

MIDDLEWARE DATABASE DRIVERS

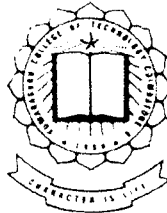
P-483

A Project Report

SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING IN
COMPUTER SCIENCE AND ENGINEERING
OF BHARATHIAR UNIVERSITY

BY
P.S. JAYA SHANKARI
Reg. No. 9937K0004

GUIDED BY
Mrs. S.DEVAKI



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
KUMARAGURU COLLEGE OF ENGINEERING
COIMBATORE - 641 006.

2000 - 2001



Department of Computer Science and Engineering
Kumaraguru College of Technology
(Affiliated to the Bharathiar University)
Coimbatore – 641 006

Project Work

CERTIFICATE

Bonafide record of the project work

Middleware Database Drivers

Done by

P.S.Jaya Shankari
(Reg. No. 9937K0004)

Submitted in partial fulfillment of the requirements
For the degree of Master of Engineering
in Computer Science and Engineering
Of the Bharathiar University



S. Srinani
Faculty Guide

S. Jayasingh 11/1/01
Head of the Department

Submitted for Viva-Voce Examination held on 20-1-2001

S. Srinani
Internal Examiner

M. Srinani 20/1/2001
External Examiner



SANPO TECHNOLOGIES

6/42A, Kongu Nagar Entrance,
Opp. Alvernia School, Trichy Road,
Ramanthapuram, Coimbatore - 641 045.
Phone : 317738 / 574314
Mobile : 98422-92292
Email : sanpo@satyam.net.in

CERTIFICATE

This is to certify that Miss. P.S. JAYA SHANKARI, final year M.E student of KUMARAGURU COLLEGE OF TECHNOLOGY, Coimbatore, has completed the project work entitled "MIDDLEWARE DATABASE DRIVERS" during the period 15th July 2000 to 12th December 2000 at our concern.

During her association with Sanpo Technologies, she is found to be sincere in her assignments. She has developed the program and installed "MIDDLEWARE DATABASE DRIVERS" upto our satisfaction.

For Sanpo Technologies,


(P.PONNUSWAMY)
Director

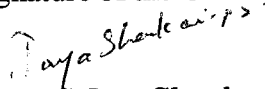


Declaration

I, P.S.Jaya Shankari hereby declare that this project work entitled "MIDDLEWARE DATABASE DRIVERS" submitted to Kumaraguru College of Technology, Coimbatore (Affiliated to Bharathiar University) is a record of original work done by me under the supervision and guidance of Mrs. S.Devaki M.S., Department of Computer Science and Engineering.

Name of the Candidate
P.S.Jaya Shankari

Register Number
9937K0004

Signature of the Candidate

(P.S.Jaya Shankari)

Countersigned by:

Staff in Charge



Mrs. S.Devaki M.S.

Assistant professor

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore – 641 006

Place : Coimbatore

Date :

Acknowledgement

I am greatly indebted to **Dr.K.K.Padmanabhan Ph.D.**, principal, KCT, for having provided all the facilities for carrying out the project.

I earnestly express my sincere thanks to **Dr.S.Thangasamy**, Head of the department, Department Of ComputerScience for encouraging me through the project

I owe a great debt to my internal guide **Mrs.S.Devaki M.S.**, Asst.Proffesor, Department Of ComputerScience for imparting an invaluable guidance, steadfast support and impeccable tutelage for the successful completion of the project.

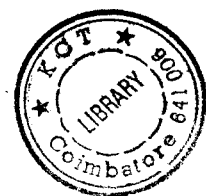
I am thankful to **Mr.T.Babu Kumar B.E.**, Software Engineer, Sanpo Technologies, for his valuable guidance and suggestions throught the project.

I express my greatfulness to **Mr.P.Ponnuswamy**, Managing Director, Sanpo Technologies, coimbatore, for giving me an oppurtunity to undertake my project in this organisation and giving all infrastructural facilities for carrying out this project.

Iam very much obliged to express my sincere thanks to **Mr.R.Kannan M.E.** , Asst.Professor, Department Of ComputerScience for his suggestions in the project.

Finally, I acknowledge my immence gratitude to all the staff members of the Department Of Computerscience and Engineering, KCT, for their encouragement and moral support in the successful completion of the project.

Synopsis



Synopsis

When Microsoft Corp. came along and started pushing Open Database Connectivity(ODBC) as the answer to simplified data access from any datasource and in any place, it worked! Developers had a standard Application Programming Interface(API) that they could depend on for virtually any platform and development tool. This solved the developer's problem. For the Network manager, however, ODBC magnifies the problem of software distribution. To access these datasources, a driver is needed for every single datasource on every single client. Suppose if there are 1000 clients and if an additional DataBase Management System(DBMS) is just installed then this requires installation of driver for that datasource on all 1000 client machines.

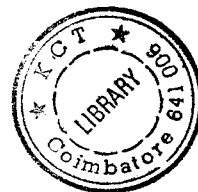
To alleviate this problem of Network Manager, this project has been developed in which only a single driver for each datasource is required irrespective of any number of client machines. It comprises of a server program and client program and a programmatical connection between them. The server program can handle as many client connections as needed and servicing is done as and when connections are needed.

Contents

Acknowledgement

Synopsis

1. Introduction	1
2. System study and problem formulation	3
2.1 Survey of the existing system	3
2.2 Limitations of the existing system	4
2.3 Objectives of the proposed system	4
2.4 Proposed system overview	5
3. Overview of ODBC and JDBC	8
3.1 ODBC	8
3.2 JDBC	9
4. Overview of the concepts involved in the project	12
4.1 Three Tier client/server,object style	12
4.1.1 Advantages of 3-Tier Architecture	13
4.2 CORBA Overview	14
4.3 CORBA Architecture	16
4.4 CORBA static method invocation	18
4.5 Benefits of CORBA ORB	20
5. Hardware and software Environment	22
6. Implementation of the project	23
6.1 Class diagram.	23
6.2 RecordSetI Interface	24
6.3 DBManager Interface	25
6.4 DBMServant Class	29
6.5 RecordSet Class	30
6.6 DBMServer Class	30
6.7 DBMClient Class	32



7. VC++ Connectivity	35
7.1 Introduction.	35
7.2 The IDL	35
7.3 VC++ Implementation	37
7.4 VC++ Client	37
7.5 VC++ Server	37
7.6 Running the client/server application	38
8. Connection Monitoring	
8.1 Illustration of Connection Monitoring	39
9. System Testing	41
9.1 Employee Management System	41
10. Conclusion	45

BIBLIOGRAPHY

APPENDICES

Appendix-A Sample Source Code

Appendix-B Sample Screens

1. Introduction

Middleware Database Drivers is mainly developed to maintain the transaction between various types of GUI tools and various types of databases in a more convenient manner than the existing one from the network administrator's point of view. The network administrator is freed from the maintenance of the database drivers on all client machines, since the client machines in this system does not require any drivers at all. All the database drivers are maintained centrally at the middleware.

This system helps the network administrator to maintain the database drivers in a single system. The system also allows the fresh programmers to manipulate with the database without even the knowledge of database programming in the programming language.

The project is based on three modules or tiers

1. Client tier
2. ODBC server tier
3. Database tier

Each module starts with a discussion of terminologies, concepts and mechanisms, followed by the software implementation. The connection program is written using CORBA3.1. An Employee Management System application is also being developed to illustrate the communication of the client and server. The client program is implemented using java1.1.5. The



same application is developed using VC++, to illustrate the reusability of code in CORBA

The system also consists of a connection monitoring module which allows the network administrator to view what all clients are connected to the database presently and also during a particular time interval.

