, password);
         dbManager.closeConnection();
         if(cAccounts != null)
         {
            IBClientInfo clientInfo = new IBClientInfo();
            clientInfo.setCAccounts(cAccounts);
            clientInfo.setAcctNum(Integer.parseInt(acctNum));
            clientInfo.setIP(clientObjRef.getClientIP());
            // fill remained information
            dbManager.connect();
            dbManager.lookupCustomer(acctNum, clientInfo);
            dbManager.closeConnection();

            RegisteredClient registeredClient = new RegisteredClient(clientObjRef,
clientInfo);
            clientList.addElement(registeredClient);
            clientIndex = clientInfo.getAcctNum();
            // Since the client is registered, server must send him all the necessary
infomation
            // first tell him information about customer service
            // List of representative currrently on service
            int numReps = repList.size();
            if(repList.size()> 0) {
               CRepInfo[] repInfoList = new CRepInfo[numReps] ;
               for(int i=0;i<numReps; i++)
               {
                  RegisteredRep rRep = (RegisteredRep)repList.elementAt(i);
                  repInfoList[i] = rRep.repInfo.getCRepInfo();
               }

               clientObjRef.setRepList(repInfoList);
               // clean up
               repInfoList = null;
            }
            // number of clients
            //clientObjRef.setNumClient(Integer.toString(getNumClientInWaitList()));
            // Update server session information
            server.updateSession(acctNum + "3" + clientInfo.getIP(),1);
         }
      }catch (Exception e) {
         System.err.println(e);

      }
      return clientIndex;
   }

   public int registerRep(Representative.RepCorba repObjRef)
   {
      int repIndex = -1;
      String  repID = repObjRef.getRepID();
      String password = repObjRef.getPassword();
      // Check if repID is a integer
      if(!IsInteger(repID)){
         return -2;
      }
      try {
         dbManager.connect();
         IBRepInfo repInfo = dbManager.lookupRepresentative(repID, password);
         dbManager.closeConnection();
         if(repInfo != null)
         {
```

```
            repInfo.setNumClient(0);
            repInfo.setEstimateWaitTime(0);
            repInfo.setIP(repObjRef.getRepIP());
            RegisteredRep registeredRep = new RegisteredRep(repObjRef, repInfo);
            repList.addElement(registeredRep);
            repIndex = repInfo.getID();
            // Update client UI related to rep info
            for(int i =0; i<clientList.size(); i++) {
               RegisteredClient registeredClient = (RegisteredClient)
clientList.elementAt(i);
               // add this rep to all registered client UI
               registeredClient.clientObjRef.updateRepList(repInfo.getCRepInfo(), 1);
               // broadcast message to all registered clients
               registeredClient.clientObjRef.setMessage("Representative " +
repInfo.getName() + " ready for service");
            }
            // Since the rep is registered, server must send him all the necessary
infomation
            // Update server session info
            server.updateSession(repID + "@" + repInfo.getIP(), 1);
         }
      }catch (Exception e) {
         System.err.println(e);
      }
      return repIndex;
   }

   public void repQuit(int repID)
   {
      // first remove this rep from registered rep list
      boolean fSuccess = false;
      RegisteredRep registeredRep = null;
      for(int i= 0; i<repList.size(); i++) {
         registeredRep = (RegisteredRep) repList.elementAt(i);
         if(registeredRep.repInfo.getID() == repID)// found
         {
            repList.removeElementAt(i);
            System.out.println("Rep "  + repID + " is removed from rep list");

            fSuccess = true;
            break;
         }
      }
      if(!fSuccess) {
         System.out.println("Error in removing " +  repID + " from rep list");
      }

      // Send message to all registered clients and reps and update correspondent info
      for(int i =0; i<clientList.size(); i++) {
            RegisteredClient registeredClient = (RegisteredClient)
clientList.elementAt(i);
            // remove this rep from all registered client UI
   registeredClient.clientObjRef.updateRepList(registeredRep.repInfo.getCRepInfo(), 0);
            // broadcast message to all registered clients
            registeredClient.clientObjRef.setMessage("Representative " +
registeredRep.repInfo.getName() + " has quitted");
      }
      // Tell the clients on this rep waiting list
      for(int i = 0; i<registeredRep.clientList.size(); i++){
         RegisteredClient regClient =
(RegisteredClient)registeredRep.clientList.elementAt(i);
         if(regClient != null) {
            if(regClient.clientObjRef != null) {
               regClient.clientObjRef.removeConnectedRep(repID);
            }
         }
         // Set this client's rep = null (data maintain on serve side)
         for(i=0; i<clientList.size(); i++){
            RegisteredClient registeredClient = (RegisteredClient)
clientList.elementAt(i);
```

```
            if(registeredClient.clientInfo.getAcctNum() ==
regClient.clientInfo.getAcctNum()) {//found
               // changed it
               registeredClient.rep = null;
               // put it back
               clientList.setElementAt(registeredClient, i);
               break;
            }
         }

      }

      // Update server session info
      server.updateSession(repID + "@" + registeredRep.repInfo.getIP(), 0);

