Employee Self Service

Man He

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

January 2003

0-612-77988-2

Abstract

Employee Self Service

Man He

An Employee Self Service System (ESSS) has been designed and implemented using Microsoft Visual C++ and Microsoft Distributed Component Object Model (DCOM). The ESSS is a real time, three-tier (presentation, application, and data), distributed application capable of running on a configurable number of personal computers. It allows users to view, change, and update their employment-related information, such as personal information, benefit selections, and retirement contributions. To support data efficiency, the database is fully controlled by the data-tier. To support network efficiency, data storage is provided by the application-tier for the frequently requested data. To support user efficiency, the users are provided with a single-window user interface that covers all the functions needed to perform a user task. An in-depth description of DCOM technology is also provided with this report.

Acknowledgements

I would like to take this opportunity to express my sincere thanks to my supervisor, Professor P. Grogono, for his kind support throughout my studies at Concordia University. Without his extraordinary guidance, this work would not have been completed.

My thanks also go to all the professors and staff in the department of computer science, especially Halina Monkiewicz, for their excellent supports during the entire studies of my Master's degree. I would like to extend special thanks to my friend, Mrs. Lan Jin, for her help in getting the necessary information for my project.

I am deeply indebted to my beloved husband for his wonderful support and understanding while I was doing my project and writing my major report at home. My deep love goes to my husband and my daughter. It is to them that I dedicate this report.

Finally, to my dearest parents, I express my cordial thanks for their continuous encouragement while I was pursuing my studies.

Table of Contents

Table of Figures

1. Introduction

This report presents the design and implementation of Employee Self Service System (ESSS). The ESSS is designed to facilitate employment-related self-service process. It provides a convenient way for employees to view and update their personal information, their benefit information, and their retirement account information. It also provides a feasible way for implementing a reliable, distributed application in Windows environment.

1.1. The Background for the ESSS

In many companies, the processes such as updating personal information, changing benefit selection, and changing retirement contribution are very time consuming and inconvenient. Employees need to fill out forms and to submit the forms to Human Resources. The Human Resources then processes the requests and enters the data into the database. This mechanism often causes problems to both employee and human resource individuals. One of the problems would be the time delay between the user request and the data entry. For certain time-critical user requests, such as retirement account fund reallocation, the delay could bring negative impact on employees. To increase the efficiency of data processing, a system called Employee Self Service was designed and implemented.

The ESSS is designed to help employees view, change, and update their personal information, their benefit selections, and their retirement contributions without going through the human resource individuals. The windows-explorer-based user interface makes the ESSS easy to learn and easy to use. Its self-service features greatly reduce the workload of human resource individuals. At the same time it offers great convenience for employees to access their employment-related information. With a simple click on one of the icons located on the left side window, the corresponding information will be quickly displayed on the right side window. In summary, the ESSS would be an excellent tool for employees in managing their employment-related information.

1.2. The needs for the component software

The ESSS is designed to run on a personal computer (PC) in Windows environment. The primary reason for using a PC and Windows operating system is their popularity and affordability for most companies. For large companies with up to several hundred thousand employees, a powerful PC might be needed to install server components. A typical powerful PC is often equipped with dual CPUs and 2 GB physical memory. Its dual CPUs are expected to offer quick response to meet a number of user requests; while its high capacity of memory is able to hold information about half a million employees. Therefore, a personal computer would be an ideal choice for hosting the Employee Self Service System.

To show the importance of component software in the ESSS, we examine two different systems. In system 1, we implement the ESSS to run on a single computer. The data to be processed can be retrieved from a remote database via a network connection. Each employee is able to access and retrieve the data from his desktop computer at his own office. With this system architecture, the number of database accesses could be very high. As a result, the database server could be overloaded, causing significant delay in system response. In the worst case, the database server could crash.

In system 2, we intend to remove the drawbacks caused by frequent database accesses. To reduce the number of accesses, we need to establish the so-called middle layers to store frequently requested data. This can be accomplished by splitting the ESSS into several components with different components to run on different computers. In such a distributed environment, the data will be loaded and stored into its dedicated middle layer after the data is first retrieved from the database. All the subsequent requests for the same data will be provided by the middle layer component without the need to access the database again. With this mechanism in place, the number of database accesses will be reduced significantly.

This distributed system architecture allows the fine-tuning of the system performance. When the number of the users increases, more components can be added into the system to maintain high performance. Based on the most recent software development, these software components could be implemented by using the so-called component software technology.

So far, two parallel technologies are available for the component software. One of the technologies — the Distributed Component Object Model (DCOM) [1][2][3] is developed by the Microsoft Cooperation to support communication among objects on different computers. The other one — the Common Object Request Broker Architecture (CORBA) [4][5] is developed by the Object Management Group (OMG) to support the invocations of operations on objects located anywhere on a network.

Both DCOM and CORBA support the following common features in the development of server components:

- Multiple programming languages. Most mainstream programming languages can be used for component implementation.
- Location transparency. Client can invoke server object without the details of server location.
- Strong security. Different levels of security checks are performed for server object invocation and access.
- Multithreaded server. Concurrent multiple-client accesses to the server object are supported.

In addition to the above common features, several key differences between DCOM and CORBA can be summarized as below:

- High-quality development tools are available for building DCOM components. On the contrary, no development tools are provided for building CORBA components.
- DCOM has large selection of commercially available ActiveX components for use. CORBA does not have such support, leading to longer development cycles on average.
- CORBA supports a wide variety of operating systems. DCOM only support Windows NT 4.0, Windows 95/98, Windows XP, Windows 2000, Sun Solaris 2.5, and Digital UNIX.
- CORBA supports implementation inheritance, one interface can inherit another's components. DCOM only supports interface inheritance, the derived interface does not inherit the components available under the original implementation.

For the ESSS where PC and Windows operating system are used, it is more natural to use DCOM rather than CORBA to implement the system since more development support is available for DCOM in Windows environment.

1.3. The Distributed Application for the ESSS

A typical application that interacts with a user, such as the ESSS, usually consists of three elements: presentation, application logic, and data. Presentation focuses on interacting with the user. Application logic performs calculations and determines the flow of execution of the application. Data elements manage information that must persist across sessions or be shared between users.

Two-tier, or standard client/server applications, as described in Section 1.2, group presentation and application logic components on a single client computer and access a shared data source using a network connection. The advantage of such a configuration is that the data is centralized. This centralization benefits an organization by sharing data, providing consistency in accessed data, and reducing duplication and maintenance. But there are also a number of limitations for two-tier applications, such as poor scalability, poor maintainability, poor reusability, and poor network performance.

In three-tier architectures, presentation, application logic, and data elements are conceptually separated. Presentation components manage user interaction and request application services by calling middle-tier (application-tier) components. Application components perform business logic and make requests to access databases in the data-tier. Application design becomes more flexible because clients can call server-based components to complete a request, and components can call other components to improve code reuse. Three-tier applications that are implemented by using multiple servers across a network are referred to as *distributed applications*. Three-tier architectures are often called *server centric*, because they uniquely enable application components to run on middle-tier servers, independent of both the presentation interface and database implementation. The separation of application logic from presentation tier and data element offers many benefits:

- Multi-language support. Application components can be developed using general programming languages.
- Centralized components. Components can be centralized for easy development, maintenance, and deployment.
- Load balancing. Application components can be spread across multiple servers, allowing for better scalability.
- Efficient data access. The problems for database connection are minimized since the database now sees only the application component, and not all of its clients. Also database connections and drivers are not required on the client.
- Improved security. Middle-tier application components can be secured centrally using a common infrastructure. Access can be granted or denied on a component-by-component basis, simplifying administration.
- Simplified access to external resources. Access to external resources, such as mainframe applications and other databases, is simplified; a gateway server becomes another component that is used by the application.

The ESSS fits well into three-tier distributed application. The employees interact with the system through the graphic user interface provided by the presentation tier. The application services are provided to the presentation-tier component by the middle-tier components. In the ESSS, it is the middle tiers that provide the essential data storage to reduce the network traffic and database accesses. The data-tier component manages the accesses to the database, as for example changing employee personal information. All these software components should be configured and distributed into several different PCs to maintain the maximum performance of the system. The structure of three-tier application is shown in Figure 1. The adjacent tiers are connected through the network.

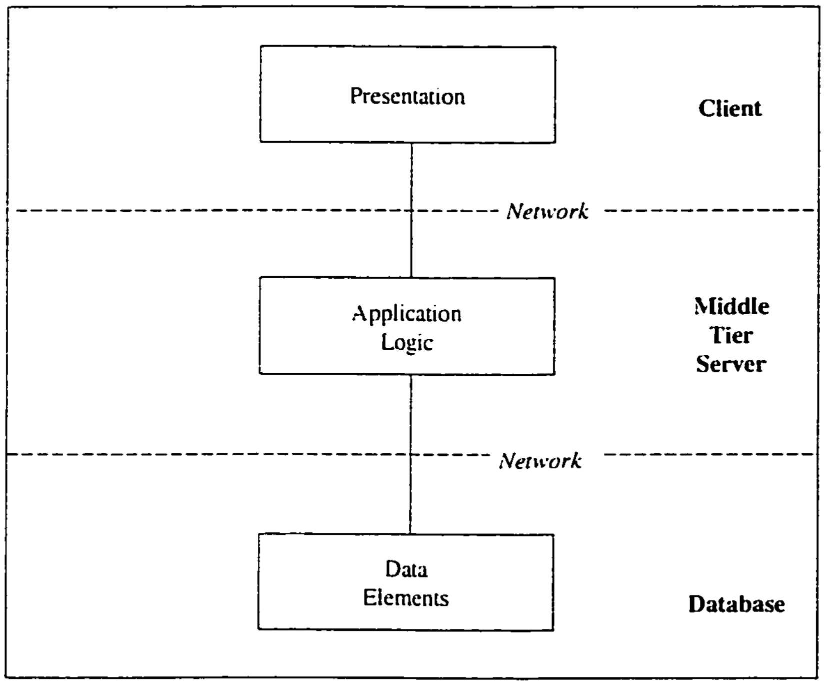


Figure 1. Structure for the Three-tier Application

The reason for the ESSS to use distributed three-tier application is not only the resource sharing, but also the benefits gained from distributing the components into several different computers. By partitioning and distributing a complex application into presentation, application logic, and data sections, we can achieve the following benefits:

- Scalability: As the number of users or workload increases, more server components can be added. Thus the application does not downgrade.
- Reliability: When a hardware or software failure should occur in some components, the system could continue functioning through the other components.
- Efficiency: Problems in certain components could be located and resolved quickly without the need to shut down the whole system.

2. Inside DCOM

In this chapter, we present an in-depth description of the Microsoft Distributed Component Object Model (DCOM). This component technology will be used in the implementation of application-tier and data-tier of the ESSS.

DCOM is a specification and set of services that allow software developer to create modular, object-oriented, customizable, and distributed applications using a number of languages [6]. It supports communication among objects on different computers, whether on a local area network (LAN), a wide area network (WAN), or even the Internet. In DCOM architecture, applications are built from packaged binary components with well-defined interface [1][2][3][6]. DCOM allows flexible update of existing applications, provides a higher-degree of application customization, encourages large-scale software reuse, and provides a natural migration path to distributed applications. The Component Object Model (COM) [9] is an approach to achieving component software architecture. COM specifies a way for creating components and for building applications from components. Specifically, it provides a binary standard that components and their clients must follow to ensure dynamic inter-operability. DCOM is the distributed extension of COM. It is an application-level protocol for object-oriented remote procedure call. With DCOM technology, an application can be configured and distributed at any locations. Because DCOM is an extension of COM, one can take advantage of the existing COM-based components and knowledge to quickly build a new distributed application.

2.1. Object Activation

One of the important features of DCOM is that it has a mechanism for establishing connections to components and creating new instances of components either locally or remotely. In the COM/DCOM world, object classes are named with globally unique identifiers (GUIDs), which are called Class IDs. These Class IDs are nothing more than fairly large integers (128 bits) that provide a collision free, decentralized namespace for object classes. The COM libraries provide *CoCreateInstanceEx*, *CoGetInstanceFromFile* for remote object creation, which in turn calls *QueryInterface* implemented in the server component.

In order to be able to create a remote object, the COM libraries need to know the network name of the server. Once the server name and the Class Identifier (CLSID) are known, the service control manager (SCM) on the client machine connects to the SCM on the server machine and requests creation of this object. As shown in Figure 2, the DCOM protocol is layered on top of the OSF DCE RPC specification [10]. Next to the DCE RPC is the *Security Provider* offering security checks for the method calls between client and object. Proxy and stub are responsible for marshalling and unmarshalling (see next section) for any method call. Their details will be given in the following section.

Two fundamental mechanisms allow clients to specify the remote server name when an object is created:

- As a fixed configuration in the system registry.
- As an explicit parameter to *CoCreateInstanceEx*, *CoGetInstanceFromFile*.

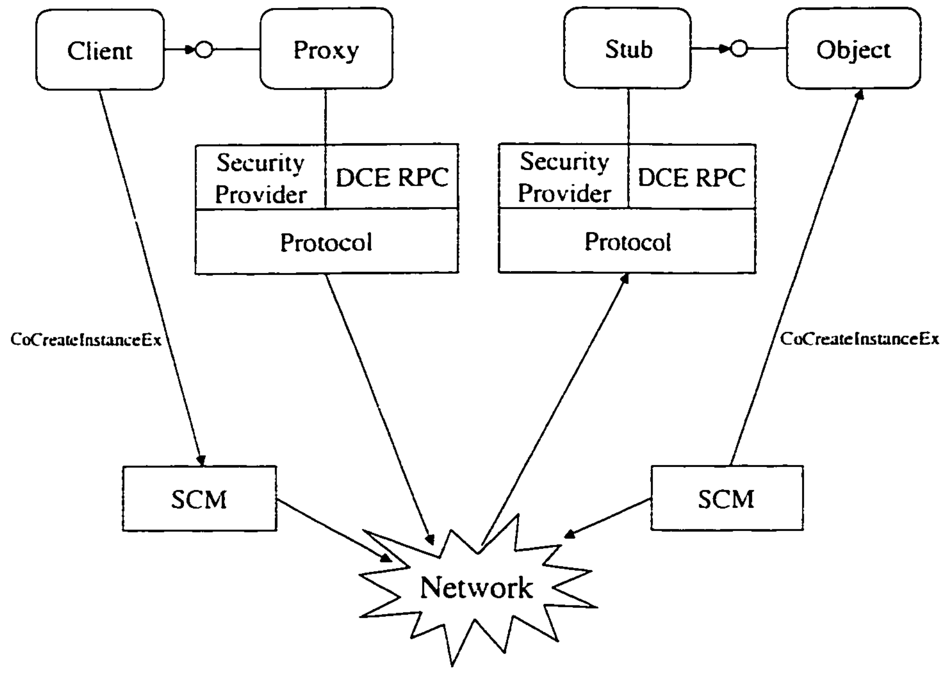


Figure 2. DCOM Architecture

The first mechanism is extremely useful for maintaining location transparency. By making the remote server name part of the server configuration information registered on the client machine, clients do not have to worry about maintaining or obtaining the server location. If the server name changes, the registry is changed and the application continues to work without further action.

Some applications require explicit run-time control over the server to be connected. For this kind of application, COM allows the remote server name to be explicitly specified as a parameter to *CoCreateInstanceEx*, *CoGetInstanceFromFile*. The developer of the client code is in complete control of the server name being used by COM for remote activation.

2.2. Marshaling and Unmarshaling

When a client wants to call an object in another address space, the parameters to the method call must be passed from the client's process to the object's process. The client places the parameters on the stack. For remote invocations, the caller and the object don't share the same stack. Some COM libraries need to read all parameters from the stack and write them to a memory buffer so they can be transmitted over a network. The process of reading parameters from the stack into a memory buffer is called "marshaling". The counterpart to marshaling is the process of reading the flattened parameter data and recreating a stack that looks exactly like the original stack set up by the caller. This process is called "unmarshaling". Once the stack is recreated, the object can be called. As the call returns, any return values and output parameters need to be marshaled from the object's stack, sent back to the client, and unmarshaled into the client's stack.

COM provides sophisticated mechanisms for marshaling and unmarshaling method parameters that build on the remote procedure call (RPC) infrastructure defined as part of the distributed computing environment (DCE) standard. In order for COM to be able to marshal and unmarshal parameters correctly, it needs to know the exact method signature, including all data types. This information is provided using an interface definition language (IDL), which is also built on top of the DCE RPC [10] standard. IDL files are compiled using a special IDL compiler — MIDL compiler, which is part of the Win32 SDK. The IDL compiler generates C source files that contain the code for performing the marshaling and unmarshaling for the interface described in the IDL file. The client-side code is called the "proxy," while the code running on the object side is called the "stub." The proxies and stubs generated by MIDL are COM objects that are loaded by the COM libraries when needed. By looking up the Interface Id (IID) from the system registry, COM can find the proxy/stub combination for a particular interface.

2.3. Object Connection Control

An object's lifetime is controlled by a mechanism called reference counting, which uses the *AddRef* and *Release* methods of interface *IUnknown*. Every COM/DCOM interface must inherit from interface *IUnknown*; every COM/DCOM component must implement *AddRef* and *Release*. *AddRef* and *Release* are called quite often, and sending every call to a remote object would introduce a serious network performance penalty. Hence, DCOM optimizes *AddRef* and *Release* calls for remote objects. To do so, remote reference counting is conducted per interface rather than per connection, allowing for greater network efficiency.

Remote reference counting would be entirely adequate if clients never terminated abnormally, but in fact they do. To make system robust in the face of clients terminating abnormally when they hold remote references, a *Pinging* mechanism is used for detecting any client abnormal termination. At every elapse of predefined ping period time, the server object sends a ping signal to the connected client. If the ping period elapses without receiving a ping on that server object, all the remote references to interfaces associated with that server object are considered "expired" and can be garbage collected.

2.4. DCOM Threading Models

To support multi-clients accessing the server objects concurrently, multi-threading models are required for the DCOM architecture.

2.4.1. Single-Threaded Apartments (STA)

In a single-threaded apartment, each object lives in a thread. Each thread must initialize COM using either **CoInitialize** or **CoInitializeEx.**

The basic concurrency unit in an STA is the individual thread that initializes COM. If two objects, for example A and B, live in the same apartment, and A is processing a call, no other client can make a call, to either A or B, until A completes its service. As a result, instance data that is exclusive to an object need not be protected, because only one thread can ever enter this instance of the object. But this may delay the server response and thus downgrade the system performance.

2.4.2. Multithreaded Apartment (MTA)

A multithreaded apartment is an easier model compared to STA. Incoming RPC calls directly use the thread assigned by the RPC run time. The object does not live in any specific thread. Clients from any thread can directly call any object inside the MTA. However, MTA requires extreme caution on the shared data. Multiple threads can call an object method at the same time. Therefore the object must provide synchronized access to any instance using synchronization primitives such as critical section and semaphores.