2. System Study and Problem Formulation

2.1 Survey of the Existing System

In an existing client/server system, ODBC driver along with the database specific drivers are installed in all client machines.

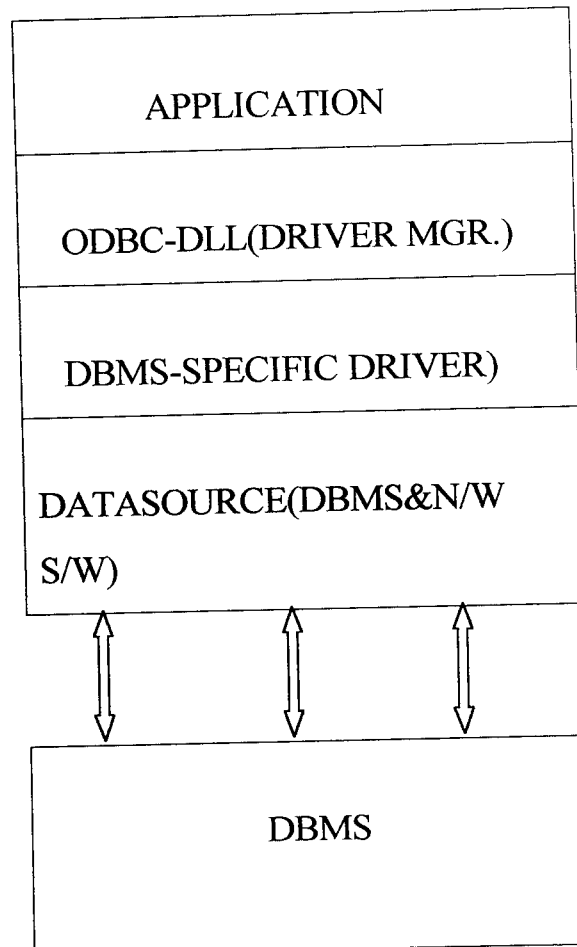


Figure 2.1 ODBC Architecture

ODBC is a layered architecture. The client side layers provide the API to the application as well as define datasources to their use. Driver Manager is the visible part of ODBC, where drivers are added and datasources configured on each client.

Below the Driver Manager are the ODBC drivers themselves. These are specific components that interface with individual datasources. These drives contain the logic to interface over the network using the database server specific middleware. Ofcourse, each database vendor's middleware product(SQL *Net, DB-Library,etc) must also be installed on client machines. This whole driver portion - from the ODBC driver through the network layers to the datasource itself - is the most complex part of the ODBC architecture.

2.2 Limitations of the existing System

1. Since the driver must be present on all client machines, once the administrator has installed the database vendor's software on the server computer, he or she then visits each client computer to install that specific driver on all client machines. Thereafter each time a network loses or gains a client machine, the administrator performs maintenance tasks to install or remove driver software

2. The database access middleware running at each client computer must be configured into the protocol stack by an administrator, and typically all clients should run the same version of the middleware.

2.3 Objectives of the Proposed System

The main objective of this project is to make database driver management easier by having only one driver for each datasource present in the network, rather than having all vendor specific database drivers on all client computers.

2.4 Proposed System Overview

The proposed system consists of a middleware ODBC server. On the ODBC server a driver is installed for every datasource. This requires installing no drivers on the client computer.

The Usecase diagram is shown in figure 2.4.1

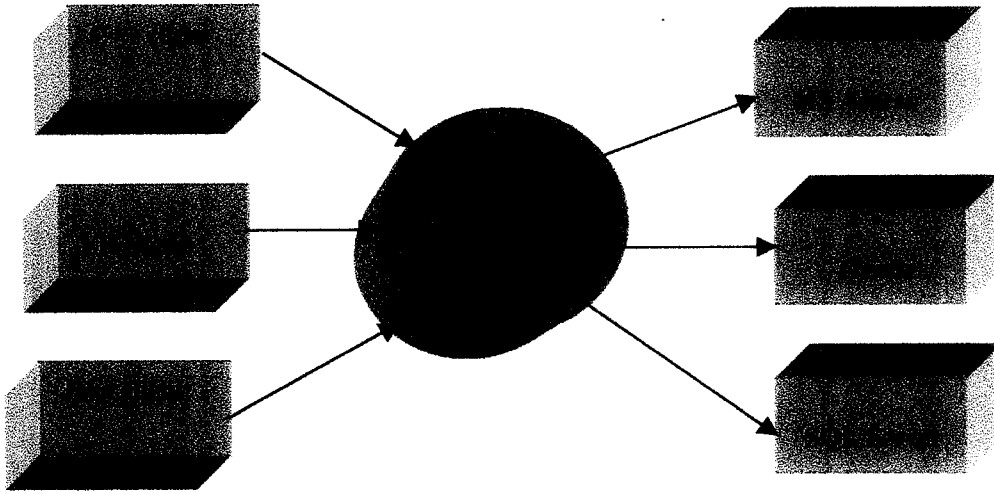
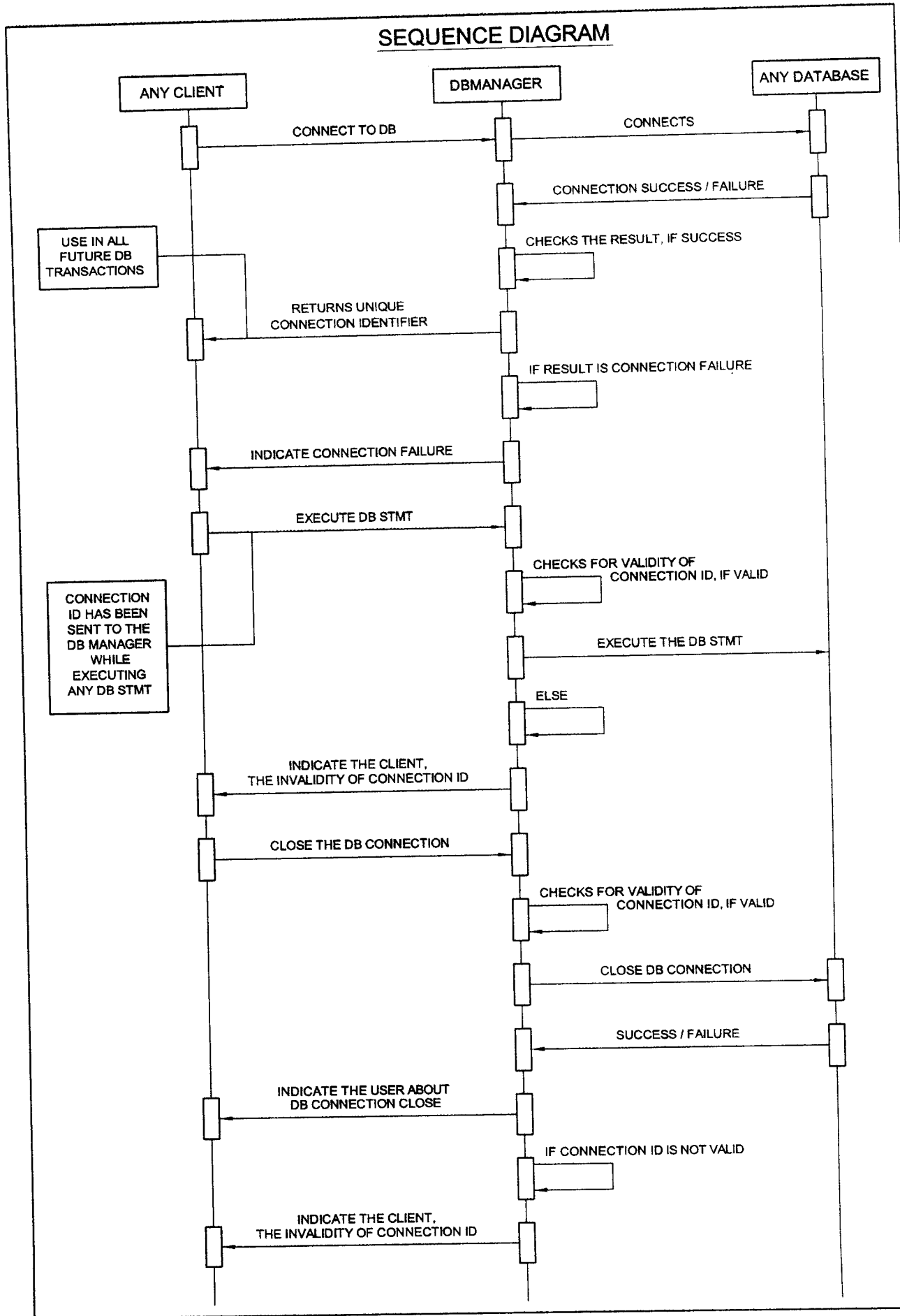