      // clean up
      registeredRep.repInfo = null;
      registeredRep.clientList = null;
      registeredRep = null;
   }
   public void clientQuit(int clientID)
   {
      // first remove this client from registered client list
      boolean fSuccess = false;
      RegisteredClient registeredClient = null;
      String clientName = "";
      for(int i= 0; i<clientList.size(); i++) {
         registeredClient = (RegisteredClient) clientList.elementAt(i);
         if(registeredClient.clientInfo.getAcctNum() == clientID)// found
         {
            clientList.removeElementAt(i);
            System.out.println("Client " + clientID + " is removed from client list");
            fSuccess = true;
            break;
         }
      }

      if(!fSuccess) {
         System.out.println("Error in removing " +  clientID + " from client list");
      }
      clientName = registeredClient.clientInfo.getName();
      // Update server session information
      server.updateSession(clientID + "@" + registeredClient.clientInfo.getIP(),0);
      // if this client is on rep's waiting list, remove it from that perticular rep
      if(registeredClient.rep != null) {
         System.out.println("removing client from rep list");
         if(registeredClient.rep.repObjRef != null) {
            registeredClient.rep.repObjRef.updateClientWaitList(
               registeredClient.clientInfo.getCClientInfo(),
               0);
         }
         // Remove this client from rep's waiting list
         registeredClient.rep.removeClient(registeredClient);
         // now this rep is in idle state
         registeredClient.rep.repObjRef.updateStatus(0);
         // update the registered rep information(number of client decrease by 1
         int repID = registeredClient.rep.repInfo.getID();
         int index = getRegisteredRepIndex(repID);
         if(index >=0) {
            // change it
            registeredClient.rep.repInfo.decrementWaitingList();
            // put it back
            repList.setElementAt(registeredClient.rep, index);
         }
         CRepInfo cRepInfo = registeredClient.rep.repInfo.getCRepInfo();
         // broadcast to registered clients to midified the rep info
         for(int i =0; i<clientList.size(); i++) {
            registeredClient = (RegisteredClient) clientList.elementAt(i);
            registeredClient.clientObjRef.updateRepList(cRepInfo, 2); //2 mean modify
            // broadcast message to all registered clients
            registeredClient.clientObjRef.setMessage("Client " +
```

```
                                                clientID +
                                                "(" +
                                                clientName +
                                                ") has quited");
            }

            // Update the waiting list order of the clients hehind this client(advance by 1)
            updateWaitListPos(registeredClient.rep, clientID, 1);
        }

    }
    public void connectToRep(int clientID, int repID){

        if(repID == -1) {
            // assign a rep based on load balance
        } else { // Client chosen representative
            // place this client in this rep waiting list
            RegisteredRep regRep = getRegisteredRepFromID(repID);
            RegisteredClient regClient = getRegisteredClientFromID(clientID);
            regRep.addNewClient(regClient);
            // set client's serving rep
            regClient.rep = regRep;
            // now this client is on waiting list
            regClient.clientObjRef.updateStatus(1);
            // update the registered rep information(number of client increase by 1)
            int index = getRegisteredRepIndex(repID);
            // change it
            regRep.repInfo.incrementWaitingList();
            // put it back
            repList.setElementAt(regRep, index);
            CRepInfo cRepInfo = regRep.repInfo.getCRepInfo();
            // broadcast to registered clients to midified the rep info
            for(int i =0; i<clientList.size(); i++) {
                RegisteredClient registeredClient = (RegisteredClient)
clientList.elementAt(i);
                registeredClient.clientObjRef.updateRepList(cRepInfo, 2);
                // broadcast message to all registered clients
                //registeredClient.clientObjRef.setMessage("Representative " + cRepInfo.name
+ " has new client");
            }
            try{
                // send this full client information to the rep
                regRep.repObjRef.updateClientWaitList(
                    regClient.clientInfo.getCClientInfo(),
                    1);
            }catch (Exception e) {
                System.err.println(e);
            }
        }
    }

    public void disconnectToRep(int clientID, int repID){

        // remove this client from this rep waiting list
        RegisteredRep regRep = getRegisteredRepFromID(repID);
        RegisteredClient regClient = getRegisteredClientFromID(clientID);
        regRep.removeClient(regClient);
        // set client's serving rep to null
        regClient.rep = null;
        // now this client is not connected to any rep
        regClient.clientObjRef.updateStatus(0);
        // now this rep is in idle state
        regRep.repObjRef.updateStatus(0);
        // update the registered rep information(number of client decrease by 1
        int index = getRegisteredRepIndex(repID);
        // change it
        regRep.repInfo.decrementWaitingList();
        // put it back
        repList.setElementAt(regRep, index);
        CRepInfo cRepInfo = regRep.repInfo.getCRepInfo();
        // broadcast to registered clients to midified the rep info
```

```
        for(int i =0; i<clientList.size(); i++) {
            RegisteredClient registeredClient = (RegisteredClient) clientList.elementAt(i);
            registeredClient.clientObjRef.updateRepList(cRepInfo, 2);
            // broadcast message to all registered clients
            registeredClient.clientObjRef.setMessage("Client " +
                                        registeredClient.clientInfo.getAcctNum() +
                                        "(" +
                                        registeredClient.clientInfo.getName() +
                                        ") is disconnected");
        }
        // update the waiting list order of the clients hehind this client(advance by 1)
        updateWaitListPos(regRep, clientID, 1);
        // inform remote rep to remove this client from waiting list
        regRep.repObjRef.updateClientWaitList(
              regClient.clientInfo.getCClientInfo(),
              0);
    }

    public Client.ClientCorba getClientCallbackObj(int clientID){
        RegisteredClient regClient = getRegisteredClientFromID(clientID);
        if(regClient != null) {
            Client.ClientCorba clientObjRef = regClient.clientObjRef;
            return clientObjRef;
        }else {
            return null;
        }