Although complicated to implement, MTA objects offer the possibility of higher performance and better scalability than STA objects since the generic synchronization that COM performs on STA is relatively expensive [1][2][3]. A good design technique is to isolate the critical areas of an application into separate objects and move the critical objects into MTAs to achieve high overall performance and scalability. Before using MTA, a thread must initialize COM by calling **CoInitializeEx**.

2.5. Security Issues

One of the most difficult issues in the design of distributed application is that of security. DCOM provides an extensible and customizable security framework for the developers.

2.5.1. Access Security

The most obvious security requirement on distributed applications is the need to protect objects against unauthorized access. Only authorized users are supposed to be able to connect to an object. Current implementations of DCOM provide declarative access control on a per-process level. Existing components can be securely integrated into a distributed application by simply configuring their security policy appropriately.

2.5.2. Launch Security

Another related requirement on a distributed infrastructure is to maintain control of object creation. Since all COM objects on a machine are potentially accessible via DCOM, it is critical to prevent unauthorized users from creating instances of these objects. For this purpose, the COM libraries perform special security validations on object activation. If a new instance of an object is to be created, COM validates that the caller has sufficient privileges to perform this operation. The privilege information is configured in the registry, external to the object.

2.5.3. Authentication

The above mechanisms for access and launch permission checks require some mechanism for determining the security identity of the client. This client authentication is performed by one of the security providers, which returns unique session tokens that are used for ongoing authentication once the initial connection has been established. The initial authentication often requires multiple round trips between caller and object.

DCOM uses access tokens to speed up security checks on calls. To avoid the additional overhead of passing the access token on each and every call, DCOM by default only requires authentication when the initial connection between two machines is established. It then caches the access token on the server side and uses it automatically whenever it detects a call from the same client. For many applications this level of authentication is a good compromise between performance and security. However, some applications may require additional authentication on every call, as for example passing in credit card information or other sensitive information; in these cases, the object might require calls to be individually authenticated.

2.5.4. Impersonation levels

A more subtle implication of security in distributed applications is the issue of protecting callers from malicious objects. Since DCOM allows objects to impersonate callers, objects can actually perform operations that they do not have sufficient privileges to perform alone. To prevent malicious objects from using the caller's credentials, the caller can indicate what it wants to allow objects to do with the security token it obtains. The following options are currently defined:

- Anonymous: The object is not allowed to obtain the identity of the caller. This is the safest setting for the client but the least powerful for the object.
- Identify: The object can detect the security identity of the caller, but can not impersonate the caller. This call is still safe for the client since the object will not be able to perform operations using the security credentials of the caller.
- Impersonate: The object can impersonate and perform local operations, but it can not call other objects on behalf of the caller. This mode is potentially insecure for the caller, since it allows the object to use the client's security credential to perform arbitrary operations.

These options are defined as part of the Windows NT security infrastructure. Again, DCOM allows these settings to be both programmatically controlled and externally configured.

3. System Requirements

In the area of software engineering, the development of software is based on software requirements. These requirements must be fully satisfied by the system implementation. After the implementation is completed, the system must be validated against the requirements. An invalid implementation must be corrected and validated again. In this chapter, the requirements for the ESSS are presented.

3.1. Hardware requirement

A number of personal computers are needed for installing the distributed ESSS. Internet connection facility is required for transferring the data between the different software components of the ESSS. At least three powerful PCs are needed for installing three server components. To achieve top system performance, a larger number of powerful PCs might be needed for server components. The number of PCs needed for installing client side application depends on the number of users. These PCs should be equipped with color monitors for displaying the graphic user interface. Mouse and keyboard are also required for entering user input. For testing purpose, at least two PCs are needed to demonstrate the distributed features of the ESSS.

3.2. Software requirement

The ESSS is a distributed, real-time application designed to be used in Windows environment. Since Windows 95/98 has weak security features, Windows NT 4.0 or above, Windows *XP*, or Windows 2000 are recommended. To guarantee the real-time and distributed feature of the ESSS, the implementation is performed using Microsoft Visual C++, MFC, and distributed component object models (DCOM). Before using the ESSS, MFC library and DCOM should be properly installed.

3.3. Requirements for Graphic User Interface

The ESSS consists of two main windows as its user interface: the login window and the data explorer window. The specifications for these two windows are not intended to cover all the possible user requirements.

3.3.1. Login Window

Upon the execution of the ESSS program, a login window is displayed to the user.

- User is prompted to enter user ID and password.
- The system starts processing user login information after user clicks on "Login" button with a mouse.
- Invalid login information pops up a message box informing the user that the system failed to LOG him onto the system. Upon the acknowledgement of the system message, the system clears the password fields and the user can try again.
- A successful login brings up the data explorer window.
- Clicking on "Cancel" button will terminate the ESSS application.

3.3.2. Data Explorer Window

Upon a successful login, a data explorer window appears. The data explorer window is a two-way split window that contains left side window and right side window. On the left side window displays a tree-structured function menu. Nothing is displayed on the right side window before any user request is made. To exit the data explorer window, click on the "X" button on the upper right side corner.

The function menu contains four basic categories — Personal Information, Paycheck Information, Benefit Elections, and Retirement Savings Plan, which are represented by a document folder on the left side window. Clicking on one of the folders once will open the folder and display the functions available for the particular category. Clicking on the folder once again will close the folder and hide the functions for this particular category.

3.3.2.1. Personal Information Category

- User can view his personal information by clicking on "Display" icon. The personal information will be displayed on the right side window.
- User can update his personal information by clicking on "Change" icon. An information update page will appear on the right side window. By entering the new personal information into the update page and clicking on "Submit" button, the user's new personal information will be updated in the database and the related middle tier servers. Upon the completion of the update, a message box will pop up to inform the user whether the update is successful.

3.3.2.2. Paycheck Information Category

- User can view his most recent pay information by clicking on "Most Recent Pay Information" icon. The pay information will be displayed on the right side window.

3.3.2.3. Benefit Elections

- User can view his dental insurance coverage by clicking on "Dental Plan" icon. The dental coverage information will be displayed on the right side window.

- User can view his medical insurance coverage by clicking on "Medical Plan" icon. The medical coverage information will be displayed on the right side window.
- User can view his vision care coverage by clicking on "Vision Plan" icon. The vision coverage information will be displayed on the right side window.
- User can view his basic life insurance coverage by clicking on "Basic Life" icon. The life insurance information will be displayed on the right side window.

3.3.2.4. Retirement Savings Plan

- User can view his retirement savings plan account balance by clicking on "Account Balance" icon. The account balance for different funds will be displayed on the right side window.
- User can reallocate his total account balance by clicking on "Fund Reallocation". A fund reallocation page will appear on the right side window. The user should enter the new percentage of the total balance for each available fund and then click on "Submit" button, the account balance for each fund will be reallocated and the information will be updated in the database as well as in the related middle tier servers. Upon the completion of transaction, a message box will pop up to inform the user whether the transaction is successful.
- User can change his total fund contribution direction by clicking on "Investment Direction". An investment direction page will appear on the right side window. The user should enter the new percentage of monthly retirement contribution for each available fund and then click on "Submit" button, the new investment direction for each fund will be saved in the database. Upon the completion of transaction, a message box will pop up to inform the user whether the transaction is successful.

- User can change the contribution rate to his retirement savings plan by clicking on "Contribution Rate" icon. A contribution rate page will appear on the right side window. The user should enter the new percentage of the monthly income as the future contribution rate to the retirement savings plan and then click on "Submit" button, whereupon the new contribution rate for the future investment will be saved in the database. Upon the completion of transaction, a message box will pop up to inform the user whether the transaction is successful.

3.4. Requirements for Middle-tier Servers

To support the three-tier distributed application, the ESSS is designed to have server components run on different PCs. These components form the application-tier of the ESSS. They should support the following features to meet the performance requirements.

- The data services should be obtained by calling data-tier component only.
- The data should be loaded and stored into middle tier servers after being retrieved from the database for the first time. Frequently requested data should be distributed into different server computers to avoid overloading of database server.
- The number of middle-tier servers should be determined from the capacity of the server computers and the maximum amount of data to be managed.

3.5. Requirements for Data-tier Server

In some companies, the databases may be built in the UNIX environment, and in others they may be built in Windows environment. To support the database location transparency, a data access server is needed for the data-tier. Only the data access server can directly access either local database or remote database whether it is on UNIX or on Windows. Such a configuration allows the middle-tier servers completely independent of database operations.

4. System Design

System design is an essential step in developing a reliable, secure, and user-friendly application. A complete analysis of users and the system environment is critical for a successful system design.

4.1. User Description

- The ESSS is to be used by the employees to perform employment-related self-services.
- The ESSS is designed to be easy to use. Low computer literacy is required for employees to use the system.
- Heavy usage is expected for employees to view paycheck information around each payday.

4.2. Design Consideration

The ESSS intends to be designed as a three-tier distributed application. The entire system design includes the design of presentation-tier — the Graphic User Interface, the design of application-tier — the application services, and the design of data-tier — the database accesses.

4.2.1. Presentation-tier — Graphic User Interface

Based on the user description, following points are considered in the design of graphic user interface:

- The ESSS is a distributed multi-user application; data are transferred between the server site and client (user) site through the Network. Database access is expected to be very heavy around the payday. In order to improve the performance, the data is loaded and stored into a number of data control servers after being retrieved from the database for the first time. The data will be returned from the dedicated data control server for any subsequent data retrieval without accessing the database again.
- Different kind of users may have different levels of knowledge in computer usage. Thus the ESSS is designed to be error-protected. User input should not cause system error or incorrect database updates. To ensure this, we need to reduce the keyboard input as far as possible since keyboard input is a potential source of error. In the ESSS, all the keyboard inputs are protected with data format checking. All the other interactions between users and the ESSS are mouse-oriented, which effectively protects the system data from any invalid input.
- The ESSS is designed to teach the user about the system incrementally. An informative message box is popped up for each invalid or incorrect user action, which allows the user to learn what is the cause and how to correct it.
- The ESSS is designed to be consistent. Once the database is updated, the same information will be updated across the middle tier servers and the user interface. This ensures the consistency between different end-users.

4.2.2. Middle-tier — Application Services

Based on the requirements for the middle-tier servers, the following points are considered as the guideline for designing the middle-tier servers. As mentioned in the introduction of this report, the middle-tier components/servers are going to be implemented using DCOM, therefore DCOM is considered part of our design consideration.

- The number of servers should be configurable based on the maximum workload each server can handle.
- Multiple users should be allowed to connect to the servers and be provided with application services. To support maximum performance of the servers, the concurrent accesses to the servers should be allowed; to avoid the possible errors caused by the race conditions, each critical operation should be protected by either the critical sections or semaphores.
- The middle-tier servers should control the number of requests to the data-tier to reduce the workload imposed on the database. Thus the frequently requested data should be kept in the memory of server computers after being retrieved from the database. To maintain data consistency, the servers should update all the data copies stored in the server computers and the copies stored in the client objects for each data update. Critical sections should be used in these active data updates.
- To make the best use of the data loaded from data server through network, and thus reduce the data traffic over the network, data servers should be implemented as persistent DCOM server so that the loaded data is always there to serve the user requests. This can be achieved by implementing the data control server as NT service.

4.2.3. Data-tier — Database Access

From the requirements for the data-tier, the following points need to be considered in the design of data-tier component.

- To reduce workload imposed on database, the number of database accesses must be minimized.
- A set of services will be presented to the middle-tier servers by the data access server. All the database operations should be conducted through the public interface of the data access server, and be encapsulated into the data access server to avoid invalid access to the database by the outside world.

4.3. Task Analysis

There are two distinguished tasks involved in the ESSS: information display task and information update task. Each of these tasks corresponds to one specific type of user action, that is: an employee views the employment-related information and an employee updates or changes the employment-related information.

The detailed descriptions of these tasks and their hierarchical task analysis diagram (HTA) are respectively given as below:

- **Information Display Task**: The task starts from login session. After successful login, the employee can perform information display task that includes employee personal information display, most recent pay information display, dental coverage information display, medical coverage information display, vision coverage information display, basic life coverage information display, and retirement savings plan account information display. The task ends when the required information is displayed to the user. The HTA diagram is shown in Figure 3.

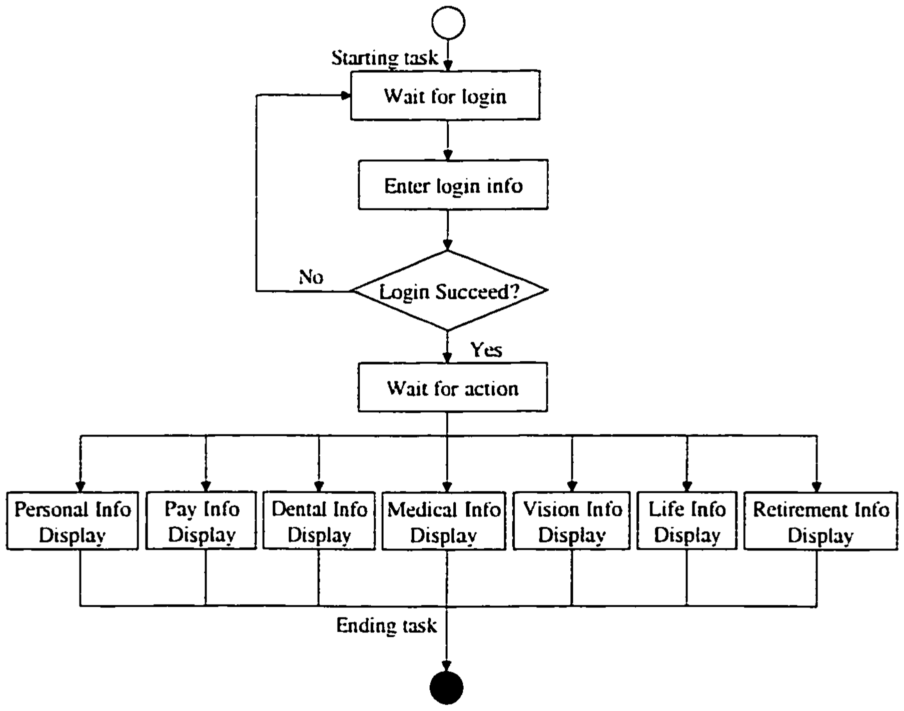


Figure 3. Task Analysis Diagram for Information Display

- **Information Update Task:** the task starts from login session. After successful login, the employee can perform information update task that includes employee personal information update, retirement savings plan fund reallocation, retirement savings plan investment direction change, and future retirement savings plan contribution rate change. The employee will be prompted an information update page to enter his new information. The task will end when user submits the new information and receives an acknowledgment of successful update. The HTA diagram is shown in Figure 4

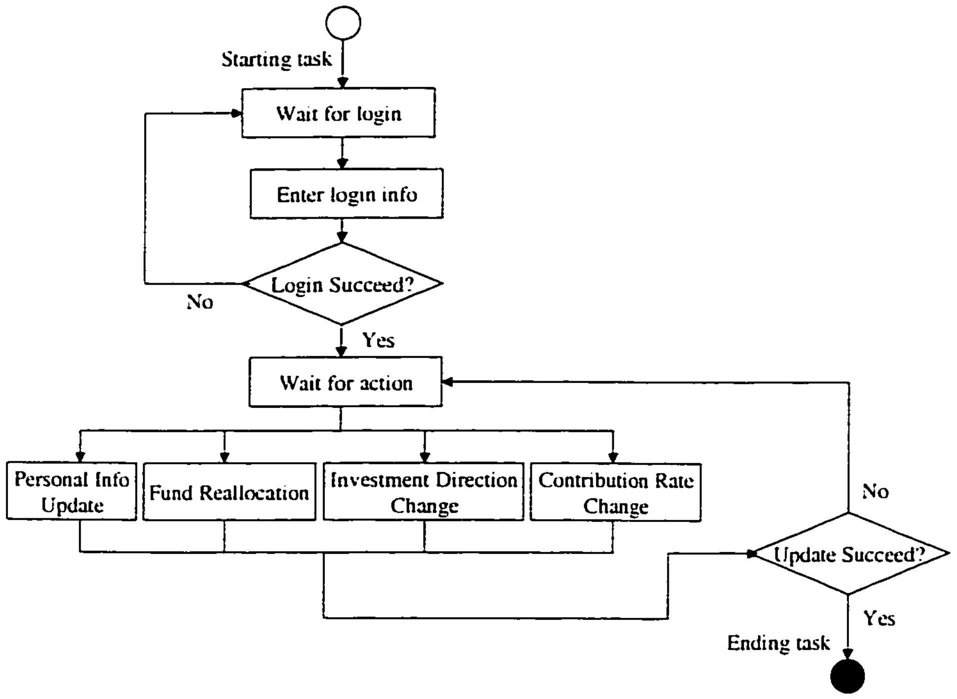


Figure 4. Task Analysis Diagram for Information Update

4.4. System Architecture

The ESSS architecture design is shown is Figure 5. The entire system consists of five basic modules, the *GUI* module, the *GuiBase* module, the *Data Control* module, the *Security Manager* module, and the *Data Access* module.

User interacts with the system via *GUI* module. The *GUI* module provides the users with convenient graphic user interface. The user can either get the information from the system or enter the data into the system through the graphic user interface.