Figure 2.4.1 Usecase diagram

In the above Usecase Diagram, C++ client, Java client and VB client are connected to the backends namely Oracle, SQL Server and MS Access via the DBManager. Here DBManager is the middleware ODBC server which consists of the required database drivers. The client machines here do not require any drivers to be installed on them.

The sequence diagram shown in figure 2.4.2 illustrates the sequence in which the connections are established and exited.

SEQUENCE DIAGRAM



The above sequence diagram shows how a client establishes a connection with the DBManager and acquires the respective database.

The client when required to connect to any database, it first gets connected to the DBManager. The DBManager in turn establishes the connection with the required database and checks if the connection with the database is successfully accomplished. If success, DBManager returns a unique connection identifier for each connection established. This connection identifier is used in all future database transactions. If failure, DBManager indicates the connection failure to the client.

During execution of the database statement, connection identifier is sent to the DBManager. The DBManager checks for the validity of the connection identifier. If the connection identifier is valid, it executes the statement on connecting database, If the connection identifier is invalid, it indicates the client regarding the invalidity of the connection identifier.

To close with the database connection, the DBManager checks for the validity of the connection identifier. If valid, it ends the database connection and indicates the client about the close of the connection. Else it indicates the invalidity of the connection identifier to the client.

3.1 ODBC

ODBC is a multidatabase API for programs that uses SQL statements to access data. An ODBC based program can access heterogenous databases without needing source code changes.

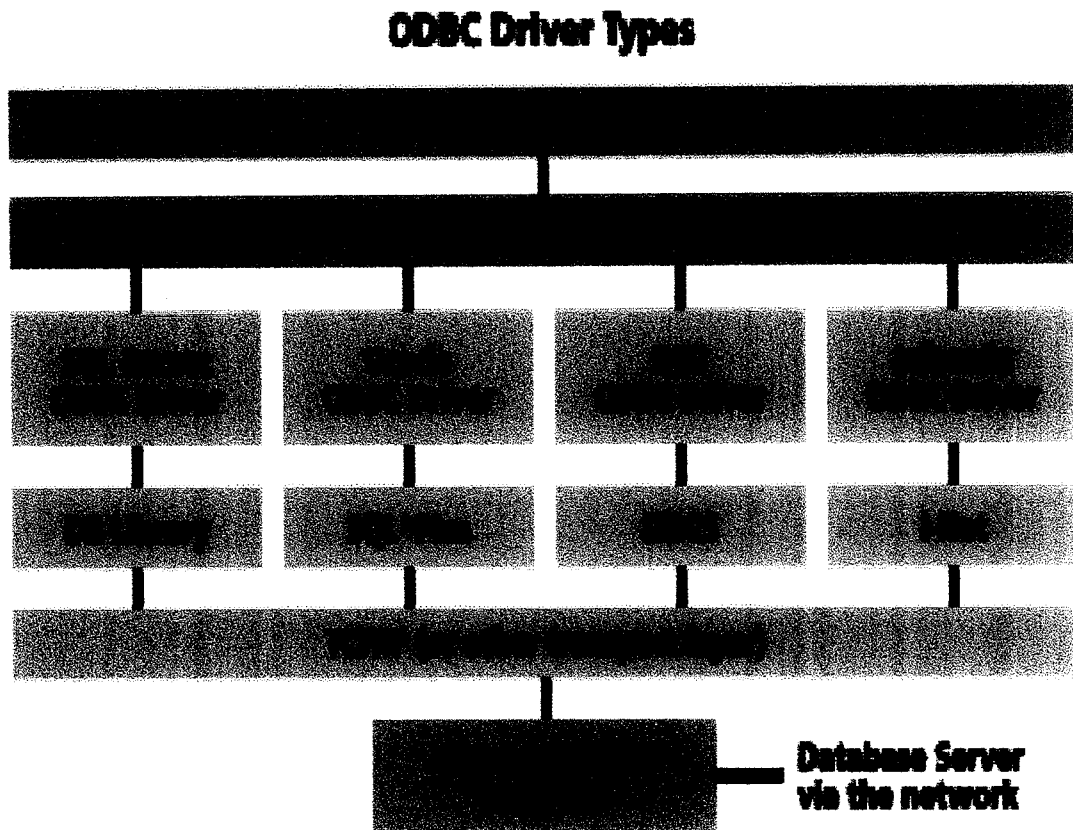


Figure 3.1 ODBC Driver types

The ODBC driver types are shown in Figure 3.1. ODBC defines the client side of the database connectivity but not the server side. ODBC middleware drivers typically rely on the underlying presence of a vendor's proprietary driver (SQL *Net, in the case of Oracle). ODBC drivers transform ODBC calls into vendor specific access requests and responses. As a result, network administrators must install and configure not only an ODBC driver

on each client, but the underlying vendor specific proprietary driver. (Referred from URL [1] in Bibliography)

ODBC's addition of an extra layer of insulating middleware is both its strength and its weakness. ODBC presents a common standard interface no matter what the vendor specific middleware might look like, but ODBC consumes some memory, and in its earliest incarnations slowed data access noticeably.

3.2 JDBC

JDBC is a collection of database access middleware drivers that provide Java programs with a call-level SQL API. Java applications use the Java API to connect to databases, store and retrieve database content, thus making JDBC a Java enabled delivery mechanism for SQL. JDBC is to Java programs what ODBC is to programs written in languages other than Java. JDBC includes four types of drivers. Figure 3.2 illustrates the JDBC driver types. (Referred from Book [1] in Bibliography). The JDBC driver types are as follows

Type1 Driver

These drivers use a bridging technology to access a database. It is a JDBC-ODBC bridge that provides JDBC connectivity via ODBC drivers. It provides a gateway to the ODBC API. Implementations of that API in turn do the actual database access.