    }
    // broadcast messagaes to all registered client and rep before shutdown
    public void serverDown(){
        // Send message to all registered clients and reps and update correspondent info
        for(int i =0; i<clientList.size(); i++) {
            RegisteredClient registeredClient = (RegisteredClient) clientList.elementAt(i);
            // set server down flag
            registeredClient.clientObjRef.serverDown();
            // broadcast message to all registered clients
            registeredClient.clientObjRef.setMessage("Server is down");
        }
        for(int i =0; i<repList.size(); i++) {
            RegisteredRep registeredRep = (RegisteredRep) repList.elementAt(i);
            // set server down flag
            registeredRep.repObjRef.serverDown();
            // broadcast message to all registered reps
            registeredRep.repObjRef.setMessage("Server is down");
        }
    }

    // Database access functions
    public String findBillAcctNum(int bankAcctNum, String billName){
        String billAcctNum = null;
        try {
            dbManager.connect();
            billAcctNum = dbManager.findBillAcctNum(Integer.toString(bankAcctNum),
billName);
            if(billAcctNum == null) {// corba object method can not return null value
                billAcctNum = "";
            }
            dbManager.closeConnection();
        } catch (Exception e) {
            System.err.println(e);
        }
        return billAcctNum;
    }
    public synchronized CBillInfo getBillInfo(String acctNum){
        CBillInfo cBillInfo = new CBillInfo();
        if(cBillInfo != null) {
            try {
                dbManager.connect();
                dbManager.lookupBillAccount(acctNum, cBillInfo);
                dbManager.closeConnection();
            } catch (Exception e) {
```

```
            System.err.println(e);
         }
      }
      return cBillInfo;
   }
   public CAccount[] getAccounts(int clientID) {
      CAccount[] cAccounts = null;
      RegisteredClient regClient = getRegisteredClientFromID(clientID);
      if(regClient != null){
         if(regClient.clientInfo != null) {
            cAccounts = regClient.clientInfo.getCAccount();
            if(cAccounts != null) {
               for(int i=0; i<5; i++) {
                  if(cAccounts[i] == null) {
                     cAccounts[i] = cAccounts[0];//avoid crash for corba
                  }
               }
               return cAccounts;
            }
         }
      }
      return null;
   }
   public String getTransaction(int clientID, String acctType){
      String acctNum = Integer.toString(clientID);
      String trans = null;
      try {
            dbManager.connect();
            trans = dbManager.viewAccountTranaction(acctNum, acctType, null, null);
            dbManager.closeConnection();
      } catch (Exception e) {
            System.err.println(e);
      }
      if(trans != null) {
         return trans;
      } else {
         return "";
      }
   }
   public void setAccounts(int clientID, CAccount[] cAccounts) {
      if(cAccounts != null) {
         for(int i=0; i<5; i++) {
            if(cAccounts[i] == null) {
               cAccounts[i] = cAccounts[0];//avoid crash for corba
            }
         }
      }
      RegisteredClient regClient = getRegisteredClientFromID(clientID);
      if(regClient != null){
         if(regClient.clientInfo != null) {
            regClient.clientInfo.setCAccounts(cAccounts);
         }
      }
   }
   public void payBill(String acctNum, float amount, int bankAcctNum, String
bankAcctType) {
      String amountToPay = Float.toString(amount);
      String strBankAcctNum = Integer.toString(bankAcctNum);
      CAccount[] cAccounts = null;
      try {
            dbManager.connect();
            dbManager.payBillAccount(acctNum, amountToPay);
            dbManager.updateAccountBalance(strBankAcctNum, bankAcctType, "-" +
amountToPay);
            dbManager.createTransaction(strBankAcctNum, bankAcctType, "IIB BELL Bill",
"DEBIT", amountToPay);
            // get updated accounts
            cAccounts = dbManager.lookupAccount(strBankAcctNum, null);
            dbManager.closeConnection();

      } catch (Exception e) {
```

```
            System.err.println(e);
        }
        if(cAccounts != null) {// update accounts
           setAccounts(bankAcctNum, cAccounts);
        }
    }
    public void termDeposit(int acctNum, String acctType, float capital, float
interestRate,
                        String startDate, String matureDate) {
        String strAcctNum = Integer.toString(acctNum);
        String strCapital = Float.toString(capital);
        CAccount[] cAccounts = null;
        try {
            dbManager.connect();
            dbManager.createTermDeposit(strAcctNum, capital, interestRate, startDate,
matureDate);
            dbManager.updateAccountBalance(strAcctNum, acctType, "-" + capital);
            dbManager.createTransaction(strAcctNum, acctType, "IIB WITHDRAW", "DEBIT",
strCapital);
            // get updated accounts
            cAccounts = dbManager.lookupAccount(strAcctNum, null);
            dbManager.closeConnection();