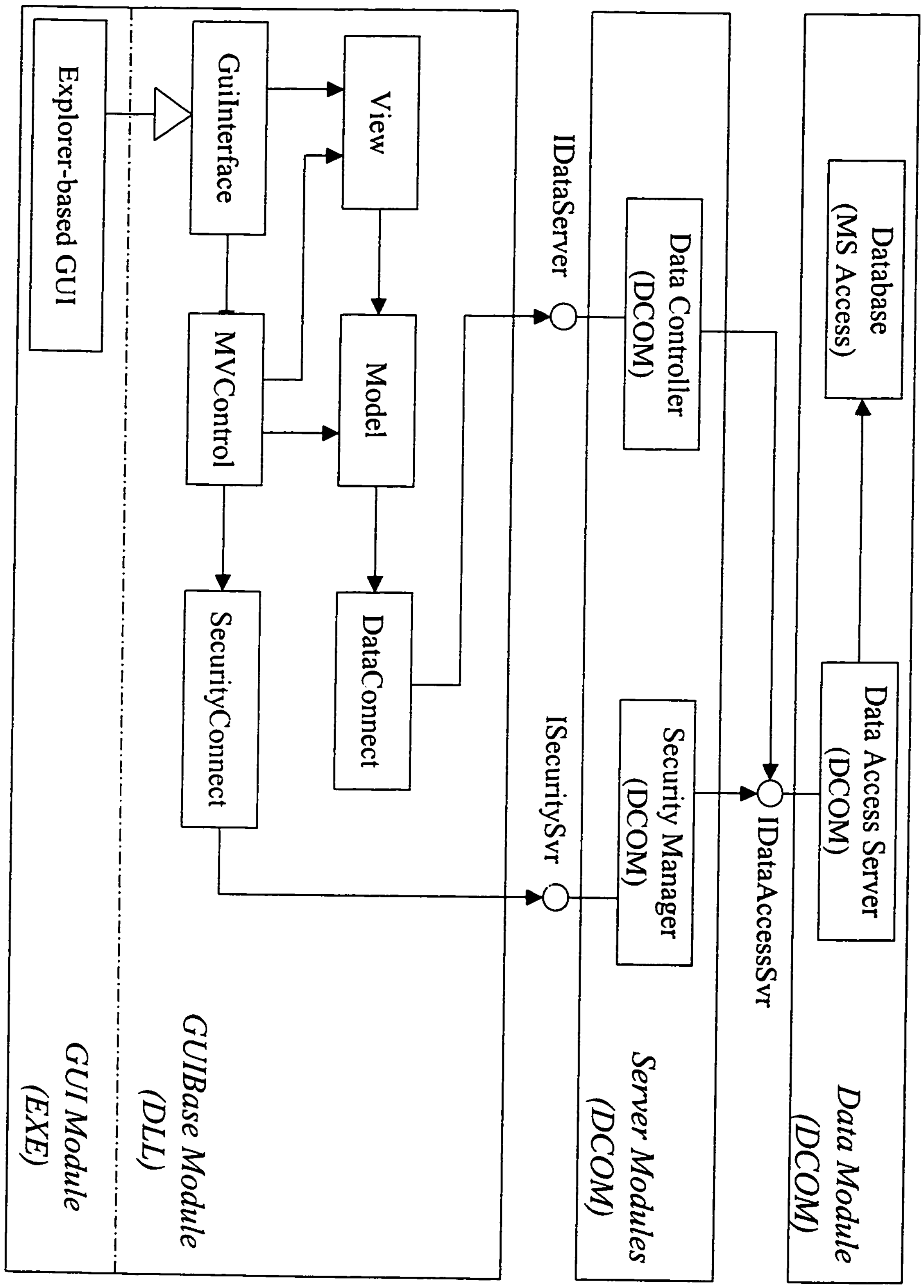


Figure 5. the ESSS Architecture

A *GuiBase* module is built on top of *GUI* module to support the processing of user requests through the *GUI* module. The *GuiBase* module interacts with the outside world via *CGuiInterface* class that exports the necessary functions for use by *GUI* module. The *CUiView* class is responsible for getting the required information for displaying the graphic user interface, while the *CUiModel* class is responsible for managing and processing the user data. The *CMVControl* class is responsible for creating and deleting the Model and View objects as needed. At initialization stage, *GuiBase* module creates the proper Model and View objects. It then establishes the connections between *GuiBase* and the middle-tier servers — *DataControler* and *SecruityMgr* via *CDataConnect* and *CSecurityConnect* classes.

As mentioned above, two middle-tier servers, the *DataControler* and the *SecurityMgr*, are to be implemented using DCOM technology. One of the great benefits gained by using DCOM is its capability of dynamic invocation and termination of the servers. The server is launched when there is an active request to the server. The server automatically shut down when released by the user process. Thus the system resource is consumed only when there is an active connection to the server. Since a PC has limited system resource, this mechanism will improve PC performance.

The basic functionality of these middle-tier servers is to provide necessary application services to the *GuiBase* module so that the information display and update tasks could be completed. These application services are exported by the public interfaces — *IDataServer* and *ISecuritySvr*. In addition to the application services, middle-tier servers should also keep a copy of the essential data frequently requested by the user. After the data is retrieved from the database for the first time, the essential data is loaded into the memory of the server computer. When the second user requests the same data as the first user does, the corresponding middle-tier server simply get the data from its data storage and immediately returns it to the user via *GuiBase* module. Therefore database access is avoided for any subsequent data requests, and the workload for database is thus greatly reduced. The data services are provided by the *DataAccess* server through the public interface *IDataAccessSvr*. *DataAccess* server is also implemented by DCOM technology. It directly accesses the database through its inner class that encapsulates all the database operations. The only way for the outside world to access the database is using the methods provided in *IDataAccessSvr*.

4.5. System Design — Object Model

The diagrams displayed in Figures 6, 7, 8, 9, 10 present the modules and their classes in details.

- The class diagram for *GUI* module is presented in Figure 6. When the system is started, *CActionCtroler* graphic object will be created. Upon the run-time situation, it creates the other graphic objects. It also initiates the other parts of the system by calling into *GuiBase* Module through the exported class *CGuiInterface*.
- The class diagram for *GuiBase* module is presented in Figure 7. Inside *GuiBase* module, only *CGuiInterface* class is exported to the *GUI* module. At the initial stage, all the related objects are properly created. When a function call is made from *GUI* module to a function defined in the exported class *CGuiInterface* of *GuiBase* module, the *CGuiInterface* calls into other objects of *GuiBase* to get the services requested by the *GUI* module. These other objects may further request the middle-tier services by

calling the public functions defined in *IDataServer* and *ISecuritySvr*. These public interfaces are respectively implemented in *DataControler* module and *SecurityMgr* module.

- The class diagram for *DataControler* module is presented in Figure 8. When a data request is received from *GuiBase* module, it will first search its data storage in *CDataStorage* class. If necessary it will call the public functions defined in *IDataAccessSvr*, which is implemented in *DataAccess* module. It then returns the requested data to *GuiBase* module.
- The class diagram for *SecurityMgr* module is presented in Figure 9. When a password information is requested from *GuiBase* module, it will first search its data storage in *CDataStorage*. If data not found, it will call the public functions defined in *IDataAccessSvr* to get the password information and return it to *GuiBase* module.
- The class diagram for *DataAccess* module is presented in Figure 10. When data request is received from either *DataControler* or *SecurityMgr*, it will get the data records from the database and return them to the caller.