Type2 Driver

These drivers are native API drivers. The driver contains Java code that calls native C or C++ methods provided by the individual database

JDBC Driver Types

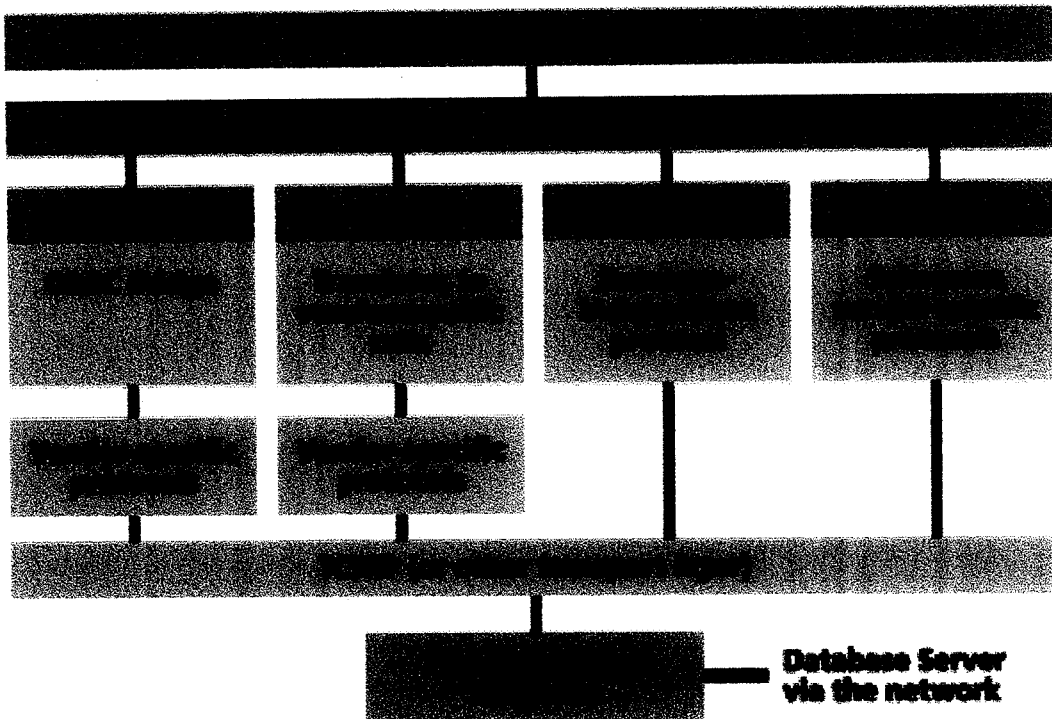


Figure 3.2 JDBC driver types

vendors that perform the database access. i.e., Type2 JDBC driver is typically a direct bridge to the proprietary call-level API of the database product. It does not require the presence of ODBC.

Type3 Driver

These drivers provide a client with the generic network API that is then translated into database specific access at the server level. In otherwords, the JDBC driver on the client uses sockets to call a middleware application on the server that transfers client requests into an API specific to the desired driver. Type3 drivers are highly generic—it will run on any Java enabled platform with the TCP/IP connection to the database server.



Type4 Driver

Using network protocols built into the database engine, Type4 JDBC drivers talk directly to the database using Java sockets. This is the most direct pure Java solution. In otherwords, Type4 Java drivers converts JDBC requests into a database vendor's database product.

4. Overview of the concepts involved in the project

4.1 3-Tier client/server Architecture, object style

This project involves 3-tier architecture where the first tier is the client, second tier is the ODBC server and the third tier is the database server. The key characteristic of a 3-Tier client-server architecture is the separation of a distributed computing environment into presentation, functionality, and data components, such that there is a well-defined interface between each component, and the software used to implement each component can be replaced easily. The 3-tier client/server architecture in an object style is as shown in figure 4.1

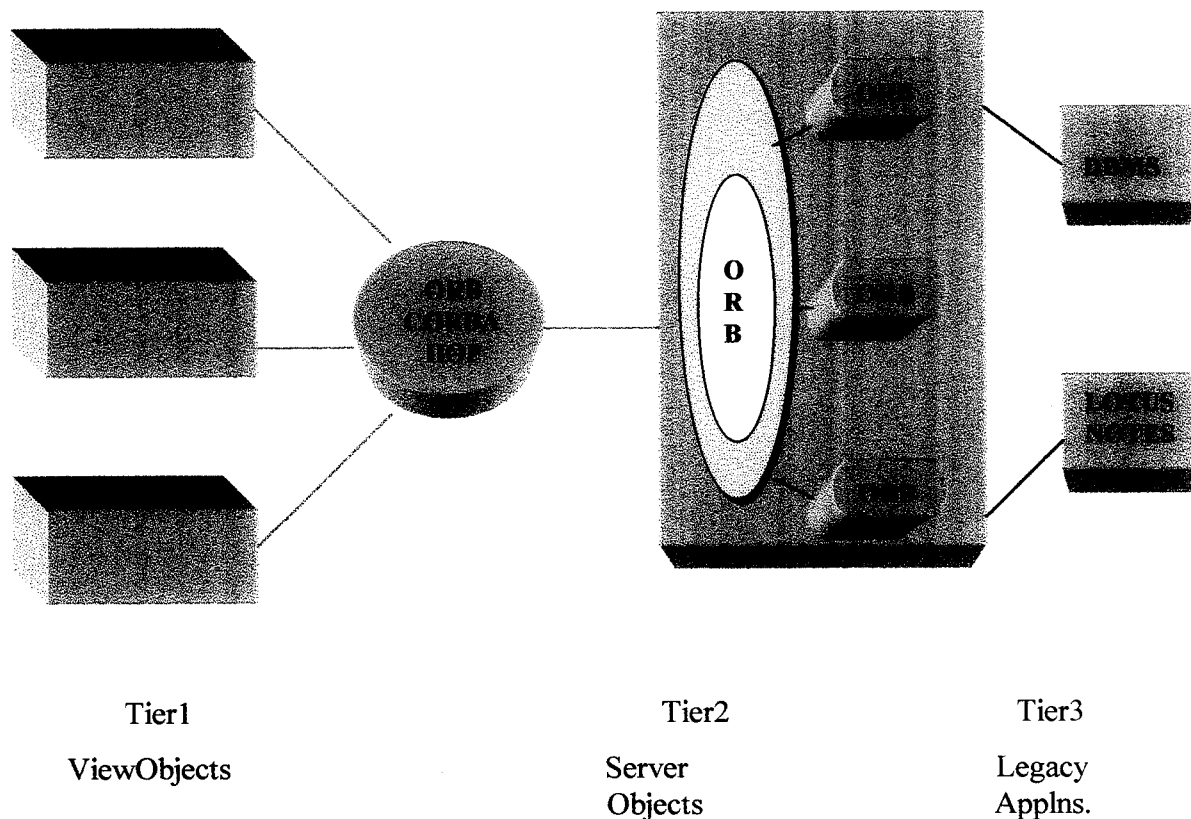


Figure 4.1 3-Tier Client/Server, Object Style

The first tier represents the visual aspects of the business object-one or more visual objects may each provide a different view. These visual objects typically live on the client.In the middle tier are server objects that represent the persistent data and the business logic functions. In the third tier are existing databases and legacy applications.

Middle-tier server objects interact with their clients and implement the logic of the business object. They can extract their persistent state from multiple data sources.The server object provides an integrated model of the disparate data sources and backend applications.

The clients typically interact with the middle-tier server objects via an ORB. Clients must never interact directly with third-tier data sources. These sources must be totally encapsulated and abstracted by the middle-tier server objects. In addition, middle-tier objects can communicate with each other via a server ORB that they can use to balance loads, orchestrate distributed transactions, and exchange business events. This makes ORB based business objects very scalable. Finally, the server objects communicate with the third tier using traditional middleware.(Referred from Book [2] in bibliography)

4.1.1 Advantages Of 3-Tier Architecture

Isolation of concerns

The major advantage is that the front end clients are clearly separated from the back end data manipulation facilities. This allows details of the data storage mechanisms such as which database is used, record structure and field names to be abstracted away from the client processes. All the front end sees is an abstract operation request which takes input and output parameters.

Enabling database migration

Database restructuring, upgrades, migration or other changes can be performed without the necessity to stop or alter the client programs.

Front end modifications

Similarly, new front end clients can be introduced or old ones removed without any need to modify the databases or provide new access mechanisms.