        } catch (Exception e) {
            System.err.println(e);
        }
        if(cAccounts != null) {// update accounts
           setAccounts(acctNum, cAccounts);
        }
    }
}

class RegisteredRep
{
    public Representative.RepCorba repObjRef;
    public IBRepInfo repInfo;
    // This rep waiting list
    public Vector clientList;

    public RegisteredRep()
    {
        repObjRef = null;
        repInfo = null;
        clientList = new Vector();
    }

    public RegisteredRep(Representative.RepCorba repObjRef, IBRepInfo repInfo)
    {
        this.repObjRef = repObjRef;
        this.repInfo = repInfo;
        clientList = new Vector();
    }

    public int getNumClients()
    {
        return clientList.size();
    }
    public RegisteredRep(Representative.RepCorba repObjRef, Vector clientList)
    {
        this.repObjRef = repObjRef;
        this.clientList = clientList;
    }
    protected void addNewClient(/*Client.ClientCorba clientObjRef*/RegisteredClient
regClient)
    {
        clientList.addElement(regClient);
    }
    protected void removeClient(RegisteredClient regClient)
    {
        clientList.removeElement(regClient);
    }
```

```
    protected void removeAllClients()
    {
        clientList.removeAllElements();
    }
}

class RegisteredClient
{
    public Client.ClientCorba clientObjRef;
    public IBClientInfo clientInfo;
    // This rep for which the registered client is waiting for(or currently serving)
    public RegisteredRep rep;

    public RegisteredClient()
    {
        clientObjRef = null;
        clientInfo = null;
        rep = null;
    }
    public RegisteredClient(Client.ClientCorba clientObjRef, RegisteredRep rep)
    {
        this.clientObjRef = clientObjRef;
        this.rep = rep;
    }

    public RegisteredClient(Client.ClientCorba clientObjRef, IBClientInfo clientInfo)
    {
        this.clientObjRef = clientObjRef;
        this.clientInfo = clientInfo;
        this.rep = null;
    }

    protected void addToRepWaitingList(RegisteredRep rep)
    {
        this.rep = rep;
        rep.addNewClient(this);
    }
}
```

```
/*************************************************************
 * File name:         IBDBManager.java
 * Author:            Haiyu Huang
 * Date:              Oct 29, 2000
 * Descriptions:      IIB Database manager. this class implements
 *                    all database manipulation functions. only server
 *                    side corba object Bank manager can call the
 *                    methods in this class
 *************************************************************/
import java.sql.*;
import java.util.*;

public class IBDBManager
  {
  static Connection con;      // A connection to the database
  static Statement stmt;      // All purpose statement

  public IBDBManager()
  {
  }

  public void loadInitBankDatabase() throws Exception {
     Vector tables = findBankTables();
     if (tables.isEmpty()) {
        connect();
        IBDBCreator dbCreator = new IBDBCreator();
        dbCreator.createDB();
        closeConnection();
     }
  }

  public void connect() throws Exception
  {
   String databaseName = "hhuang";
   String userName = "hhuang";
   String password = "Sql2000";
   try
   {
       Class.forName("org.gjt.mm.mysql.Driver").newInstance();
   } catch(Exception e) {
       System.err.println("Unable to load driver.");
       e.printStackTrace();
   }
   try
   {
       con = DriverManager.getConnection(
         "jdbc:mysql://arachne.cs.concordia.ca/" + databaseName + "?user="
         + userName + "=" + password,userName, password);
   }catch (SQLException e) {
       System.out.println("SQLException: " + e.getMessage());
       System.out.println("SQLState:     " + e.getSQLState());
       System.out.println("VendorError:  " + e.getErrorCode());
       System.err.println("System Exception in connect");
       System.err.println(e);
       throw e;
   }
   System.out.println("Connect to database");
  }

  public void closeConnection() throws Exception
  {
    try
    {
      System.out.println("Closing connection");
      con.close();
    } catch (Exception e)
    { System.err.println("System Exception in closeConnection");
      System.err.println(e);
      throw e;
    }
  }
```

```
   private Vector findBankTables( ) throws Exception
   {
      Vector tables = new Vector();
    try {

       Statement stmt = con.createStatement();
      String query = "SHOW TABLES";
      String table = "" ;
      // Get list of table name in database
      ResultSet rset = stmt.executeQuery(query);
      while (rset.next()) {
       table = rset.getString(1);
       tables.addElement(table);
      }
      rset.close();
      stmt.close();
    } catch(Exception e)
     {
      System.err.println("System Exception in finding tables");
       System.err.println(e);
       throw e;
     }
    return tables;
   }

   // See if given account exists
   public CAccount[] lookupAccount(String acctNum, String password) throws SQLException
   {
      CAccount[] cAccounts = null;
      Statement stmt;
      System.out.println("look up account " + acctNum + " password " + password);
      try {
      stmt = con.createStatement();
      String query;
      if(password != null) {
       query = "SELECT * FROM ACCOUNT " +
             "WHERE ACCT_NUM = " + acctNum +
             " AND PASSWORD = '" + password + "'";
      }else {
       query = "SELECT * FROM ACCOUNT " +
             "WHERE ACCT_NUM = " + acctNum;
      }
       System.out.println(query);
      ResultSet rset = stmt.executeQuery(query);
      boolean fAccountCreated = false;
      int numAcct = 0;
      while (rset.next()) {
         numAcct += 1;
         if(numAcct == 1) {
          cAccounts = new CAccount[5];
          System.out.println("account " + acctNum + " password " + password + " found");
         }
         CAccount cAccount= new CAccount();
         cAccount.type = (char)rset.getByte(3);
         cAccount.branchID = rset.getInt(4);
         cAccount.balance = rset.getFloat(5);
         if(numAcct > 5) {
          System.out.println("account " + acctNum + " password " + password + " has more
than 5 accts");
          break;
         }
         // Add to accout lists
         cAccounts[numAcct-1] = cAccount;
      }
       if(numAcct == 0) {
         System.out.println("account " + acctNum + " password " + password + " not
found");
      }
      rset.close();
      stmt.close();
```