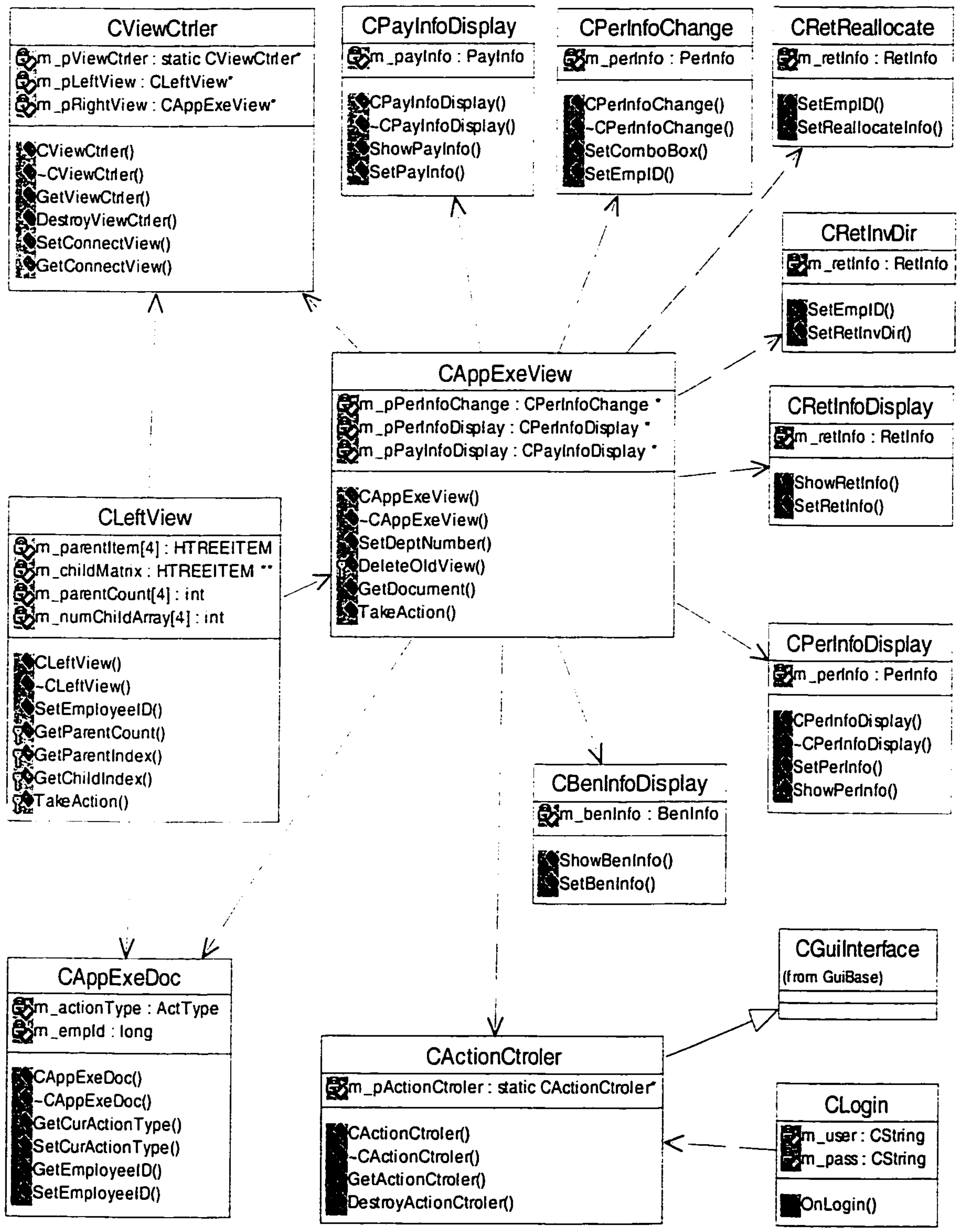


Figure 6. Class Definitions for GUI Module

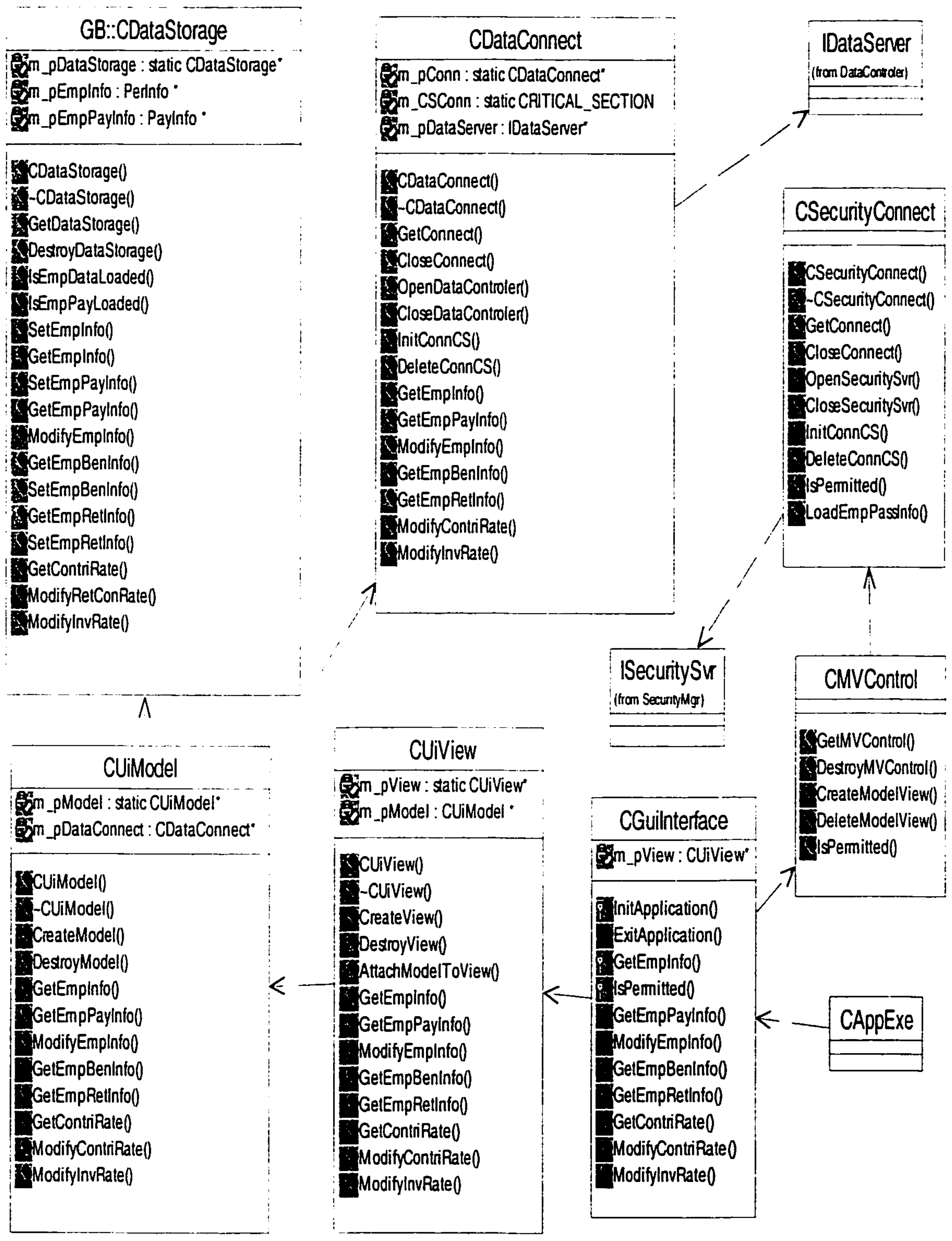


Figure 7. Class Definitions for GUIBase Module

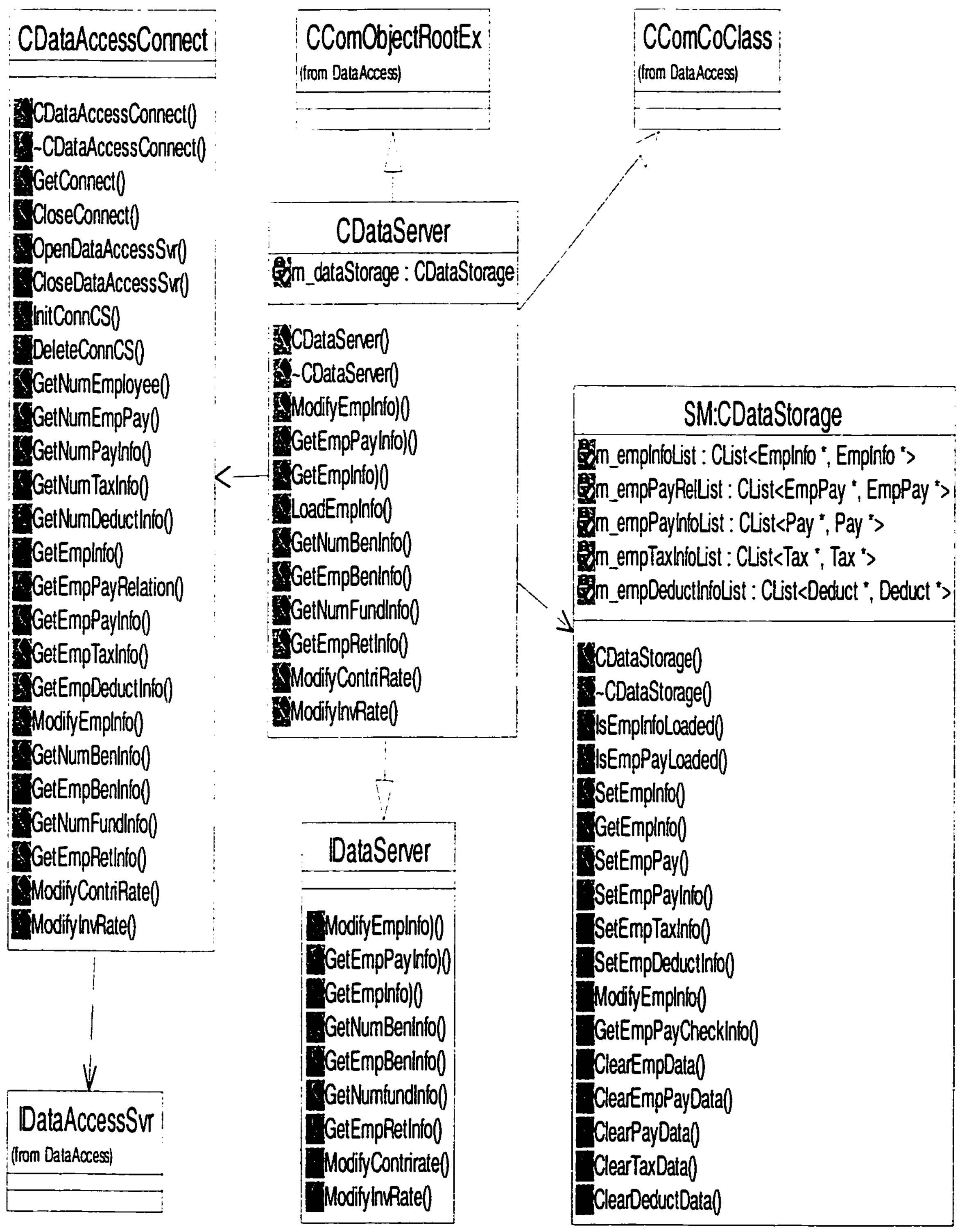


Figure 8. Class Definitions for DataCtroler Module

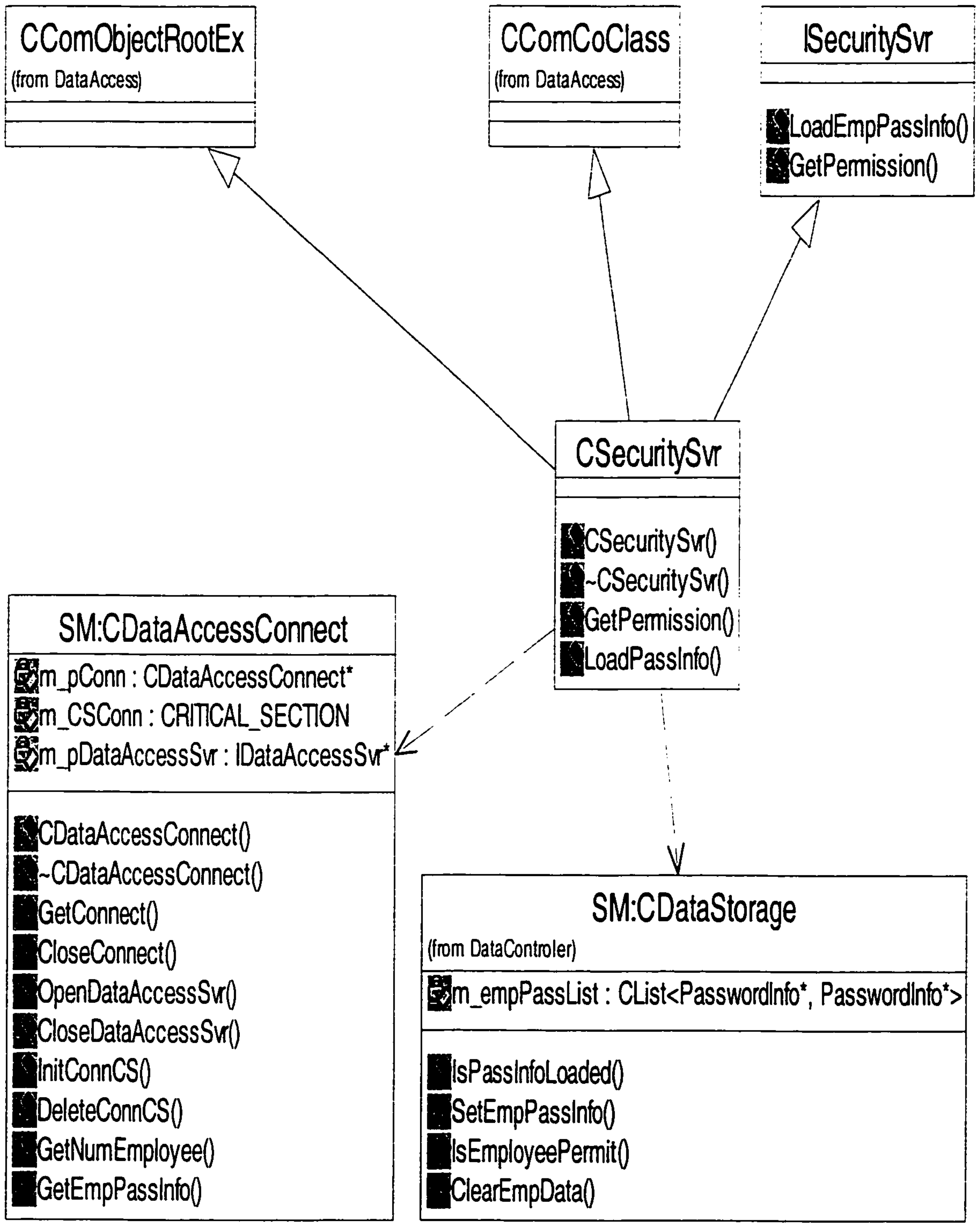


Figure 9. Class Definitions for SecurityMgr Module

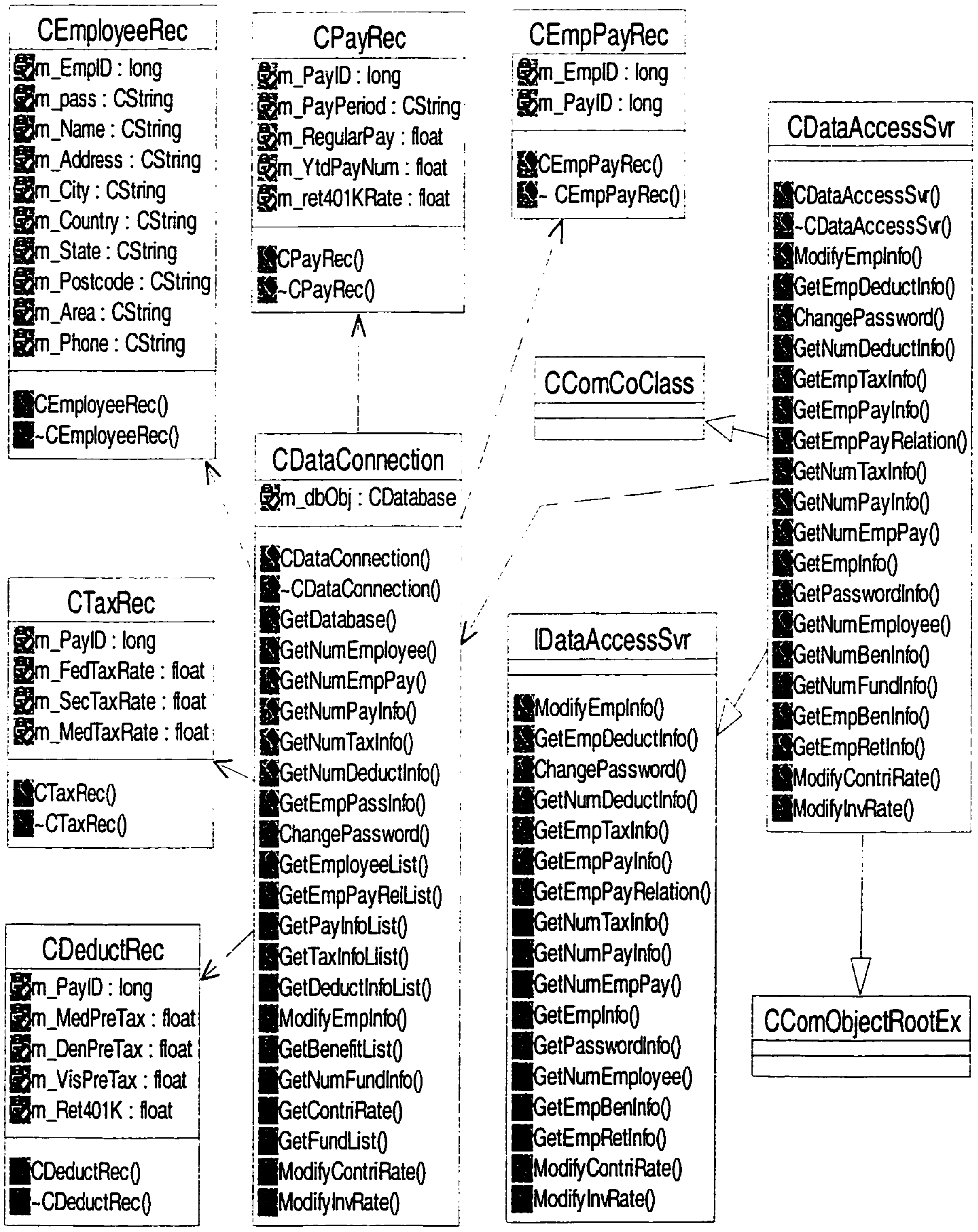


Figure 10. Class Definitions for DataAccess Module

4.6. System Design — Dynamic Model

We present sequence diagrams for the major use case scenarios. These are respectively displayed in Figures 11, 12, 13, 14, 15.

- The use case sequence diagram for loading an application is presented in Figure 11. The use case starts from a call to *InitApplication()* which in turn creates *Model* and *View* objects. Then it establishes the connections with middle-tier servers by calling *CoCreateInstanceEx()*.
- The use case sequence diagram for exiting an application is presented in Figure 12. The use case starts from a call to *Exit()* which in turn deletes *Model* and *View* objects. It then disconnects middle-tier servers from *GuiBase* by calling *Release()*.
- The use case sequence diagram for user login is presented in Figure 13. The use case starts from user entering password information. It goes into *SecurityMgr* to check the user permission by calling *IsPermitted()*, and create the data explorer window if permission is granted.
- The use case sequence diagram for personal information display is presented in Figure 14. The use case starts from a user requesting to view his personal information. The request is transferred to the data-tier server through middle-tier *DataControler* server. Finally the data is retrieved from database by the data-tier server and returned to the user via *DataControler* server.
- The use case sequence diagram for personal information update is presented in Figure 15. The use case starts from user entering new information through information update page. The new data is transferred to the data-tier server through middle-tier DataControler server and finally set into database.