Data from multiple sources

A client may require data from a number of servers. This can be handled easily because a CORBA server can communicate with multiple servers at a time.

Reduced connections

With M client programs and N data servers direct connection requires $M \times N$ connections; use of a 3-Tier architecture reduces this to $M + N$ connections.

Reduced database loading

In a Three Tier Architecture, not only does the database machine benefit from fewer connections but any data caching operations result in fewer data operations and therefore less throughput. In addition, this saving is concentrated on those very queries that are most commonly performed, thus reducing the potential for conflict on any hot-spots in the data.(Referred from URL [2] in Bibliography).

4.2 CORBA Overview

In a normal object oriented program written in C++ or Java, the program contains all of the classes that the application needs. The compiler

compiles and links these objects. When these objects are instantiated with the **new** statement, they are created in the application's memory space and methods execute as part of a single process.

CORBA is a technology that allows a client application to call objects that reside on a server. The server may be running on the same machine or on a machine 3,000 miles away. At its most basic level, CORBA is extremely simple - instead of instantiating an object in our process's memory space, we instantiate the object on a server machine somewhere on the network. The calls to the object's methods and the parameters passed to the methods are then packaged in the form of a network packet and sent to the server. The function actually runs on the server, and any results are sent back to the caller through the network.

The advantage of this approach is that the server can act as a high-powered, central shared resource. The server can have databases or other local services that the server-based CORBA objects can access. The disadvantage, of course, is speed. The number of function calls you can make per second is severely limited by the speed of the network. That holds true for any technology that uses a network connection for data transport, so CORBA is not unique. DCOM and even sockets suffer from the same delays.

One thing unique to the CORBA approach is a facility called the ORB (Object Request Broker). The ORB runs as part of the client and server processes and handles the network connection between the two. In this project VisiBroker ORB is used. (Referred from URL [3] in Bibliography). With VisiBroker, the client and server machines also use a service called the OSAGENT. The OSAGENT is contacted by a client ORB at a standard port number (generally 14000). The OSAGENT acts as a directory that helps a client ORB to find an object that it is looking for.

4.3 CORBA Architecture

The architecture of CORBA is shown in figure 4.3

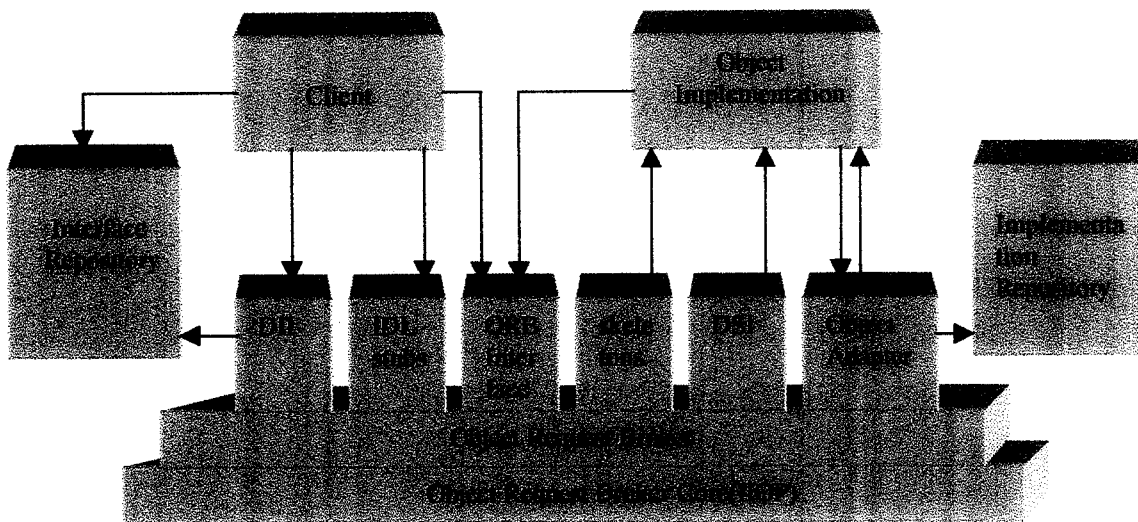


Figure 4.3 Structure of CORBA ORB

The Client IDL stubs

The Client IDL stubs provide static interfaces to object services. These precompiled stubs define how clients invoke corresponding services on the servers. A client must have an IDL stub for each interface it uses on the server. The stubs include code to perform marshaling.

The Dynamic Invocation Interface(DII)

DII lets us discover methods to be invoked at runtime. CORBA defines standard APIs for looking up the metadata that defines the server interface, generating the parameters, issuing the remote call, and getting back the results.

The Interface Repository APIs

The Interface Repository is a run-time distributed database that contains machine-readable versions of the IDL-defined interfaces.

The Object Request Broker(ORB)

The ORB is the central component in CORBA. It provides a mechanism for transparently communicating client requests to target object implementations. It simplifies distributed programming by decoupling the client from the details of the method invocations, and hence makes client requests appear to be local procedure calls.

The Object Interface

The Object Interface consists of few APIs to local services that may be of interest to an application.

The Server IDL stubs(skeletons)

The server IDL stubs provide static interfaces to each exported by the server.

The Dynamic skeleton Interface(DSI)

DSI provides a run-time binding mechanism for servers that need to handle incoming method calls for components that do not have IDL based compiled skeletons. The Dynamic skeleton looks at the parameter values in an incoming message to figure out who it's for--that is, the target object and method.

The Object Adapter

The Object Adapter sits on the top of the ORB's core communication services and obtains requests for services on behalf of the server objects. It provides the run-time environment for instantiating server objects, passing requests to them, and assigning them Object IDs. CORBA calls the IDs Object References. The OA also registers the classes it supports and their run-time instances(i.e.,objects) with the Implementation Repository.

The Implementation Repository

The Implementation Repository provides a run-time repository of information about the classes a server supports, the objects that are instantiated, and their IDs. (Referred from Book [2] in Bibliography).

4.4 CORBA static method invocation

Since static method invocation is involved in the project, the following discussion gives a brief overview of the static method invocation in CORBA.

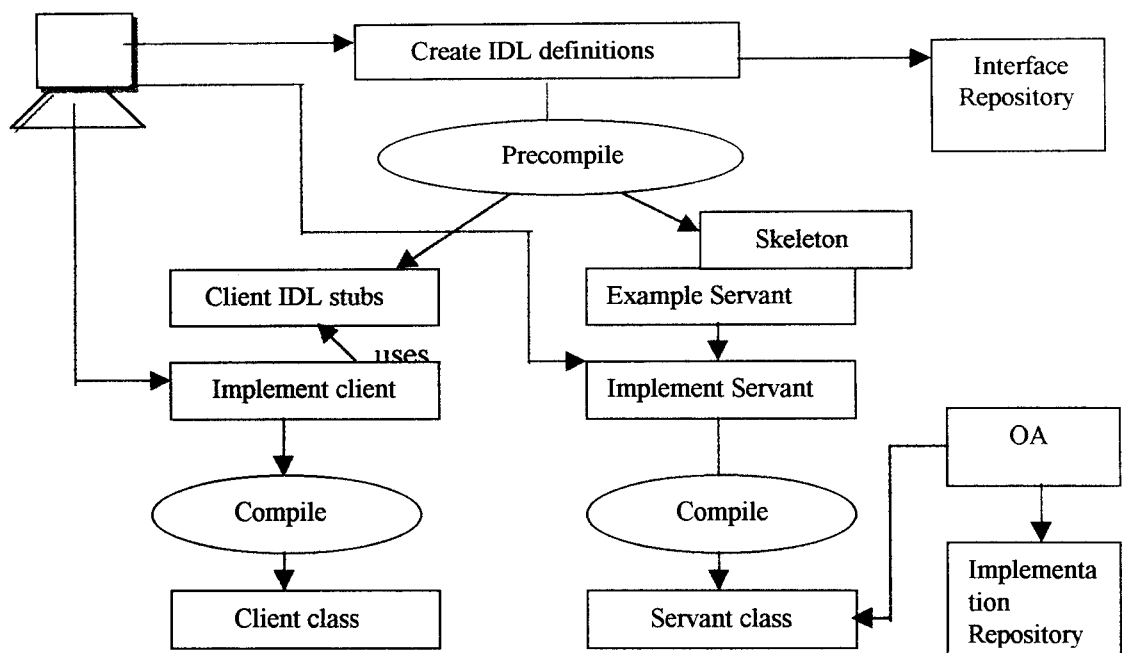


Figure 4.4 Static method invocation

Figure 4.4 shows the steps we go through to create server classes, provide interface stubs for them, store their definitions in the Interface repository, instantiating the objects at run-time, and record their presence with the Implementation Repository. The steps involved are as follows