```
    } catch (SQLException e) {
      System.out.println("SQLException: " + e.getMessage());
      System.out.println("SQLState:     " + e.getSQLState());
      System.err.println("System Exception in lookup account");
      System.err.println(e);
      throw e;
    }
     return cAccounts;
  }
  // look up customer information, the return value is filled inside clientInfo
  public boolean lookupCustomer(String acctNum, IBClientInfo clientInfo){
     boolean fSuccess = false;
     Statement stmt;
     System.out.println("look up customer " + acctNum);
     try {
     stmt = con.createStatement();
     String query = "SELECT * FROM CUSTOMER " +
                "WHERE ACCT_NUM = " + acctNum;

      System.out.println(query);
     ResultSet rset = stmt.executeQuery(query);
     if (rset.next()) {
        clientInfo.setName(rset.getString(2));
        clientInfo.setAddress(rset.getString(3));
        clientInfo.setTelephone(rset.getString(4));
        System.out.println("Customer " + acctNum  + " found");
        fSuccess = true;
     } else {
        fSuccess = false;
        System.out.println("Customer " + acctNum + " not found");
     }
     rset.close();
     stmt.close();
    } catch (SQLException e) {
      System.out.println("SQLException: " + e.getMessage());
      System.out.println("SQLState:     " + e.getSQLState());
      System.err.println("System Exception in lookup account");
      System.err.println(e);
    }
    return fSuccess;
  }
  // See if given representative exists
  public IBRepInfo lookupRepresentative(String repID, String password) throws
SQLException
  {
     IBRepInfo repInfo = null;
     Statement stmt;
     System.out.println("look up representative " + repID + " password " + password);
     try {
     stmt = con.createStatement();
     String query = "SELECT * FROM REPRESENTATIVE " +
                "WHERE REP_ID = " + repID +
                " AND PASSWORD = '" + password + "'";
      System.out.println(query);
     ResultSet rset = stmt.executeQuery(query);
     if (rset.next()) {
        repInfo = new IBRepInfo();
        repInfo.setID(rset.getInt(1));
        repInfo.setName(rset.getString(3));
        repInfo.setBranchID(rset.getInt(4));
        repInfo.setAddress(rset.getString(5));
        repInfo.setTelephone(rset.getString(6));
        repInfo.setPhotoFile(rset.getString(7));
        System.out.println("rep " + repID + " password " + password + " found");
     } else {
        repInfo = null;
        System.out.println("rep " + repID + " password " + password + " not found");
     }
     rset.close();
     stmt.close();
    } catch (SQLException e) {
```

```
      System.out.println("SQLException: " + e.getMessage());
      System.out.println("SQLState:     " + e.getSQLState());
      System.err.println("System Exception in lookup account");
      System.err.println(e);
      throw e;
    }

   return repInfo;
  }

  public float checkAccountBalance(String acctNum, String acctType)
     throws SQLException
  {
     float balance = 0;
     Statement stmt;
     System.out.println("check account " + acctNum + " type " + acctType + " balance");
     try {
     stmt = con.createStatement();
     String query = "SELECT BALANCE FROM ACCOUNT " +
                "WHERE ACCT_NUM = " + acctNum +
                " AND ACCT_TYPE = '" + acctType + "'";
     System.out.println(query);
     ResultSet rset = stmt.executeQuery(query);
     if (rset.next()) {
       balance = rset.getFloat(1);
        System.out.println("account " + acctNum + " type " + acctType + " balance = " +
balance);
     } else {
        System.out.println("account " + acctNum + " type " + acctType + " not found");
     }
     rset.close();
     stmt.close();
    } catch (SQLException e) {
     System.out.println("SQLException: " + e.getMessage());
     System.out.println("SQLState:     " + e.getSQLState());
     System.err.println("System Exception in lookup account");
     System.err.println(e);
     throw e;
    }

   return balance;
  }

  public float checkBillAccountDue(String acctNum)
     throws SQLException
  {
     float amountDue = 0;
     Statement stmt;
     System.out.println("check bill_account " + acctNum + " current amount due");
     try {
     stmt = con.createStatement();
     String query = "SELECT AMOUNT_DUE FROM BILL_ACCOUNT" +
                " WHERE ACCT_NUM = '" + acctNum + "'";
     System.out.println(query);
     ResultSet rset = stmt.executeQuery(query);
     if (rset.next()) {
        amountDue = rset.getFloat(1);
        System.out.println("account " + acctNum + " amount due = " + amountDue);
     } else {
        System.out.println("account " + acctNum + " amount due not found");
     }
     rset.close();
     stmt.close();
    } catch (SQLException e) {
     System.out.println("SQLException: " + e.getMessage());
     System.out.println("SQLState:     " + e.getSQLState());
     System.err.println("System Exception in lookup account");
     System.err.println(e);
     throw e;
    }
```