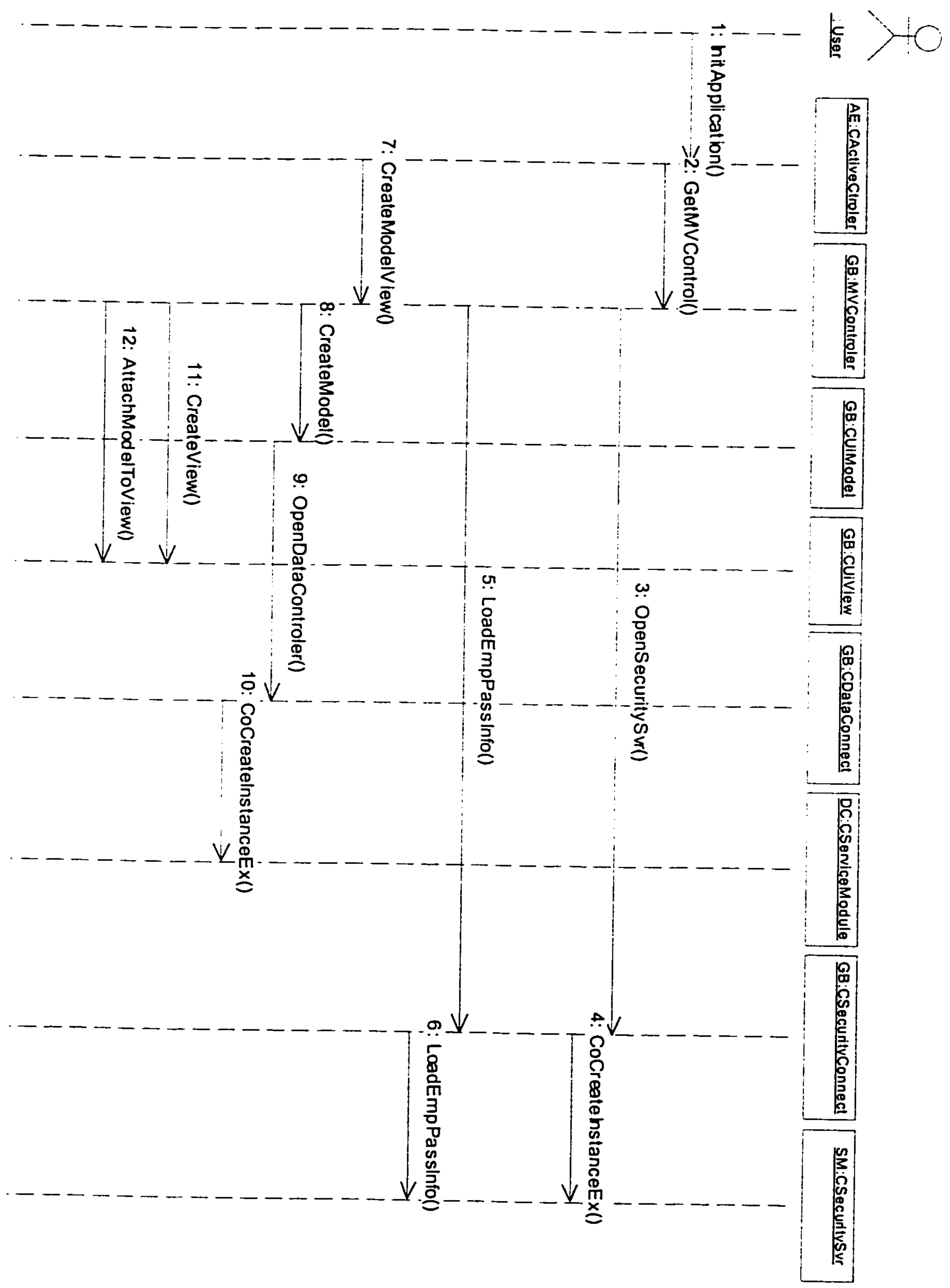


Figure 11. Use Case Diagram for Loading Application

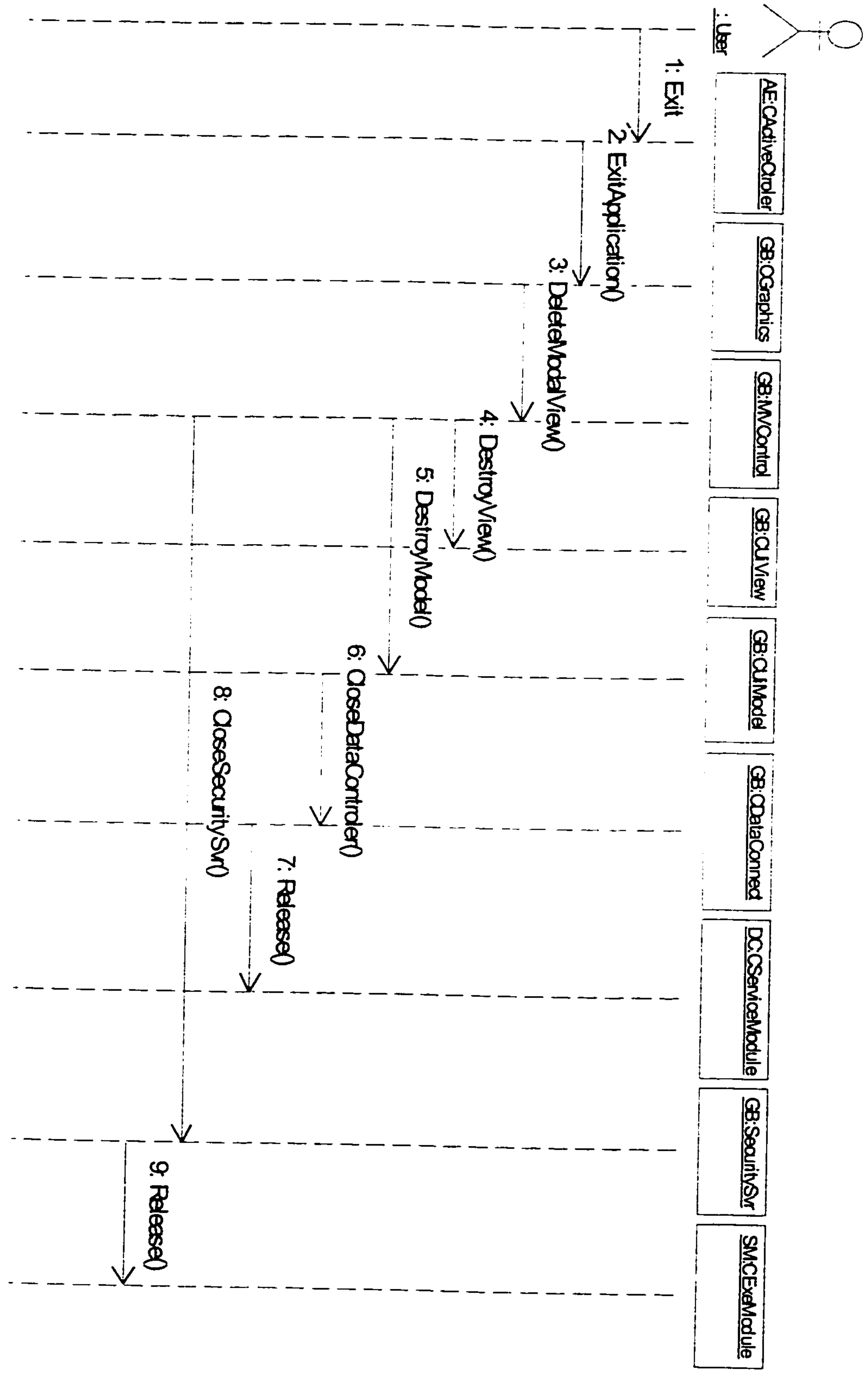


Figure 12. Use Case Diagram for Existing Application

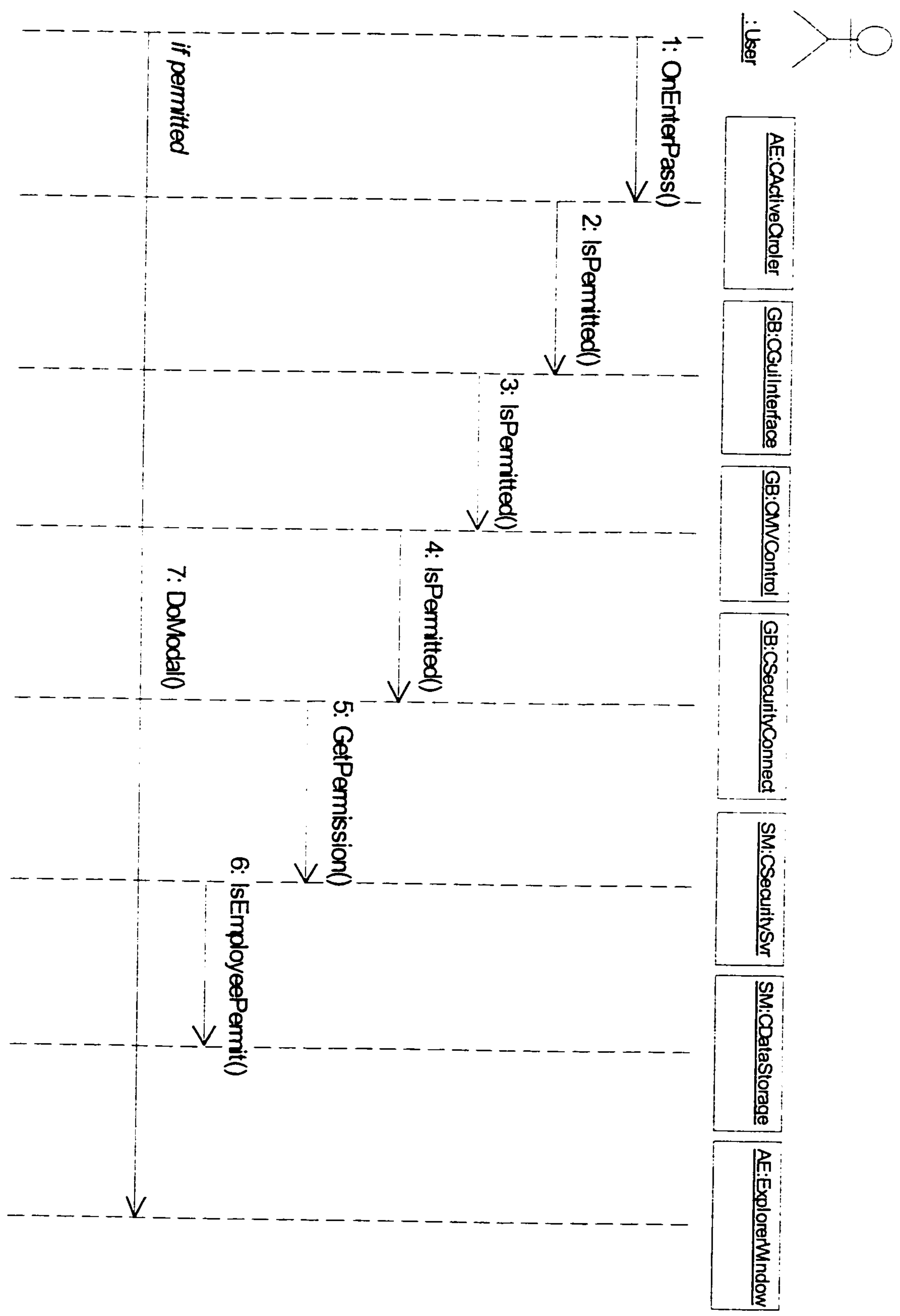


Figure 13. Use Case Diagram for User Login

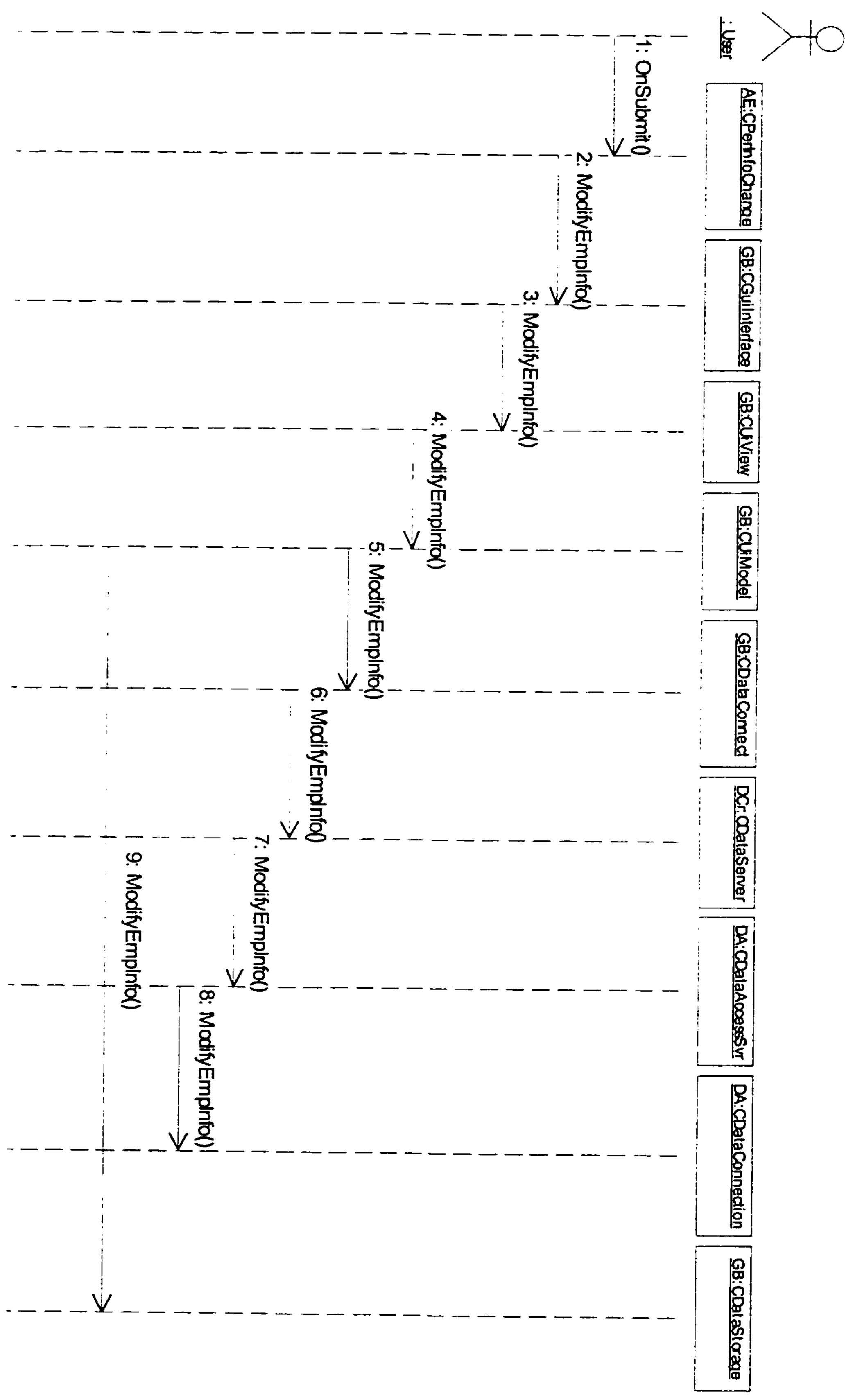


Figure 14. Use Case Diagram for Personal Info Display

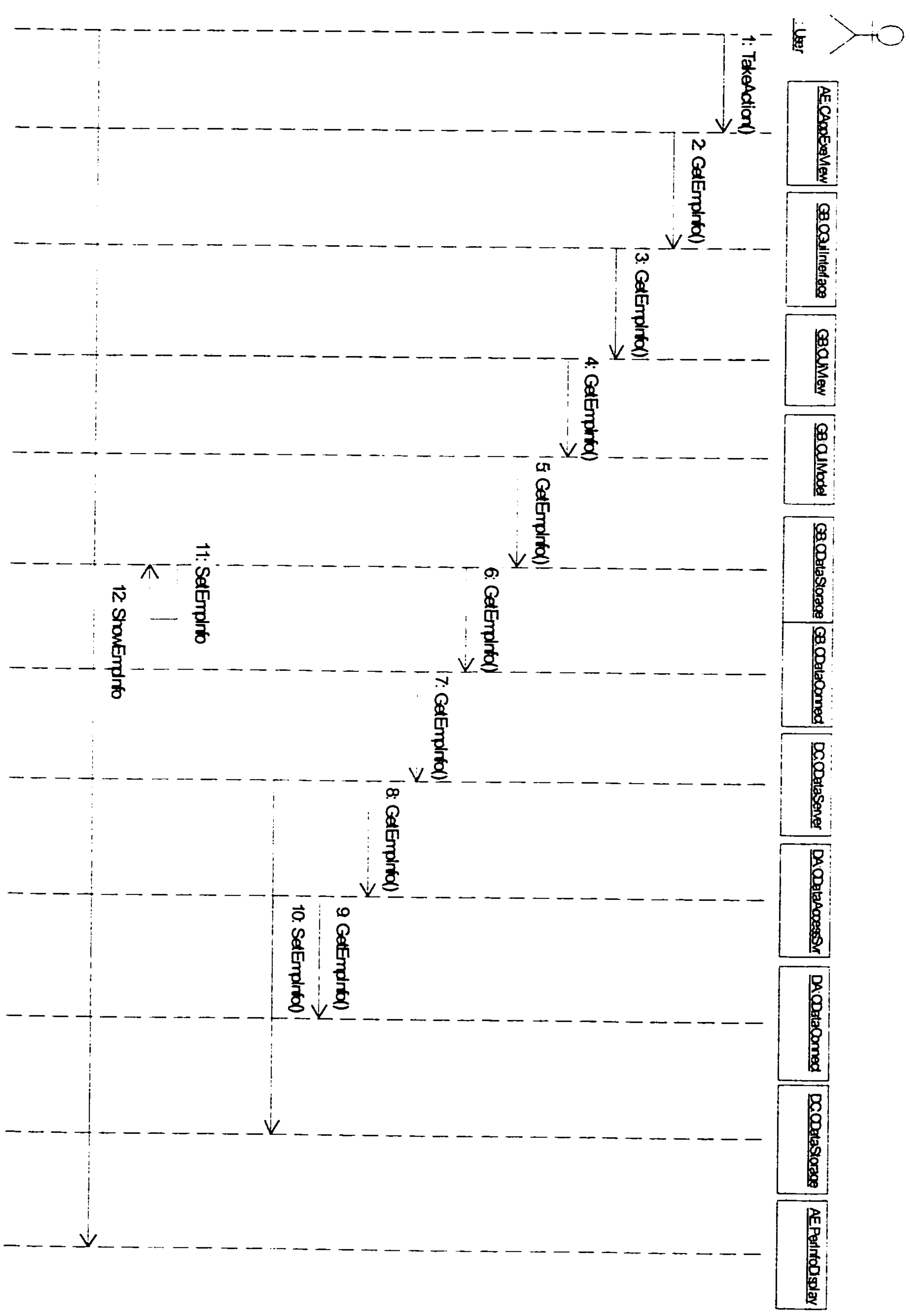


Figure 15. Use Case Diagram for Personal Info Update

4.7. Data Flow

Figure 16 describes how the data is retrieved and stored between the three tiers in the distributed system. When the application is started, the login information is loaded from the data server via middle-tier, *SecurityMgr* server. The login window is then presented to the user. After the user enters the login information, the system compares the entered information with loaded login information. A successful match will bring the data explorer window to the user. For information display, the data will be loaded from data access server via middle-tier server and displayed to the user with the requested information. For information update, the new data will be entered by the user and then be transferred to the database via middle-tier servers and data-tier servers.

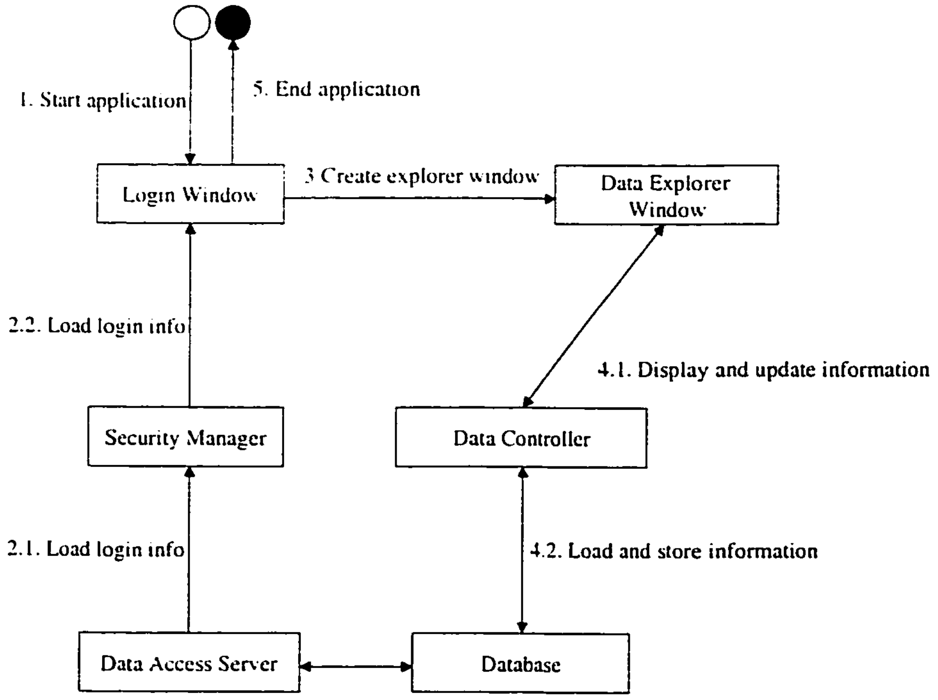


Figure 16. Data Flow and Sequence Diagram

5. System Implementation

The system implementation directly determines the reliability and usability of the ESSS. An efficient implementation of graphic user interface is the key to the effective use of the system since the GUI provides the only way for the interaction between the system and its users. To achieve top system performance, the server components must be implemented carefully. An analysis of the system features is also needed for a complete implementation of the ESSS.

5.1. Implementation Consideration

As mentioned earlier, the ESSS is designed to be a real-time, distributed system. Thus it is advantageous to use C++, Visual C++, distributed component object models (DCOM) to implement the entire system. Multithread technology should be used in the implementation of server components to improve the system response time. To keep the top system performance as number of the user increases, the ESSS must support system scalability and load balancing.

5.2. Implementation Details

In this section, we respectively explain and examine the implementation of *GUI* module, *GuiBase* module, *DataControler* module, *SecurityMgr* module, and *DataAccess* module in details. Corresponding to five modules, five different program projects are constructed by using Microsoft Visual C++.

5.2.1. GUI Module — the Graphic User Interface

Based on the user descriptions and task analysis, we choose Windows Explorer style to build our user interface. The advantages of this choice are their simplicity. These on-screen controls provide contextual information for the users, allowing them to make a correct choice before proceeding. To reduce the data transmission over the network, only login information is loaded into the system from the database at the starting of the ESSS. The implementation of *GUI* module is completed according to the class definitions given in Figure 6.

5.2.2. GuiBase Module

The *GuiBase* module is the essential part of the user process. It is built as a Dynamic Linked Library, and will be loaded by the *GUI* process once the ESSS application is launched by the user. It serves as the base module for the *GUI* module. It provides all the necessary operations for the *GUI* module to complete the user tasks. It is responsible for establishing the connections with the middle-tier servers. To get the best system performance *GuiBase* also keeps the essential data of the ESSS in its local memory.

One of the classes, *CActionCtroler*, defined in *GUI* module is inherited from class *CGuiInterface* in *GuiBase* module. Thus every protected or public method in *CGuiInterface* class is a part of members in *CActionCtroler* class. At the initial stage of the ESSS, *CActionCtroler* is created by the system, which in turn calls *InitApplication()* defined in *CGuiInterface*. The primary task of *InitApplication()* is to create the Model and View through Model-View Controller, it then establishes the connections with two middle-tier servers — the *DataControler* and the *SecurityMgr* by calling *CoCreateInstanceEx()* to create the remote server object. *CoCreateInstanceEx()* will return status information in HRESULT type. This status information informs us whether or not the remote server object is created successfully. We can use the following code to test its status:

```
HRESULT hr = CoCreateInstanceEx(...);
if (SUCCEEDED(hr)) {
        // continue
}
else {
        // error handling and terminate program
}
```

Although in most of the cases, *CoCreateInstanceEx()* and some other functions return zero for signaling success, it is not safe to simply test if it is zero since there are some other cases in which successful operations return nonzero. In all cases, we should use macro *SUCCEEDED* or *FAILED* to test the operation status. The implementation of *GUIBase* module is completed according to the class definitions given in Figure 7.

5.2.3. DataControler Module

DataControler server is one of the middle-tier servers and is the essential component of the ESSS. It provides most of the application services required to complete the user tasks. All the dynamically changeable data is managed by the *DataControler* server and the amount of these data is about four-fifths [or "80%"] of the total data managed by middle-tier servers. In addition, *DataControler* server provides more application services than the *SecurityMgr* server does. As a result, *DataControler* might need to be executed in several powerful PCs in which all the system resources should be reserved for providing the ESSS services.

When implementing DCOM server, one must decide which type of server is most appropriate. One type of server, the ordinary DCOM server, always unloads itself when no more client connecting to it. The other type of server is the so-called "NT service", a persistent server that exists even when there is no more client connection. The advantage of ordinary DCOM server is its automatic release of system resource. This characteristic is crucial for an ordinary PC where system resource is limited. The benefit of the NT service server is its ability of reducing the amount of network traffic. Assume only one client is currently connected with a DCOM server of NT service type, and assume that the required data is already loaded into DCOM server. After finishing the user task, the client calls *Release()* to releases server and then quits. Although there is no more client connection to the server, the server does not unload itself and the data managed by the server still exists. When another client makes a new connection to the server, the existing data can be reused. As a result, the subsequent network data transmissions and database accesses are avoided. If ordinary DCOM server is used, all the data kept in the DCOM server will be destroyed when the server unloads itself. When another client makes a new connection, the server must reload the data through the network transmissions and database accesses.

Since there is a significant amount of data managed by the *DataControler* server, we implemented *DataControler* server as an "NT service" to avoid frequent data transmissions and database accesses. To protect multiple clients from accessing the shared data simultaneously, we use a critical section as shown in the following format:

```
EnterCriticalSection(&m_clientlSect);
```

```
        //enter critical operation codes here
        LeaveCriticalSection(&m_clientISect);
```

The implementation of *DataControler* module is based on the class definitions presented in Figure 8.

5.2.4. SecurityMgr Module

SecurityMgr is a middle-tier server responsible for controlling user access to the system. Upon receiving the client request for verifying password information, *SecurityMgr* will look at its local memory to determine whether or not the password information is already loaded from the data server. In case the password information is not loaded, it will loads the password information from data access server, sets the password information into its local memory, and then return password information to the client. *SecurityMgr* only manages a small amount of data — the password information, so it is implemented as ordinary DCOM server. When no more active client is connected to *SecurityMgr*, it will unload itself to release the system resource. Although the next client request will cause a new data transmission, the workload imposed on the network will not be very heavy because the amount of data transmitted over the network is limited. The complete implementation of the *SecurityMgr* module is conducted according to the class definitions given in Figure 9.

5.2.5. DataAccess Module

DataAccess is a data-tier server responsible for offering the data services for the middle-tier servers. To guarantee the safe operation of database accesses from the outside of the data access server, the database operations are encapsulated in the data access server. To do so, we implemented *DataAccess* server to export a set of services and functions. The database can be accessed by the outside world only through these services and functions defined in the public interface. Thus if these services and functions are carefully coded, the safe database operations will be guaranteed.

As discussed in the previous sections, middle-tier servers are implemented to minimize the database access by keeping the frequently requested data into their local memory. For this reason, *DataAccess* server does not need to keep the data retrieved from database. Once database service is requested, it opens database and makes corresponding database operation. It immediately closes the database after the database services are completed. This will help the system keep the minimum number of database connections. The implementation for the *DataAcces*s module is performed according to the class definitions presented in Figure 10.

5.2.6. Project Settings

To complete their common tasks, dependency relationships between these modules must be preserved. In order to successfully compile and build these modules, we need to make appropriate settings for these different projects.

- *DataAccess* module is built as a data-tier server component, only providing the data access services to the other modules. So it does not depend on any other modules. As a result, no specific settings are needed to build this module.
- *DataControler* module is built as a middle-tier server component, an NT service. It not only provides the application services to the client, but also requires the data services from the data-tier server when needed. Thus we need to do some settings for

DataControler module. First of all, we must include a header file named "dataaccess.h" that tells the compiler the complete definition of data-tier services available for the *DataControler* module. To allow the compiler to find the specified include file, we need to add a corresponding path to the project path-include list, which can be done by using the sub-menu "options..." of the main menu "Tools". Moreover we must add a file named "DataAccess_i.c" into the project, which allows the compiler to know all the interface definitions. This file can be added by using the sub-menu "Add to project" from the main menu "Project".

- *SecurityMgr* module is built as a middle-tier server component. It has the similar functionality as *DataControler* component. Therefore same settings need to be performed for *SecurityMgr* module.
- *GuiBase* module is built as a presentation-tier DLL component, responsible for initializing the application, establishing the connections to the middle-tier servers, requiring the needed application services from middle-tier servers, and providing the corresponding data to the graphic user interface. Therefore *GuiBase* module depends on *DataControler* module and *SecurityMgr* module. To successfully compile the *GuiBase* module, the header files, "DataControler.h" and "SecurityMgr.h" respectively from *DataControler* and *SecurityMgr* projects, must be inserted into the project. To allow the compiler to find the interface definitions for all the middle-tier servers used in the *GuiBase* module, the files "DataControler_i.c" and "SecurityMgr_i.c" must be added into the *GuiBase* project.
- *GUI* module is built as a presentation-tier EXE component, responsible for starting the ESSS application and providing the graphic user interface to the users. It will load

"GuiBase.dll" into its workspace and requires application services from the middle-tier servers through the functions implemented in the *GuiBase* module. Thus *GUI* module depends on *GuiBase* module. All the functions accessible to the *GUI* module are exported from the class *CGuiInterface* of *GuiBase* module. Therefore the header file, "*GuiInterface.h*", needs to be included in the *GUI* project. To successfully compile the project, the compiler also needs to know the external functions provided in the *GuiBase*. For this purpose we need to add "*GuiBase.lib*" into the *GUI* project, which can be done by adding "*GuiBase.lib*" into the project settings.