Define Server interfaces using Interface Definition Language

The IDL is the means by which objects tell their potential clients what operations are available and how they should be invoked. The IDL definition

language defines the type of objects, their attributes, the methods they export, and the method parameters.

Bind the interface definitions to the Interface Repository

The IDL information is loaded in an Interface Repository that programs can access at run-time.

Run the IDL file through the language compiler

A typical CORBA precompiler is capable of generating atleast three types of output files:

1. import files that describe the objects to an Interface Repository
2. client stubs for the IDL defined methods-- these stubs are invoked by a client program that needs to statically access the IDL-defined services via ORB.
3. server skeletons that call the methods on the server. They are also called up-call interfaces.

Add the implementation code

The code that implements the methods in the skeleton must be supplied. In otherwords server classes must be created.

Compile the code

The code is compiled using a regular language compiler.

Register the run-time objects with the Implementation Repository

The Implementation Repository must know which classes are supported on a particular server. The ORB uses this information to locate active objects or to request the activation of objects on a particular server.

Instantiate the objects on the server

At startup time, a server Object Adapter may instantiate on demand server objects that service remote client method invocations. These run-time objects are instances of our implementation classes.

Implement the client code

The client invokes CORBA objects using language-specific bindings. These stubs take care of all marshaling and unmarshaling.

Compile the client code

This is done using a regular language compiler.

4.5 Benefits of using CORBA ORB

CORBA is the best client/server middleware ever defined. To illustrate why CORBA ORBs make such great client/server middleware, the short list of benefits that every CORBA ORB provides is figured out.(Referred from Book [2] in Bibliography).

Static and dynamic method invocations

A CORBA ORB lets us to either statically define our method invocations at compile time, or it lets us dynamically discover them at run-time.

High-level language bindings

A CORBA ORB lets us invoke methods on server objects using our high-level language of choice. It doesn't matter in what language server objects are written in. CORBA separates interface from implementation and provides language-neutral data types that makes it possible to call objects across language and operating system boundaries.

Self-describing system

Every CORBA ORB must support an Interface Repository that contains real-time information describing the functions a server provides and their parameters. The clients use metadata to discover how to invoke services at run time. No other form of client/server middleware provides this type of run-time metadata and language independent definitions of all its services.

Local/Remote Transparency

An ORB can run in a standalone mode on a laptop, or it can be interconnected to every other ORB in the universe using Internet Inter ORB Protocols(IIOP). An ORB can broker inter-object calls within a single process, multiple processes running within the same machine, or multiple processes running across networks and operating system.

Built-in security and transactions

The ORB includes context information in its messages to handle security and transactions across machine and ORB boundaries.

Polymorphic messaging

In contrast to other forms of middleware, an ORB does not simply invoke a remote function—it invokes a function on a target object.

5. Hardware and Software Environment

The hardware environment under which the system is developed is as

follows:

- A LAN with atleast two PentiumII machines
- 64 MB RAM
- 4.3 GB HDD

The software requirements of the system are as follows

- Windows98 Operating system
- JDK 1.2
- VC++ 6.0
- Visigenic Vbroker ORB for Java and CORBA3.1
- TCP/IP installed.
- MS Access

6. Implementation Of the Project

6.1 Class Diagram

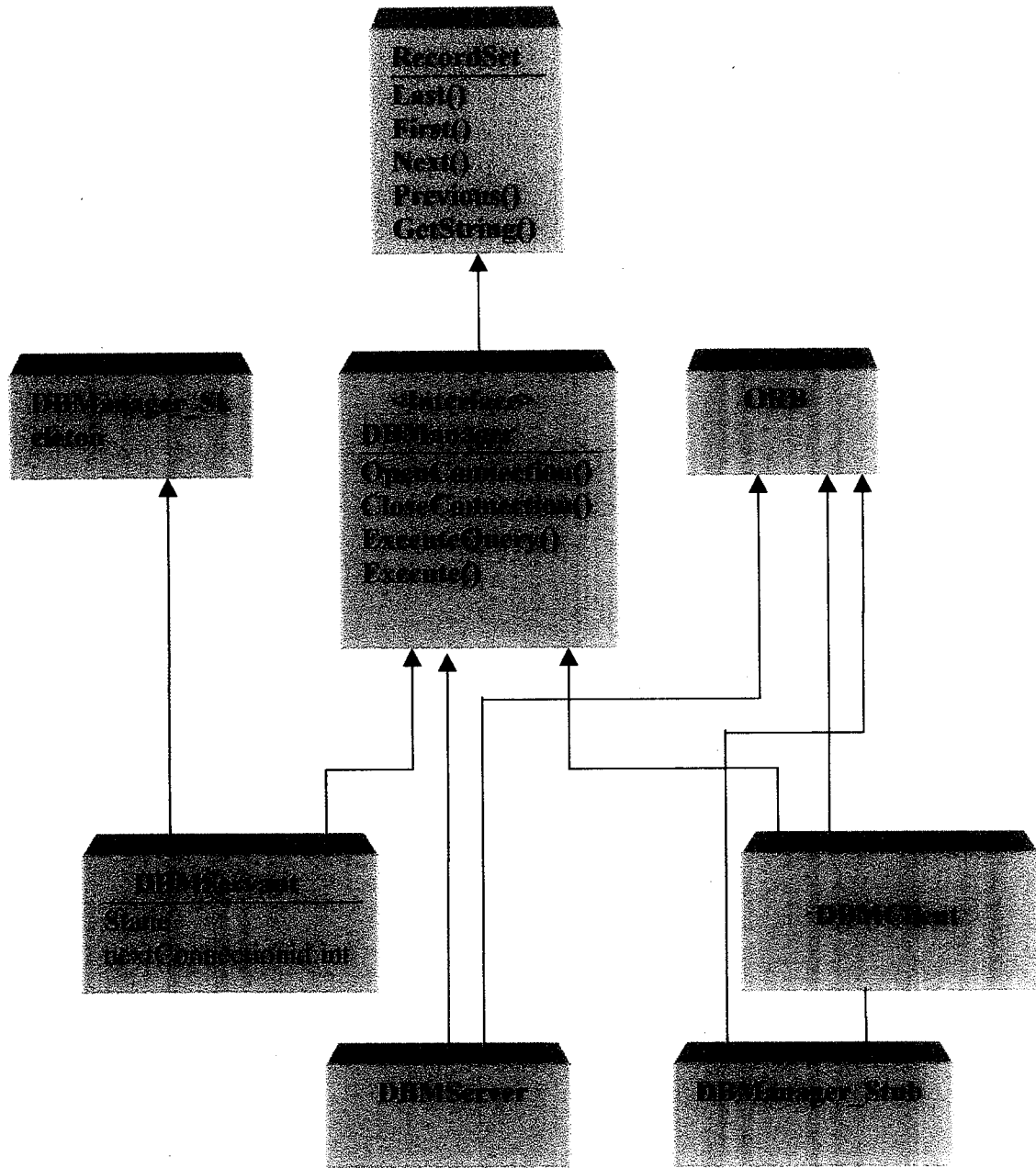


Figure 6.1 Class diagram

The Class diagram is shown in figure 6.1. The implementation of the project is as follows

The system is divided into three parts

- 1.DBMServant
- 2.DBMServer
- 3.The Client

The DBMServant is the actual driver. All the database drivers are present here.