```
   return amountDue;
  }
  public String findBillAcctNum(String bankAcctNum, String billName)
  throws SQLException
  {
     String billAcctNum = null;
     Statement stmt;
     try {
     stmt = con.createStatement();
     String query = "SELECT ACCT_NUM FROM BILL_ACCOUNT" +
                " WHERE BANK_ACCT_NUM =" + bankAcctNum +
                " AND ACCT_NAME = '" + billName + "'";
     System.out.println(query);
     ResultSet rset = stmt.executeQuery(query);
     if (rset.next()) {
        billAcctNum = rset.getString(1);
        System.out.println("account " + bankAcctNum + billName + " bill account found" );
     } else {
        System.out.println("account " + bankAcctNum + billName + " bill Account not
found");
     }
     rset.close();
     stmt.close();
    } catch (SQLException e) {
      System.out.println("SQLException: " + e.getMessage());
      System.out.println("SQLState:     " + e.getSQLState());
      System.err.println("System Exception in lookup account");
      System.err.println(e);
      throw e;
    }

   return billAcctNum;
  }
  public boolean lookupBillAccount(String acctNum, CBillInfo cBillInfo)
     throws SQLException
  {
     boolean fSuccess = false;
     Statement stmt;
     System.out.println("check bill_account " + acctNum + " current amount due");
     try {
     stmt = con.createStatement();
     String query = "SELECT * FROM BILL_ACCOUNT" +
                " WHERE ACCT_NUM = '" + acctNum + "'";
     System.out.println(query);
     ResultSet rset = stmt.executeQuery(query);
     if (rset.next()) {
        cBillInfo.acctNum = rset.getString(1);
        cBillInfo.name =  rset.getString(2);
        cBillInfo.balanceForward =  rset.getFloat(4);
        cBillInfo.amountDue = rset.getFloat(5);
        System.out.println("account " + acctNum + cBillInfo.name + " bill acct found");
        fSuccess = true;
     } else {
        fSuccess = true;
        System.out.println("account " + acctNum + " bill account not found");
     }
     rset.close();
     stmt.close();
    } catch (SQLException e) {
      System.out.println("SQLException: " + e.getMessage());
      System.out.println("SQLState:     " + e.getSQLState());
      System.err.println("System Exception in lookup account");
      System.err.println(e);
      throw e;
    }
  return fSuccess;
  }

  public String viewAccountTranaction(String acctNum, String acctType,
        String startDate, String endDate) throws SQLException
  {
```

```
      String transactions = "";
      Statement stmt;
      System.out.println("check account " + acctNum + " type " + acctType + "
transactions");
      try {
      stmt = con.createStatement();
      String query;
      if(startDate != null && endDate != null) {
       query = "SELECT * FROM TRANSACTION " +
                   "WHERE ACCT_NUM = " + acctNum +
                   " AND ACCT_TYPE = '" + acctType + "'" +
                   " AND TRANS_DATE >= '" + startDate + "'" +
                   " AND TRANS_DATE <= '" + endDate + "'";
      }else {// get all transaction
         query = "SELECT * FROM TRANSACTION " +
                   "WHERE ACCT_NUM = " + acctNum +
                   " AND ACCT_TYPE = '" + acctType + "'";
      }
       System.out.println(query);
      ResultSet rset = stmt.executeQuery(query);
      while (rset.next()) {
        //Transactions = rset.getFloat(1);
        transactions = transactions + rset.getString(4) + "\t"
                              + rset.getString(5) + "\t"
                              + rset.getFloat(6) + "\t"
                              + rset.getDate(7) + "\n";
      }
      System.out.println("account " + acctNum + " type " + acctType + " transactions found
");
      rset.close();
      stmt.close();
     } catch (SQLException e) {
       System.out.println("SQLException: " + e.getMessage());
       System.out.println("SQLState:     " + e.getSQLState());
       System.err.println("System Exception in lookup account");
       System.err.println(e);
       throw e;
     }

    return transactions;
   }

   public boolean updateAccountBalance(String acctNum, String acctType, String amount)
      throws SQLException
   {
      boolean fSuccess = false;
      Statement stmt;
      System.out.println("update account " + acctNum + " type " + acctType + " balance");
    try {
       stmt = con.createStatement();
       String query = "UPDATE ACCOUNT SET BALANCE = BALANCE" + amount +
                 " WHERE ACCT_NUM = " + acctNum +
                 " AND ACCT_TYPE = '" + acctType + "'";
       System.out.println(query);
       stmt.executeUpdate(query);
       System.out.println("account " + acctNum + " type " + acctType + " balance is
updated");
       stmt.close();
       fSuccess = true;
     } catch (SQLException e) {
       System.out.println("SQLException: " + e.getMessage());
       System.out.println("SQLState:     " + e.getSQLState());
       System.err.println("System Exception in lookup account");
       System.err.println(e);
       throw e;
     }
    return fSuccess;
   }
   public void payBillAccount(String acctNum, String amount)
      throws SQLException
   {
```