6. Installation and Execution

After the implementation of the ESSS is completed, we need to distribute and configure the different components of the ESSS into their designated computers so that they can work together in a distributed environment.

6.1. The Distribution of Dynamic Linked Library

During initialization of the ESSS, the "*GuiBase.dll*" generated from *GuiBase* module will be loaded into the workspace of "*ESSS.exe*" application. To allow the "*ESSS.exe*" to find the "*GuiBase.dll*", "*GuiBase.dll*" must be set in one of the following locations:

- Windows NT system32 directory.
- The directories specified in the path setting of Windows NT system environment.
- The same directory where the executable "*ESSS.exe*" locates.

The easiest way is to set the output files of the *GUI* project and the *GuiBase* project to the same common directory. This can be achieved by compiling the object files into their common locations.

6.2. Server side configuration

Each DCOM server must be registered into the system registry of the computers where they can be launched upon client's request. Usually the server project is built by "ATL COM AppWizard" to support automatic registration of the server interface and their classes at the compile time. Therefore, if the server is built on one computer and is going to run on the same machine, the registration step is not needed for that server. For *DataControler* server, we should repeat the server registration process. This is because that only the ordinary DCOM server can be registered by the automatic registration. On the other hand, if the server is built on one computer and is to be executed on another computer, we also need to do the server registration process. If a server is implemented as NT service, it can be either registered as ordinary DCOM server or registered as "NT service". Otherwise it can be registered only as ordinary DCOM server. To register *DataControler* as "NT service" we enter the following command at command line:

DataControler /Service

To register *SecurityMgr* as ordinary DCOM server, we use the following command:

SecurityMgr /RegServer

The registration for all the other ordinary DCOM servers uses the same process as that of *SecurityMgr* server.

For *DataControler* server, we can remove the server registration from the system registry by using the following command at command line:

DataControler /UnregServer

All the other servers follow the same un-registration process.

When server registration is done, we have to set the server access permission and launch permission, which can be performed by using "DCOMCNFG.EXE" provided in the Windows NT package. When "DCOMCNFG.EXE" is executed, a DCOM server configuration dialog box appears. First of all, one should make sure that the check box "Enable Distributed COM on this computer" is checked on the "Default Properties" page. The other settings on this page should be kept unchanged. We then need to go to the "applications" page, which displays all the DCOM objects and interfaces being registered in the system. To make a configuration for a particular server, one should find and select the server to be configured, then click "properties" button. Another dialog box will appear. To allow particular clients capable of launching and accessing the DCOM server remotely, one should set the related options from Security page and Identity page. From "Security" page, one can give the access permission to the particular users from remote computers by clicking the corresponding "Edit..." button and then add these users onto the *AccessPermission* list. Similarly, one can give launch permission to the particular users from remote computers by clicking the corresponding "Edit..." button and then add these users onto the *LaunchPermission* list. From "Identity" page, one should select "This user" from three of the option radio buttons. By clicking "Browse..." button, One can select a user from user list. The selected user should have an account on that particular server computer and should be able to launch the server. One should also type in the password needed for that particular user to log onto that computer. After the above process is completed, the server is ready to receive the client requests and provides the services to the client.

6.3. Client side configuration

Client side configuration includes the registration for the Proxy and the registration for the remote server.

6.3.1. Registration for the Proxy

Proxy is used by the client in the process that performs *Marshalling* and *Unmashalling* for the calls between the client and the server. Before registering the server Proxy, one should generate Proxy DLL using "nmake.exe" program provided by the Microsoft visual studio. When server project is established, a Proxy makefile is generated by the AppWizard. The name of the make file is set by the AppWizard to be the project name postfixed with "ps.mk". In the following we use *DataControler* server as an example to explain how to generate and register the Proxy DLL.

Among the project files for *DataControler* server, one can find a Proxy makefile named as "DataControlerps.mk". This file should be copied to the computer on the client side to generate a Proxy DLL and then make proper registration. One can enter the following command at the command line to generate the Proxy DLL file:

nmake DataControlerps.mk

"nmake.exe" is one of the Microsoft program maintenance utilities that builds projects based on commands contained in a description file. The output of "nmake.exe" for Proxy makefile is the Proxy DLL file named as "DataControlerps.dll". This Proxy DLL can be registered using "regsvr32.exe" program that is provided with Windows NT package. At the command line, type the following command:

regsvr32 DataControlerps.dll

The result of the registration, either successful or failed, will be informed to the user. If one wants to remove the Proxy registration after it successfully registered, one can use the following command:

regsvr32 /u DataControlerps.dll

The registrations for all the server proxies should follow the same procedures as described above. Once the server proxies are successfully registered, we can go to the next step.

6.3.2. Registration for Remote Servers

The servers should be registered on both server computer and client computer. It is natural to think that the server should be registered on the server computer since system need to find the server when a client request arrives. For the same reason, the server should also be registered on the client computer. When a client needs the services from the server, it will ask the service control manager (SCM) to create the server object for him. The SCM will look at the system registry to find the location of the server. If the server locates remotely, the local SCM will ask the remote SCM to create the specified server object. The client then gets the services from that server by calling its public functions.

There are two methods to register the server on the client computer. For the first one, just follow the same server registration procedure described in the server configuration. This method needs to copy the server component into the client computer.

The second method requires the export of the server registration information. After the server is registered in the server computer, one can search the registry database to find the corresponding server class and its interface, which can be accomplished by using "regedit.exe" program provided by Windows NT package. The execution of "regedit.exe" program provides the user with menu-supported presentation of registry database. By using the export options from the menu, one can get the registration information for the server class and its interface, and further save them as registration files. These files can be copied into anywhere in client computer. Simply double click on these registration files from Windows NT explorer, the server will be registered into the system registry in the client computer.

After the registration is completed on the client computer, one should use "DCOMCNFG.EXE" program to reset the location of the server, which allows the client system manager to launch and access the server on the specified location. Execution of "DCOMCNFG.EXE" program will bring up a dialog box to the user. Clicking on the "properties" button on that dialog box will bring up another dialog box from which one can set the location of the server. Simply deselect "Run application on this computer" and select your dedicated computer as remote server provider. The setting for the server location is then completed.

The registration procedure described here looks complicated. As mentioned earlier, there exists another method for running the distributed application without the needs to perform server registration on client computer. This method requires the client codes to pass the detailed server location when calling *CoCreateInstanceEx()* to create server object. The advantage of this latter method is its simplicity and control over the server. Its disadvantage is evident. Whenever the server changes its location, the client codes need to be changed and recompiled.

Using server registration on client computer can guarantee the server location transparency. One needs not to know the server location when implementing client codes. The disadvantage is that a particular client must always get services from a particular server, which makes it harder to achieve automatic server load-balancing.

6.4. Settings for Database Connection

For project demo purpose, the data accessed by the ESSS is stored in the Microsoft Access database. To allow the data-tier server to operate the data stored in the database, we use ODBC as connection interfaces from data-tier server to access database. To do so, we need to set a database connection using Microsoft ODBC driver for Access.

The database connection for the ODBC driver can be established by using the "ODBC" from Control Panel. Clicking on "ODBC" icon brings up "ODBC Data Source Administrator" dialog box from which one can add a new User DSN. When "Add" button is clicked, a list of ODBC drivers is presented to the user. Select "Microsoft ODBC for Access" and click "Finish" button, one is then prompted to enter the Data Source Name and its Description. Enter "Access Connection". After this information is entered, the database connection is established. One can also add a new file DSN by using the similar procedure. From the file DSN page, click "Add" button and select "Microsoft ODBC for Access" from ODBC driver list, then click "next". One is prompted to enter the name of the file data source. Enter the name as "Access Connection". The Microsoft Access database is then ready to be accessed by the ESSS data access server.

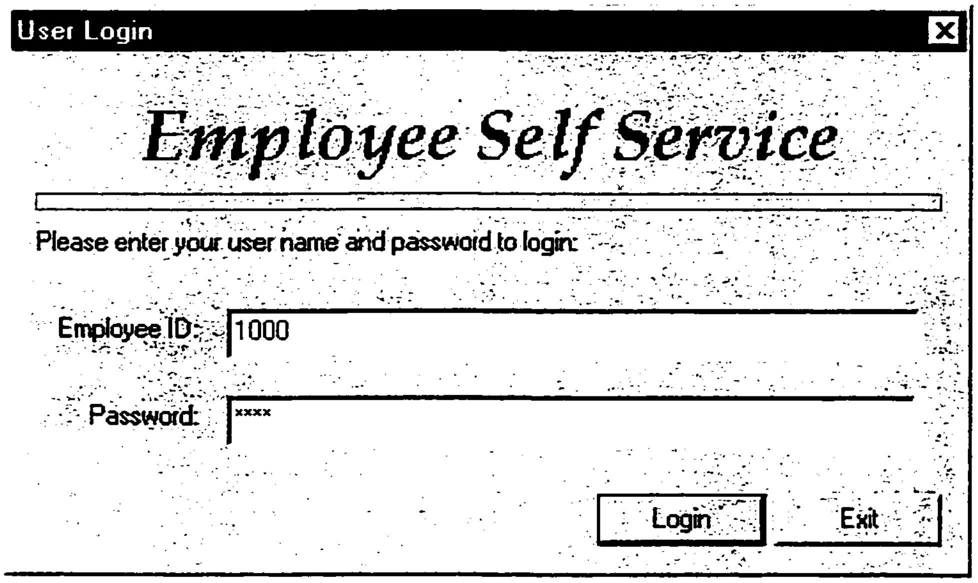
6.5. Using the ESSS

According to the configuration guidelines described above, we have conducted following settings for the ESSS to be used in a distributed environment. We used two Pentium III personal computers to execute the ESSS. Windows NT workstation is installed for these two PCs.

On the first PC, we installed Microsoft Access database with database tables properly created. We then inserted sufficient data values into these tables. We also established a new ODBC connection that is to be used by *DataAccess* server. We installed *DataAccess* server and properly register it as ordinary DCOM server. On the same PC, we installed *DataControler* server and *SecurityMgr* server. *DataControler* server is registered as NT service, and *SecurityMgr* server is registered as ordinary DCOM server. We then registered *DataAccess* server. *DataAccess* proxy DLL is also registered by using "nmake.exe" and "regsvr32.exe". On the second PC, we installed the starting program for the graphic user interface. This includes "*ESSS.exe*" and "*GuiBase.dll*". Actually the "*GuiBase.dll*" is to be loaded by "*ESSS*.exe" to its own working space and is the base of the graphic user interface. Finally, on the second PC we also made server registrations for the two middle-tier servers and configured them to run on the first PC. After the above settings, the ESSS is ready to be used from the client computer. In the following sections, we describe how the user tasks can be completed with the help of graphic user interface.

6.5.1. Login Window

Login window is shown in Figure 17. To LOG onto the system, user needs to enter a valid user ID and password. There are two buttons displayed on the login window. Clicking on "Login" button will prompt the system to process the user's login information. If login is successful, the user will be presented with data explorer window. For an invalid login, a message box will appear, informing users of the login failure. Once the user acknowledges the message, the system will flush the login fields and wait for a new login session. Clicking on "Exit" button will terminate the ESSS.



Please enter your user name and password to login:

Employee ID: 1000

Password: ****

Figure 17. Login Window

6.5.2. Data Explorer Window

After the successful login, the system presents the data explorer window to the user. As shown in Figure 18, data explorer window is a two-way split window. On the left side window displays all the functions available to the user. These functions are initially hidden inside four document folders. Upon double click on one of these folders, the hidden functions will appear and user can then make selections on these functions. The right side window is reserved for displaying and updating the employment-related information. At the start of the data explorer window, nothing will be displayed on the right side window. To exit from the ESSS, user can click on 'X' button on the upper right corner of the data explorer window.

6.5.2.1. Information Display

Users can view the specific information by clicking on the corresponding icon/function displayed on the left side of the data explorer window. These operations include personal information display, dental coverage display, medical coverage display, vision coverage display, basic life coverage display, and retirement savings plan account balance display. As shown in Figures 19, 20, 21, 22, 23, 24, and 25, both employee name and employee number are shown on the display page along with the requested information. When a new information display is requested, the old information will be flushed from the right side window, the new information will be presented.

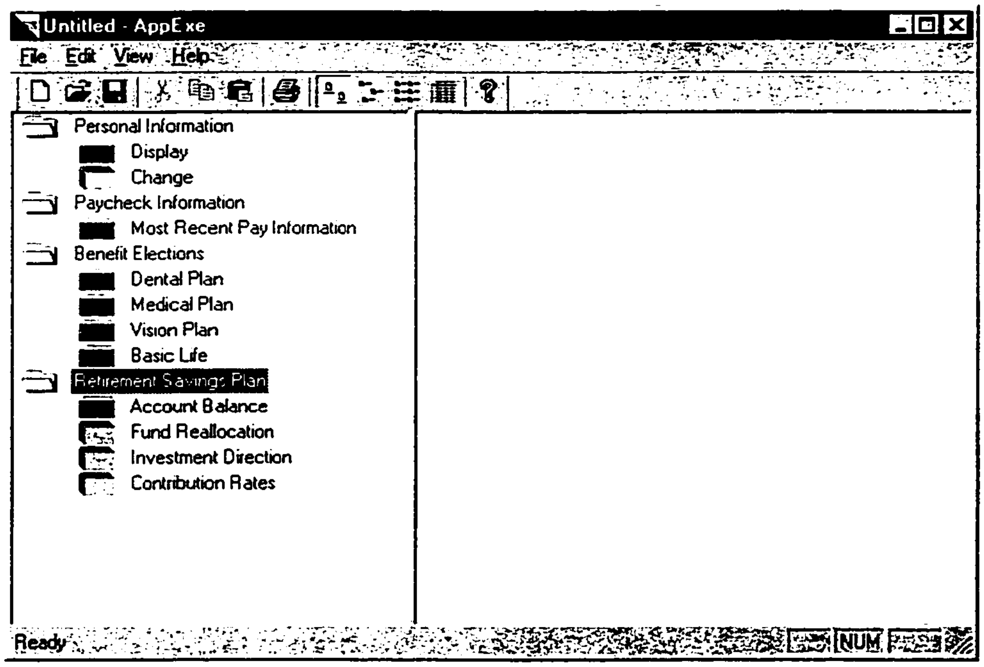
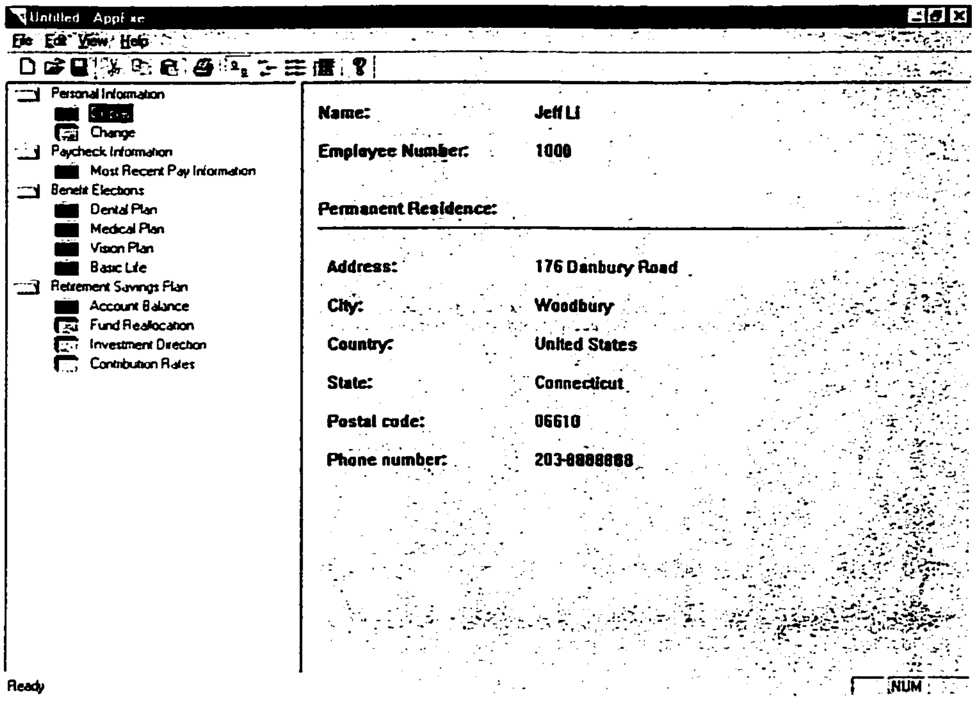


Figure 18. Data Explorer Window

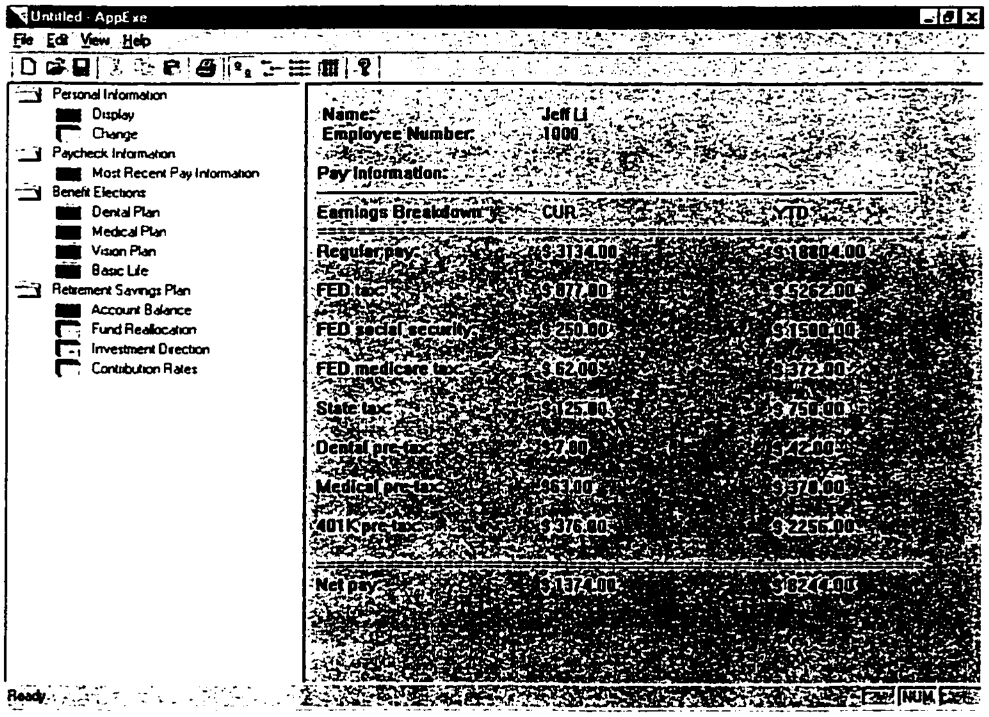


Name:	Jeff Li
Employee Number:	1000

Permanent Residence:

Address:	176 Danbury Road
City:	Woodbury
Country:	United States
State:	Connecticut
Postal code:	06610
Phone number:	203-8888888

Figure 19. Personal Information Display

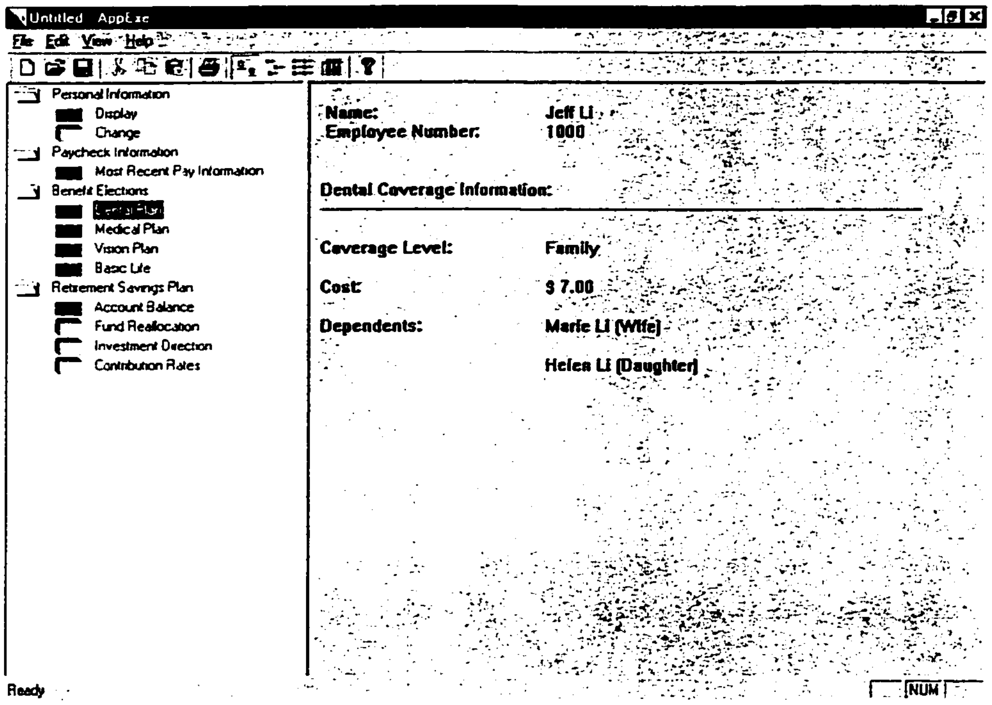


Name:	Jeff Li
Employee Number:	1000

Pay Information:

Earnings Breakdown:	CUR	YTD
Regular pay:	$ 3134.00	$ 18804.00
FED tax:	$ 877.00	$ 5262.00
FED social security:	$ 250.00	$ 1500.00
FED medicare tax:	$ 62.00	$ 372.00
State tax:	$ 125.00	$ 750.00
Dental pre-tax:	$ 7.00	$ 42.00
Medical pre-tax:	$ 63.00	$ 378.00
401K pre-tax:	$ 376.00	$ 2256.00
Net pay:	$ 1374.00	$ 8244.00

Figure 20. Pay Information Display

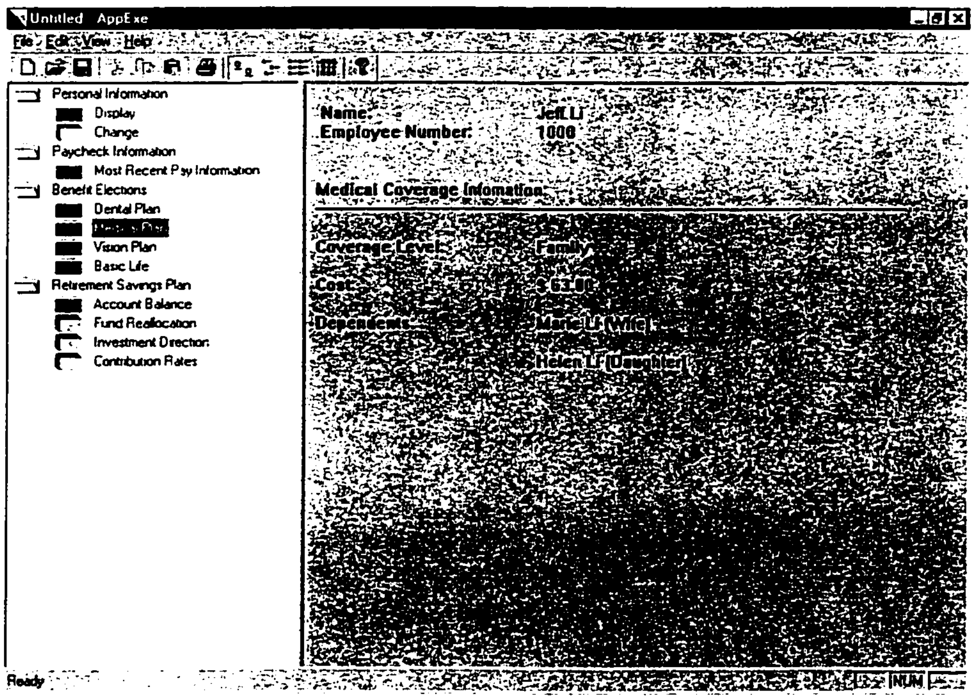


Name: Jeff Li
Employee Number: 1000

Dental Coverage Information:

Coverage Level: Family
Cost: $ 7.00
Dependents: Marie Li (Wife)
Helen Li (Daughter)

Figure 21. Dental Coverage Information Display

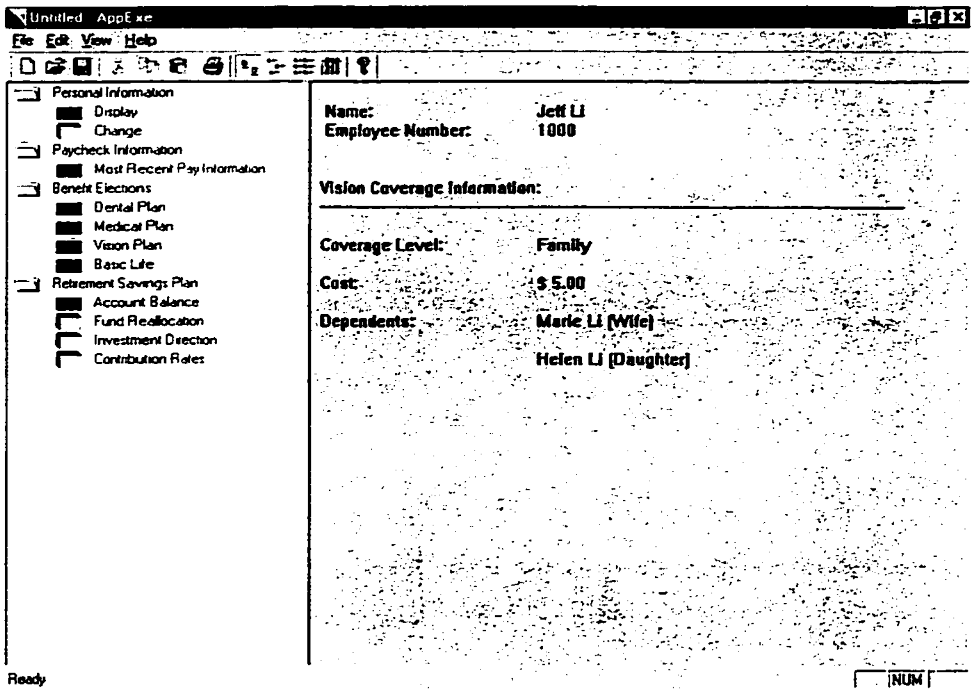


Name: Jeff Li
Employee Number: 1000

Medical Coverage Infomation:

Coverage Level: Family
Cost: $ 63.00
Dependents: Marie Li (Wife)
Helen Li (Daughter)

Figure 22. Medical Coverage Information Display

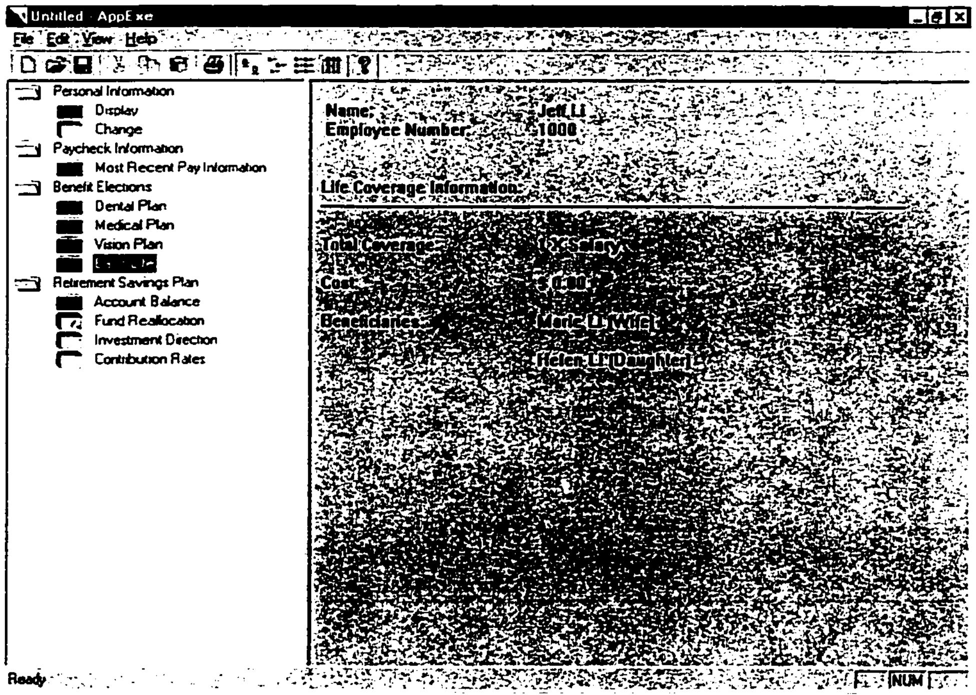


Name:	Jeff Li
Employee Number:	1000

Vision Coverage Information:

Coverage Level:	Family
Cost:	$ 5.00
Dependents:	Marie Li (Wife)
	Helen Li (Daughter)

Figure 23. Vision Coverage Information Display

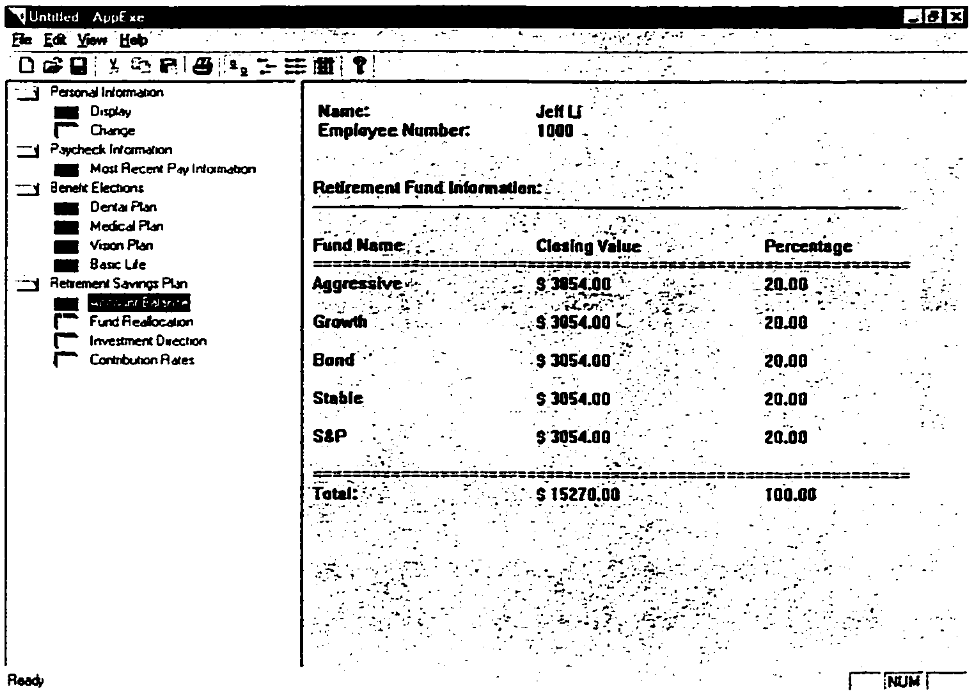


Name:	Jeff Li
Employee Number:	1000

Life Coverage Information:

Total Coverage:	1 X Salary
Cost:	$ 0.00
Beneficiaries:	Marie Li (Wife)
	Helen Li (Daughter)

Figure 24. Basic Life Coverage Information Display



Name: Jeff Li
Employee Number: 1000

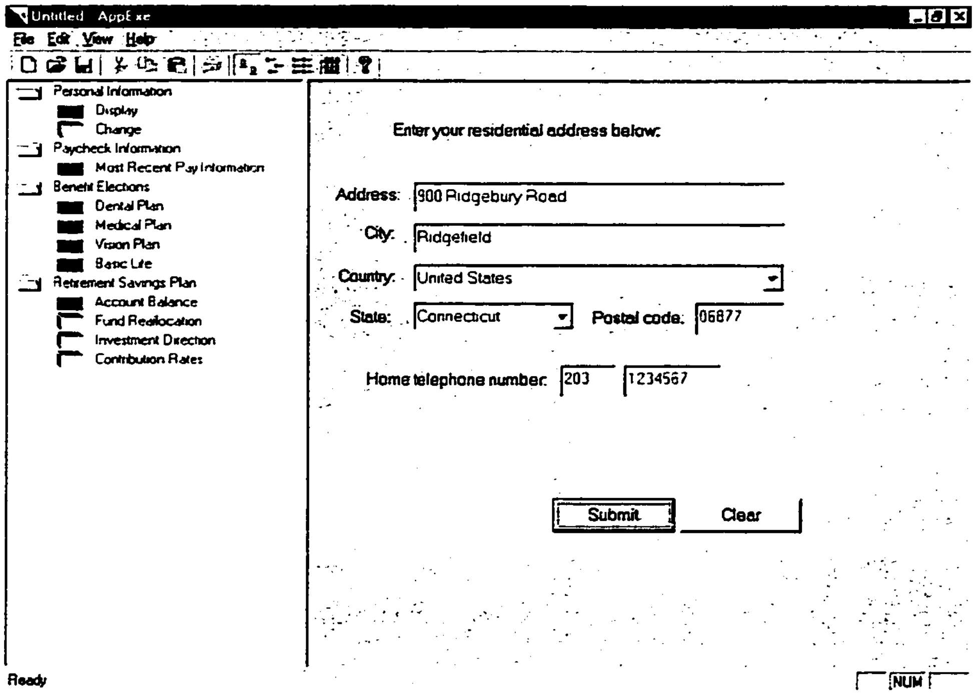
Retirement Fund Information:

Fund Name	Closing Value	Percentage
Aggressive	$ 3854.00	20.00
Growth	$ 3054.00	20.00
Bond	$ 3054.00	20.00
Stable	$ 3054.00	20.00
S&P	$ 3054.00	20.00
Total:	$ 15270.00	100.00

Figure 25. Retirement Account Balance Display

6.5.2.2. Information Update

Users can update or change the specific information by clicking on the corresponding icon/function displayed on the left side of the data explorer window. An information update page will be presented to the user. To update or change the information, the user needs to enter the new information on the update page and then submit it to the system (see Figures 26, 27, 28, and 29). For the current version of the ESSS, the user is able to perform personal information update, retirement fund reallocation, retirement investment direction change, and retirement contribution rate change. At the end of each submission, a message box will be popped up to inform the user whether or not the transaction is successful.