DBMServer is the proxy for the DBMServant. It creates an object for the DBMServant class and registers with the ORB which allows the client to access the driver.

The Client is any client that supports ODBC or JDBC. They can manipulate with the database using the DBMServant class.

The detailed description of the implementation of the project is presented below.

The first step in the development process is to create language independent IDL definitions of the server's interfaces. Here the IDL definition consists of two interfaces namely RecordSetI and DBManager. The dbm.idl file contains the IDL for both the interfaces. The interfaces are defined inside the module "dbm". A module is the CORBA equivalent of the java package or a C++ namespace. It provides a naming context for a set of related interfaces. Names must be unique within a module's scoping context. The two interfaces in the dbm module are as follows

6.2 RecordSetI Interface

The Interface RecordSetI consists of five methods as follows

- 1.last()
- 2.first()
- 3.next()

4.previous()

5.getString()

The methods last(),first(),previous() and next() are written so as to fetch the last, first, next and previous records from the given table in a particular database.

The method getString() is written so as to fetch the item from the table given the column number.

The specification of the methods are only allowed in the interface. All the implementations of the methods in this interface are provided in the class that implements this interface.

6.3 DBManager Interface

The Interface DBManager acts as the middleware. It consists of four methods as follows

1. Long openConnection(String dsn, String user, String pwd);
2. Boolean closeConnection(long connectionId);
3. RecordSetI createRecordSet(long connectionId, String query);
4. Boolean execute(long connectionId, String stmt);

The openConnection method opens a connection with the particular database given the dsn,username and password. On success, it returns a unique connectionId for each connection established. This connectionId is used for further transactions. On failure it returns zero.

The closeConnection method closes the connection that is currently opened given a valid connectionId. It returns true indicating the successful close of the connection if the connectionId provided is a valid one. If the connectionId is invalid it returns false.

The RecordSetI method executes the query which is passed as the parameter to it. It creates a RecordSet object. It returns a reference to the RecordSet Object .

The execute method executes the DML/DDDL statement which is passed as the parameter to it. On successful execution of the statement it returns true. Otherwise it returns false.

The implementation of the above methods are provided in the DBMServant class.

The idl file is compiled using a IDL-to-Java compiler. Compiling the idl file generates the following classes and the interface.

- `_DBManagerImplBase` and `_RecordSetImplBase`
- `DBManagerHelper` and `RecordSetIHelper`
- `DBManagerHolder` and `RecordSetIHolder`
- `_st_DBManager` and `_st_RecordSetI`
- `_sk_DBManager` and `_sk_RecordSetI`
- `_example_DBManager` and `_example_RecordSetI`
- `DBManager` and `RecordSetI`
- `DBManagerOperations` and `RecordSetIOperations`
- `_tie_DBManager` and `_tie_RecordSetI`

Figure 6.3 shows the Java classes and interfaces generated by the idl2java compiler. The functions of the various classes and the interface generated are as follows

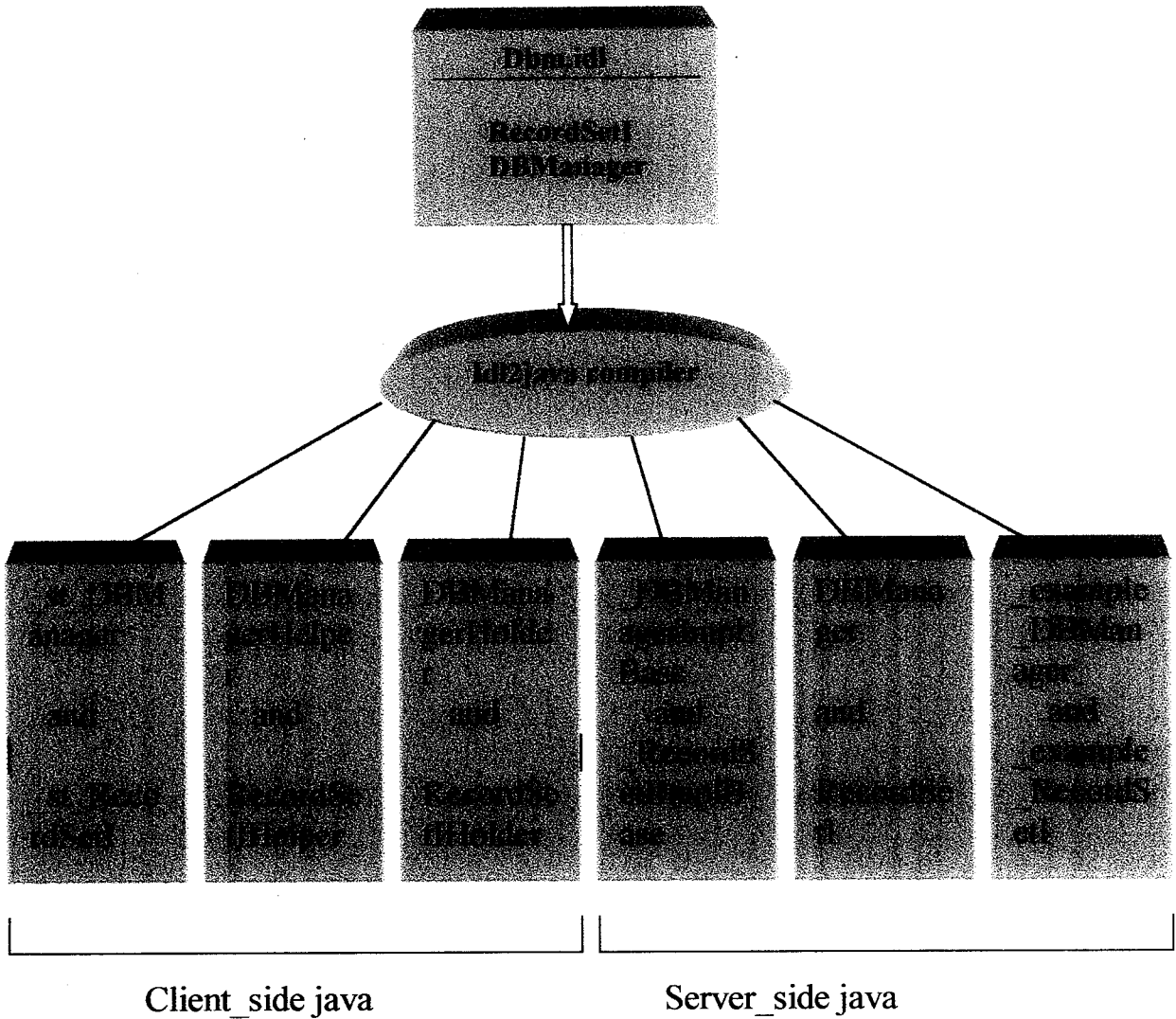


Figure 6.3 java classes and interfaces generated by idl2java compiler

_DBManagerImplBase

This is the java class that implements the CORBA server side skeleton for DBManager. It unmarshals the arguments for the DBManager object. In addition, this is the class that brings together the CORBA and Java object models. It does this by being a Java object that also implements the Java **org.omg.CORBA.object** interface. This is the root CORBA interface; all CORBA objects must implement it. The implementation of DBManager will simply inherit this dual functionality by extending the _DBManager class.