```
    Statement stmt;
    System.out.println("pay bill_account " + acctNum);
    try {
     String query = "";
     stmt = con.createStatement();
     float amountDue = checkBillAccountDue(acctNum);
     float amountPaid = (Float.valueOf(amount)).floatValue();
     if(amountDue > 0.00 ) {
        if(amountPaid <= amountDue) {
           query = "UPDATE BILL_ACCOUNT SET AMOUNT_DUE = AMOUNT_DUE-" + amount +
                   " WHERE ACCT_NUM = '" + acctNum + "'";
        }else {
           query = "UPDATE BILL_ACCOUNT SET AMOUNT_DUE = 0.00" +
                   " WHERE ACCT_NUM = '" + acctNum + "'";

        }
        System.out.println(query);
        stmt.executeUpdate(query);
     }
     // check if there is more than current amount due , deduct the forword balance
     if (  amountPaid > amountDue ) {
        float surplus = amountPaid - amountDue;
        query = "UPDATE BILL_ACCOUNT SET BALANCE_FORWORD = BALANCE_FORWORD-" + surplus +
                   " WHERE ACCT_NUM = '" + acctNum + "'";
        System.out.println(query);
        stmt.executeUpdate(query);
     }
     System.out.println("bill account " + acctNum + " is updated");

     stmt.close();
   } catch (SQLException e) {
     System.out.println("SQLException: " + e.getMessage());
     System.out.println("SQLState:     " + e.getSQLState());
     System.err.println("System Exception in lookup account");
     System.err.println(e);
     throw e;
   }
}
public boolean createTermDeposit(String acctNum, float capital, float interestRate,
                        String startDate, String matureDate){
 boolean fSuccess = false;

 // Prepared insertion SQL statement
 try {
    PreparedStatement pstmt = con.prepareStatement(
    "INSERT INTO TERM_ACCOUNT (BANK_ACCT_NUM, CAPITAL, INTEREST_RATE, S_DATE, " +
    "E_DATE) VALUES (?, ?, ?, ?, ?)");

    pstmt.setInt(1,Integer.parseInt(acctNum));
    pstmt.setFloat(2, capital);
    pstmt.setFloat(3, interestRate);
    StringTokenizer dateToks = new StringTokenizer(startDate, "/");
    int year = Integer.parseInt(dateToks.nextToken()) - 1900;//year -1900
    int month = Integer.parseInt(dateToks.nextToken());//month 0-11
    int day = Integer.parseInt(dateToks.nextToken());//day
    pstmt.setDate(4, new java.sql.Date(year, month, day));
    dateToks = new StringTokenizer(matureDate, "/");
    year = Integer.parseInt(dateToks.nextToken())-1900;//year -1900
    month = Integer.parseInt(dateToks.nextToken());//month 0-11
    day = Integer.parseInt(dateToks.nextToken());//day
    pstmt.setDate(5, new java.sql.Date(year, month, day));
    pstmt.executeUpdate();

    con.commit();
    pstmt.close();
    fSuccess = true;
    } catch(Exception e){
       System.err.println("System Exception in insertint account Data");
       System.err.println(e);
    }
```

```
      return fSuccess;
   }
   public boolean createTransaction(String acctNum, String acctType, String desc,
                              String debitCredit, String amount)
      throws SQLException
   {
      boolean fSuccess = false;
      Statement stmt;
      System.out.println("create transaction upon account " + acctNum + " type " +
acctType);
     try {
        stmt = con.createStatement();
        // Get current date
        String curDate;
        ResultSet rset = stmt.executeQuery("SELECT CURDATE()");
        if(rset.next()) {
           curDate = rset.getString(1);
        } else {
           curDate = "2000-01-01";
        }
        String query = "INSERT INTO TRANSACTION (TRANS_ID, ACCT_NUM, ACCT_TYPE,
DESCRIPTION, " +
              "DEBIT_CREDIT, AMOUNT, TRANS_DATE)" +
              " VALUES (NULL," + acctNum + ", '" + acctType + "', " + "'" + desc + "', " +
                    "'" + debitCredit + "', " + amount + ", '" + curDate + "')";
        System.out.println(query);
        stmt.executeUpdate(query);
        System.out.println("transaction is generated upon account " + acctNum);
        stmt.close();
        fSuccess = true;
     } catch (SQLException e) {
        System.out.println("SQLException: " + e.getMessage());
        System.out.println("SQLState:     " + e.getSQLState());
        System.err.println("System Exception in lookup account");
        System.err.println(e);
        throw e;
     }
    return fSuccess;
   }

    public String getDBTable(String tableName) {

       String query = "SELECT * FROM "+ tableName;
       String table = "";
       Statement stmt;
       if(tableName.equals("REPRESENTATIVE")) {
          table =  "ID" + "\t" +
                "PASSWORD" + "\t" +
                "NAME" + "\t" +
                "BRANCH_ID" + "\t" +
                //"ADDRESS" + "                                                \t" +
                "ADDRESS" + "\t" + "\t" + "\t" +
                "TELEPHONE" + "\t" +
                "PHOTO" + "\n" + "\n";

          try {
             stmt = con.createStatement();
             ResultSet rset = stmt.executeQuery(query);
             while (rset.next()) {
                table +=  rset.getInt(1) + "\t" +
                       rset.getString(2)+ "\t" + "\t" +
                      rset.getString(3) + "\t" +
                      rset.getInt(4) + "\t" + "\t" +
                      rset.getString(5) + "\t" +
                      rset.getString(6) + "\t" +
                      rset.getString(7) + "\n";
             }
             rset.close();
             stmt.close();
          } catch (SQLException e) {
             System.err.println(e);
```