Enter your residential address below:

Address: 900 Ridgebury Road

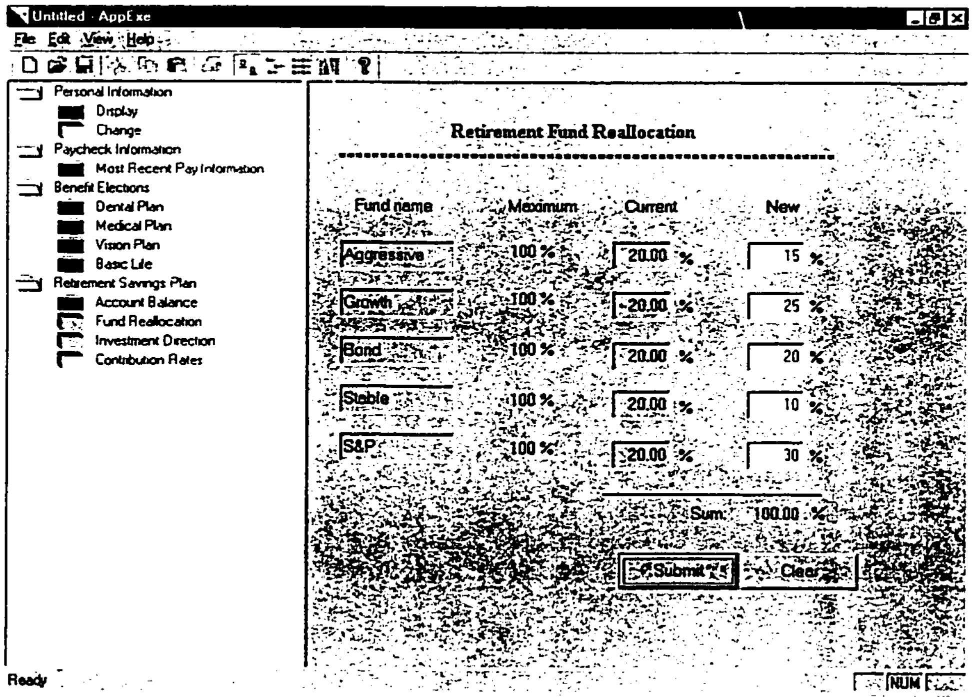
City: Ridgefield

Country: United States

State: Connecticut Postal code: 06877

Home telephone number: 203 1234567

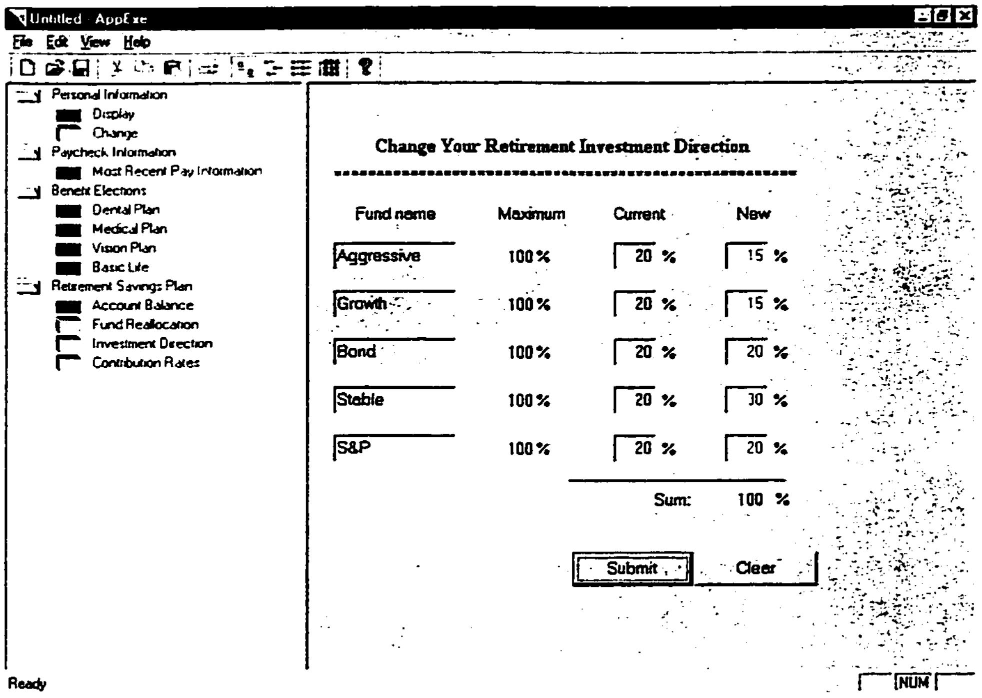
Figure 26. Personal Information Update



Retirement Fund Reallocation

Fund name	Maximum	Current	New
Aggressive	100 %	20.00 %	15 %
Growth	100 %	20.00 %	25 %
Bond	100 %	20.00 %	20 %
Stable	100 %	20.00 %	10 %
S&P	100 %	20.00 %	30 %
		Sum:	100.00 %

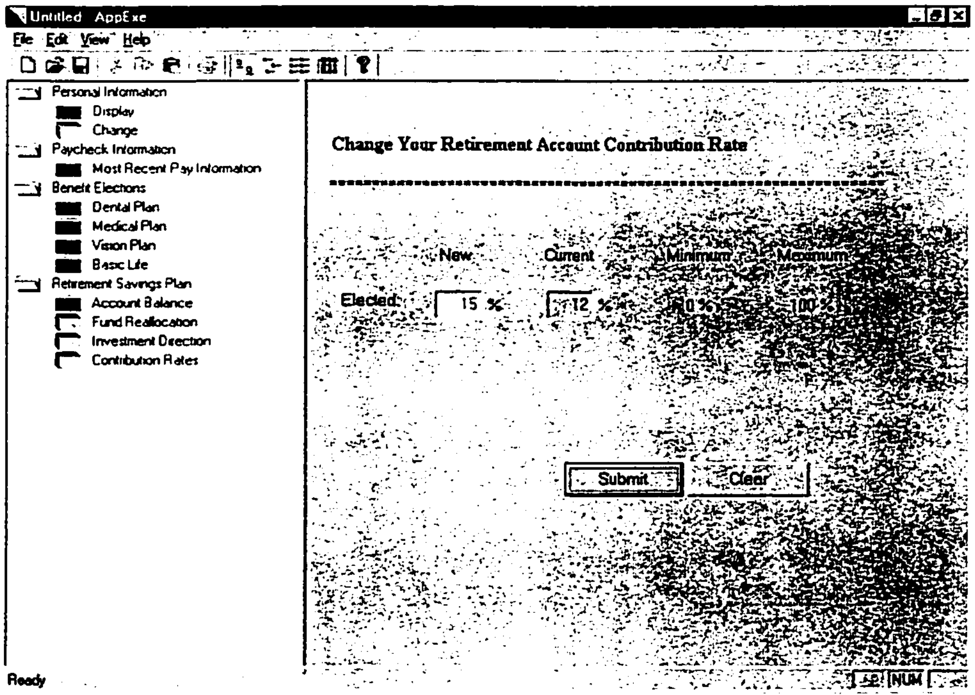
Figure 27. Retirement Fund Reallocation



Change Your Retirement Investment Direction

Fund name	Maximum	Current	New
Aggressive	100 %	20 %	15 %
Growth	100 %	20 %	15 %
Bond	100 %	20 %	20 %
Stable	100 %	20 %	30 %
S&P	100 %	20 %	20 %
		Sum:	100 %

Figure 28. Retirement Investment Direction Change



Change Your Retirement Account Contribution Rate

	New	Current	Minimum	Maximum
Elected:	15 %	12 %	0 %	100 %

Figure 29. Retirement Account Contribution Rate Change

7. Future Extension

For the current version of the ESSS, total 11 functions have been implemented, which are grouped into four basic categories — Personal Information, Paycheck Information, Benefit Elections, and Retirement Savings Plan. More functions or categories can be added for the future version of the ESSS:

- A time reporting category can be added, which may allow user to enter weekly time sheet and display past time sheet.
- A function to view previous pay information may need to be added to the paycheck information category.
- Functions that allow the user to change benefit election on an annual basis should be implemented in the future version.
- Function that allows the user to print out the requested information should also be considered in the future implementation.

8. Conclusions

Employee Self Service System (ESSS) was developed on Windows NT by using Microsoft Visual C++ and Microsoft Distributed Component Object Model (DCOM). The ESSS is a three-tier, distributed application capable of running on a number of different computers according to the configuration. It is designed and implemented to be a user-centered, employment-related self-service system. It is easy to learn and easy to use for all the users (employees). The primary goal has been achieved, which can be summarized as below:

- The ESSS is easy to learn. One basic window covers all the needs for information display and update.
- The ESSS is error-protected. Employees are allowed to make any possible operations without causing system failure.
- The ESSS is secure. The security for server component is set on the component basis with different security levels to control server access. This includes launch permission, access permission, and Windows NT permission.
- The ESSS is efficient. Users can make self-services without going through human resource individuals. The simple menu items listed on the left side window can help user quickly determine his needs.
- The ESSS is scalable. As the number of user increases, the system can be reconfigured to use more middle-tier servers to maintain the top system performance.
- The ESSS is fast. The implementation is done using object-oriented C++ language capable of generating efficient application program.

- The ESSS is reliable. Two or more middle-tier servers can be used to improve the system reliability. The failure of one server will not stop the whole system from functioning.
- The ESSS supports efficient data access. The frequently requested data is stored in the middle tier servers and the subsequent requests for the same data can be retrieved without accessing the database again.
- The ESSS supports component reuse. The ESSS can be re-built into Web-based application with the modification of only *GUI* module. All the other components can be reused.

From the experience of designing and implementing the ESSS, we are convinced that the three-tier distributed application is more efficient, more reliable, and more scalable than the conventional two-tier application. The Distributed Component Object Model provides an excellent middle-tier standard for implementing the distributed applications.

References

[1] D. Rogerson, Inside COM (Programming Series), Microsoft Press, 1997.

[2] R. Rosemary, DCOM Explained, Digital Press, 1998.

[3] J. Maloney, Distributed COM Application Development Using Visual C++ 6.0, UCI Press, 1999

[4] F. Bolton, Pure CORBA, Sams Press, 2001

[5] M. Henning and S. Vinoski, Advanced CORBA Programming with C++, Addison-Wesley Pub Co., 1999

[6] DCOM Technical Overview, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp

[7] Software Technology Review, http://www.sei.cmu.edu/str/descriptions/com.html

[8] Y.M. Wang, Y. Huang, and W.K. Fuchs, Progressive Retry for Software Error Recovery in Distributed Systems, IEEE Fault-Tolerance Computing Symp., pp. 138, 1993

[9] K.P. Birman, Building Secure and Reliable Network Applications, Manning Publications CO., 1996

[10] OSF DCE Specification, http://www.opengroup.org/pubs/catalog/f201.htm

Appendix — IDL Files

```
// DataControler.idl : IDL source for DataControler.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (DataControler.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";

typedef struct _EmpData {
	ULONG empID;
	BSTR empName;
	BSTR address;
	BSTR city;
	BSTR country;
	BSTR state;
	BSTR postcode;
	BSTR area;
	BSTR phone;
} EmpData;

typedef struct _PayCheckData {
	ULONG empID;
	BSTR name;
	BSTR payPeriod;
	float curPay;
	float curFedTax;
	float curStaTax;
	float curSecTax;
	float curMedTax;
	float curDenPreTax;
	float curMedPreTax;
	float cur401K;
	ULONG ytdNumPay;
} PayCheckData;


typedef struct _CoverData {
	ULONG empID;
	ULONG benType;
	BSTR coverType;
	ULONG cost;
} CoverData;

typedef struct _DepenData {
	BSTR name;
	BSTR relation;
} DepenData;

typedef struct _FundData {
	BSTR fundName;
	float amount;
```

```
        ULONG dirRate;
} FundData;

        [
                object,
                uuid(99015419-4D59-11D6-B6F3-0002A5633A88),
                helpstring("IDataServer Interface"),
                pointer_default(unique)
        ]
        interface IDataServer : IUnknown
        {
                [helpstring("method GetEmpInfo")]
                HRESULT GetEmpInfo([in] ULONG empId, [out] EmpData *empData);
                [helpstring("method GetEmpPayInfo")]
                HRESULT GetEmpPayInfo([in] ULONG empId, [out] PayCheckData *payData);
                [helpstring("method ModifyEmpInfo")]
                HRESULT ModifyEmpInfo([in] ULONG empId, [in] EmpData *empData);
                [helpstring("method GetNumBenInfo")]
                HRESULT GetNumBenInfo([in] ULONG empID, [out] ULONG *num);
                [helpstring("method GetNumDependent")]
                HRESULT GetNumDependent([in] ULONG empID, [out] ULONG *num);
                [helpstring("method GetEmpBenInfo")]
                HRESULT GetEmpBenInfo([in] ULONG empID, [in] ULONG numCover, [in] ULONG
numDepen, [out, size_is(numCover)] CoverData *pvecCoverData, [out, size_is(numDepen)] DepenData
*pvecDepenData);
                [helpstring("method GetNumFundInfo")]
                HRESULT GetNumFundInfo([in] ULONG empID, [out] ULONG *num);
                [helpstring("method GetEmpRetInfo")]
                HRESULT GetEmpRetInfo([in] ULONG empID, [in] ULONG numFund, [out] ULONG
*contriRate, [out, size_is(numFund)] FundData *pvecFundData);
                [helpstring("method ModifyContriRate")]
                HRESULT ModifyContriRate([in] ULONG empID, [in] ULONG contriRate);
                [helpstring("method ModifyInvRate")]
                HRESULT ModifyInvRate([in] ULONG empID, [in] ULONG numFund, [in] ULONG
contrirate, [in] ULONG rateType, [in] BSTR empName, [in, size_is(numFund)] FundData
*pvecFundData);
        };

[
        uuid(9901540D-4D59-11D6-B6F3-0002A5633A88),
        version(1.0),
        helpstring("DataControler 1.0 Type Library")
]
library DATACONTROLERLib
{
        importlib("stdole32.tlb");
        importlib("stdole2.tlb");

        [
                uuid(9901540A-4D59-11D6-B6F3-0002A5633A88),
                helpstring("DataServer Class")
        ]
        coclass DataServer
        {
                [default] interface IDataServer;
        };
```

```
};


// SecurityMgr.idl : IDL source for SecurityMgr.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (SecurityMgr.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
	[
		object,
		uuid(08E97B47-4CA4-11D6-B6F3-0002A5633A88),

		helpstring("ISecuritySvr Interface"),
		pointer_default(unique)
	]
	interface ISecuritySvr : IUnknown
	{
		[id(1), helpstring("method LoadEmpPassInfo")] HRESULT LoadEmpPassInfo();
		[id(2), helpstring("method GetPermission")] HRESULT GetPermission([in] ULONG
empId, [in] BSTR empPass, [out] ULONG* permit);
		[id(3), helpstring("method ChangePassword")] HRESULT ChangePassword([in]
ULONG empId, [in] BSTR oldPass, [in] BSTR newPass, [out] ULONG* isOK);
	};

[
	uuid(08E97B3B-4CA4-11D6-B6F3-0002A5633A88),
	version(1.0),
	helpstring("SecurityMgr 1.0 Type Library")
]
library SECURITYMGRLib
{
	importlib("stdole32.tlb");
	importlib("stdole2.tlb");

	[
		uuid(08E97B48-4CA4-11D6-B6F3-0002A5633A88),
		helpstring("SecuritySvr Class")
	]
	coclass SecuritySvr
	{
		[default] interface ISecuritySvr;
	};
};


// DataAccess.idl : IDL source for DataAccess.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (DataAccess.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";

typedef struct _EntryPassInfo {
            ULONG empId;
            BSTR empPass;
} EntryPassInfo;

typedef struct _Employee {
            ULONG empID;
            BSTR empName;
            BSTR address;
            BSTR city;
            BSTR country;
            BSTR state;
            BSTR postcode;
            BSTR area;
            BSTR phone;
} Employee;

typedef struct _EmpPayInfo {
            ULONG empID;
            ULONG payID;
} EmpPayInfo;

typedef struct _PayInfo {
            ULONG payID;
            BSTR payPeriod;
            float regularPay;
            float ret401KRate;
            ULONG ytdPayNum;
} PayInfo;

typedef struct _FundInfo {
            BSTR fundName;
            float amount;
            ULONG dirRate;
} FundInfo;

typedef struct _TaxInfo {
            ULONG payID;
            float fedTaxRate;
            float secTaxRate;
            float medTaxRate;
            float staTaxRate;
} TaxInfo;

typedef struct _DepInfo {
            BSTR depName;
            BSTR depRelation;
} DepInfo;

typedef struct _DeductInfo {
            ULONG payID;
            float medPreTax;
            float denPreTax;
            float visPreTax;
```

```
		float retPreTax;
} DeductInfo;

typedef struct _BenefitInfo {
		ULONG empID;
		BSTR benName;
		BSTR coverType;
		float cost;
} BenefitInfo;

typedef struct _RetFundInfo {
		BSTR fundName;
		float amount;
		ULONG dirRate;
} RetFundInfo;

		[
				object,
				uuid(383A4EF0-4BEC-11D6-B6F3-0002A5633A88),
				helpstring("IDataAccessSvr Interface"),
				pointer_default(unique)
		]
		interface IDataAccessSvr : IUnknown
		{
				[id(1), helpstring("method GetNumEmployee")]
				HRESULT GetNumEmployee([out] ULONG* pNumEmployee);
				[id(2), helpstring("method GetPasswordInfo")]
				HRESULT GetPasswordInfo([in] ULONG numEmployee, [out,
size_is(numEmployee)] EntryPassInfo* pvecPassInfo);
				[id(3), helpstring("method ChangePassword")]
				HRESULT ChangePassword([in] ULONG empId, [in] BSTR newPASS);
				[helpstring("method GetEmpInfo")]
				HRESULT GetEmpInfo([in] ULONG numEmp, [out, size_is(numEmp)] Employee
*pvecEmpInfo);
				[helpstring("method GetNumEmpPay")]
				HRESULT GetNumEmpPay([out] ULONG *num);
				[helpstring("method GetNumPayInfo")]
				HRESULT GetNumPayInfo([out] ULONG *num);
				[helpstring("method GetNumTaxInfo")]
				HRESULT GetNumTaxInfo([out] ULONG *num);
				[helpstring("method GetEmpPayRelation")]
				HRESULT GetEmpPayRelation([in] ULONG num, [out, size_is(num)]
EmpPayInfo *pvecEmpPay);
				[helpstring("method GetEmpPayInfo")]
				HRESULT GetEmpPayInfo([in] ULONG num, [out, size_is(num)] PayInfo
*pvecPayInfo);
				[helpstring("method GetEmpTaxInfo")]
				HRESULT GetEmpTaxInfo([in] ULONG num, [out, size_is(num)] TaxInfo
*pvecTaxInfo);
				[helpstring("method GetNumDeductInfo")]
				HRESULT GetNumDeductInfo([out] ULONG *num);
				[helpstring("method GetEmpDeductInfo")]
				HRESULT GetEmpDeductInfo([in] ULONG num, [out, size_is(num)] DeductInfo
*pvecDeductInfo);
				[helpstring("method ModifyEmpInfo")]
				HRESULT ModifyEmpInfo([in] ULONG empId, [in] Employee *pEmp);
```

```
                    [helpstring("method GetNumBenInfo")]
                    HRESULT GetNumBenInfo([in] ULONG empID, [out] ULONG *num);
                    [helpstring("method GetNumDependent")]
                    HRESULT GetNumDependent([in] ULONG empID, [out] ULONG *num);
                    [helpstring("method GetEmpBenInfo")]
                    HRESULT GetEmpBenInfo([in] ULONG empID, [in] ULONG numBen, [in]
ULONG numDep, [out, size_is(numBen)] BenefitInfo *pvecBenInfo, [out, size_is(numDep)] DepInfo
*pvecDepInfo);
                    [helpstring("method GetNumFundInfo")]
                    HRESULT GetNumFundInfo([in] ULONG empID, [out] ULONG *num);
                    [helpstring("method GetEmpRetInfo")]
                    HRESULT GetEmpRetInfo([in] ULONG empID, [in] ULONG numFund, [out]
ULONG *contriRate, [out, size_is(numFund)] RetFundInfo *pvecRetFundInfo);
                    [helpstring("method ModifyContriRate")]
                    HRESULT ModifyContriRate([in] ULONG empID, [in] ULONG contriRate);
                    [helpstring("method ModifyInvRate")]
                    HRESULT ModifyInvRate([in] ULONG empID, [in] ULONG numFund, [in]
ULONG contriRate, [in] ULONG rateType, [in] BSTR empName, [in, size_is(numFund)] RetFundInfo
*pvecRetFundInfo);

          };

[
          uuid(383A4EE3-4BEC-11D6-B6F3-0002A5633A88),
          version(1.0),
          helpstring("DataAccess 1.0 Type Library")
]
library DATAACCESSLib
{
          importlib("stdole32.tlb");
          importlib("stdole2.tlb");

          [
                    uuid(2C0A2146-4BF7-11D6-B6F3-0002A5633A88),
                    helpstring("DataAccessSvr Class")
          ]
          coclass DataAccessSvr
          {
                    [default] interface IDataAccessSvr;
          };
};
```