```
    }
} else if(tableName.equals("ACCOUNT")) {
    table =  "ACCT_NUM" + "\t" +
          "PASSWORD" + "\t" +
          "ACCT_TYPE" + "\t" +
          "BRANCH_ID" + "\t" +
          "BALANCE" + "\t" + "\t" +
          "DATE" + "\n" +"\n";

    try {
       stmt = con.createStatement();
       ResultSet rset = stmt.executeQuery(query);
       while (rset.next()) {
          table +=  rset.getInt(1) + "\t" + "\t" +
                 rset.getString(2)+ "\t" + "\t" +
                rset.getString(3) + "\t" + "\t" +
                rset.getInt(4) + "\t" + "\t" +
                rset.getFloat(5) + "\t" + "\t" +
                rset.getDate(6) + "\n";
       }
       rset.close();
       stmt.close();
    } catch (SQLException e) {
       System.err.println(e);
    }
}  else if(tableName.equals("BILL_ACCOUNT")) {
    table =  "ACCT_NUM" + "\t" + "\t" +
          "ACCT_NAME" + "\t" +
          "BANK_ACCT_NUM" + "\t" + "\t" +
          "BALANCE_FORWORD" + "\t" +
          "AMOUNT_DUE" + "\n" +"\n";

    try {
       stmt = con.createStatement();
       ResultSet rset = stmt.executeQuery(query);
       while (rset.next()) {
          if(rset.getInt(3) == 0) {
             table +=  rset.getString(1) + "\t" + "\t" +
                   rset.getString(2)+ "\t" + "\t" +
                   "NULL" + "\t" + "\t" + "\t" +
                   rset.getFloat(4) + "\t" + "\t" +"\t" +
                   rset.getFloat(5) + "\n";
          } else {
             table +=  rset.getString(1) + "\t" + "\t" +
                   rset.getString(2)+ "\t" + "\t" +
                   rset.getInt(3) + "\t" + "\t" + "\t" +
                   rset.getFloat(4) + "\t" + "\t" + "\t" +
                   rset.getFloat(5) + "\n";
          }
       }
       rset.close();
       stmt.close();
    } catch (SQLException e) {
       System.err.println(e);
    }
} else if(tableName.equals("TERM_ACCOUNT")) {
    table =  "TERM_ID" +  "\t" +
          "BANK_ACCT_NUM" + "\t" +
          "CAPITAL" + "\t" +
          "INTEREST_RATE" + "\t" +
          "S_DATE" + "\t" +
          "E_DATE" + "\n";

    try {
       stmt = con.createStatement();
       ResultSet rset = stmt.executeQuery(query);
       while (rset.next()) {
          table +=  rset.getInt(1) + "\t" +
                 rset.getInt(2)+ "\t" + "\t" +
                rset.getFloat(3) + "\t" +
                rset.getFloat(4) + "\t" +
```

```
                rset.getDate(5) + "\t" +
                rset.getDate(6) + "\n";
        }
        rset.close();
        stmt.close();
    } catch (SQLException e) {
        System.err.println(e);
    }
} else if(tableName.equals("CUSTOMER")) {
    table =  "ACCT_NUM" + "\t" +
          "NAME" + "\t" +  "\t" +
          "ADDRESS" + "\t" + "\t" + "\t" +
          "TELEPHONE" + "\n" + "\n";

    try {
        stmt = con.createStatement();
        ResultSet rset = stmt.executeQuery(query);
        while (rset.next()) {
            table +=  rset.getInt(1) + "\t" + "\t" +
                   rset.getString(2)+ "\t" + "\t" +
                  rset.getString(3) + "\t" +
                  rset.getString(4) + "\n";

        }
        rset.close();
        stmt.close();
    } catch (SQLException e) {
        System.err.println(e);
    }
}  else if(tableName.equals("TRANSACTION")) {

    table =  "TRANS_ID" + "\t" +
          "ACCT_NUM" + "\t" +
          "ACCT_TYPE" + "\t" +
          "DESCRIPTION" + "\t" +
          "DEBIT_CREDIT" + "\t" +
          "AMOUNT" + "\t" +
          "TRANS_DATE" + "\n" +"\n";

    try {
        stmt = con.createStatement();
        ResultSet rset = stmt.executeQuery(query);
        while (rset.next()) {
            table +=  rset.getInt(1) + "\t" + "\t" +
                   rset.getInt(2)+ "\t" + "\t" +
                  rset.getString(3) + "\t" + "\t" +
                  rset.getString(4) + "\t" +
                  rset.getString(5) + "\t" + "\t" +
                  rset.getFloat(6) + "\t" +
                  rset.getDate(7) + "\n";
        }
        rset.close();
        stmt.close();
    } catch (SQLException e) {
        System.err.println(e);
    }
} else if(tableName.equals("BRANCH")) {
    table =  "BRANCH_ID" + "\t" +
          "ADDRESS" + "\n";

    try {
        stmt = con.createStatement();
        ResultSet rset = stmt.executeQuery(query);
        while (rset.next()) {
            table +=  rset.getInt(1) + "\t" + "\t" +
                   rset.getString(2)+ "\n";
        }
        rset.close();
        stmt.close();
    } catch (SQLException e) {
        System.err.println(e);
```

```
            }
        }

        return table;
    }
}
```