_st_DBManager

_st_DBManager is a Java class that implements the client side stub for the DBManager object. It's an internal implementation of the DBManager interface that provides marshaling functions.

DBManagerHelper

DBManagerHelper is a Java class that provides useful helper functions for DBManager clients. The compiler automatically generates codes for a narrow function that lets clients cast CORBA object references to the DBManager type. In addition to the required CORBA helper functions, the visibroker implementations provide a very useful, but non standard function called bind; it is used by clients to locate objects of this type. In this project, this is used to find remote objects that implements the DBManager interface.

DBManagerHolder

DBManagerHolder is a Java class that holds a public instance member of type DBManager. It is used by clients and servers to pass objects of type DBManager as out and inout parameters inside method invocations.

DBManager

DBManager is a Java Interface. It maps the DBManager interface to the corresponding Java interface. The code that implements this interface must be provided.

_example_DBManager

_example_DBManager is an example class for the DBManager object implementation. It contains constructors and example methods for the methods that were found in the dbm.idl. With this example as a starting point, completing the DBManager object implementation is easy.

RecordSet

RecordSet is a Java interface. It maps the RecordSet interface to the corresponding Java interface.

The functions of the classes generated for the RecordSet are same as that for DBManager.

6.4 DBMServant

Description

The DBMServant class contains the implementations for all the methods that are specified in the Interface DBManager.

In addition DBMServant also contains the methods insertConnectionIntoDB and UpdateConnectionInDB. The method InsertConnectionInDB inserts into the database the connectionId, DSN and the time at which the connection is established to the database for every connection that is made. The method UpdateConnectionInDB insert the exit time of a particular given its connection ID. These information are later used for monitoring the connections.

With the CORBA inheritance model, the server implementation class is always derived from the corresponding _DBManagerImplBase class. This is how the servant class inherits the functionality of both the CORBA and Java object models. The skeleton function is also inherited from this class. These are up-calls that allow the ORB to automatically invoke the object's methods. In addition to implementing the DBMServer functions, the constructor for the DBMServant class. The constructor calls its super—in this case, the parent is the skeleton class—to create a named implementation object. This means that each instance of DBMServant will have a persistent name.

Usage

The usage of this class is to get the reference to this object and invoke `openConnection` to get unique `connectionId`. All other database operations like executing a DDL/DML/SQL can be done by invoking the appropriate methods.

Related Class

The class that is related to the `DBMServant` class is the `RecordSet` class. The queries in this class is sent to the `RecordSet` class for actual execution. After the execution of the query the results are returned back to the `DBMServant` class.

6.5 RecordSet class

Description

The `RecordSet` class contains all the implementations of the methods present in the `RecordSetI` interface. The server implementation is derived from the corresponding `RecordSetImplBase` class.

Usage

The usage of the `RecordSet` class is to get the reference to this object and invoke the appropriate navigation methods.

6.6 DBMServer class

Description:

The `DBMServer` class provides the main function on the server side. The main program is managing multiple objects, each running in its own thread.

Usage:

The DBMServer class performs the following functions:

1. Initialize the ORB.
2. Initialize the BOA.
3. Create a DBMServant object.
4. Export to the ORB the newly created object.
5. Wait for incoming requests.

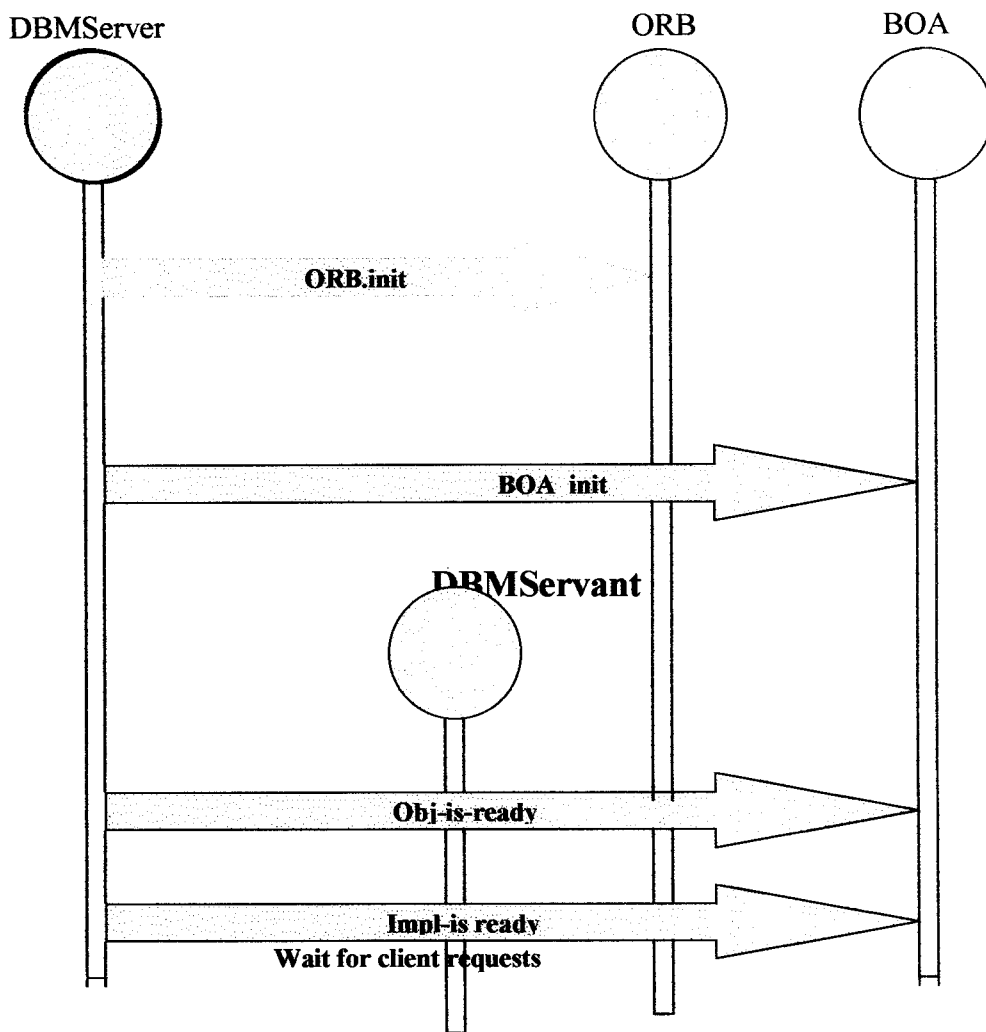


Figure 6.6 object interaction diagram for server side

The **org.omg.CORBA.ORB** class is part of the visibroker run-time. It implements in Java most of the function that OMG specifies in the CORBA::ORB interface. The `org.omg.CORBA.ORB.init` is a Java class method. This method returns an object of type `org.omg.CORBA.ORB`. This method also lets to pass in the main program's command line arguments, which lets to set certain ORB related properties at run-time. Now, a reference to the visibroker ORB object is obtained. This object is invoked to initialize the BOA. The `BOA_init` call returns an object reference for BOA. This reference is then used to register with the BOA the newly created DBMServant object. Finally, the BOA is told that the object is ready for business. The server side is now complete.

Related Classes:

The classes related to the DBMServer class is the DBMServant class.

6.7 DBMClient class

Description:

There are two client programs. One is the Java client and the other is the VC++ client.

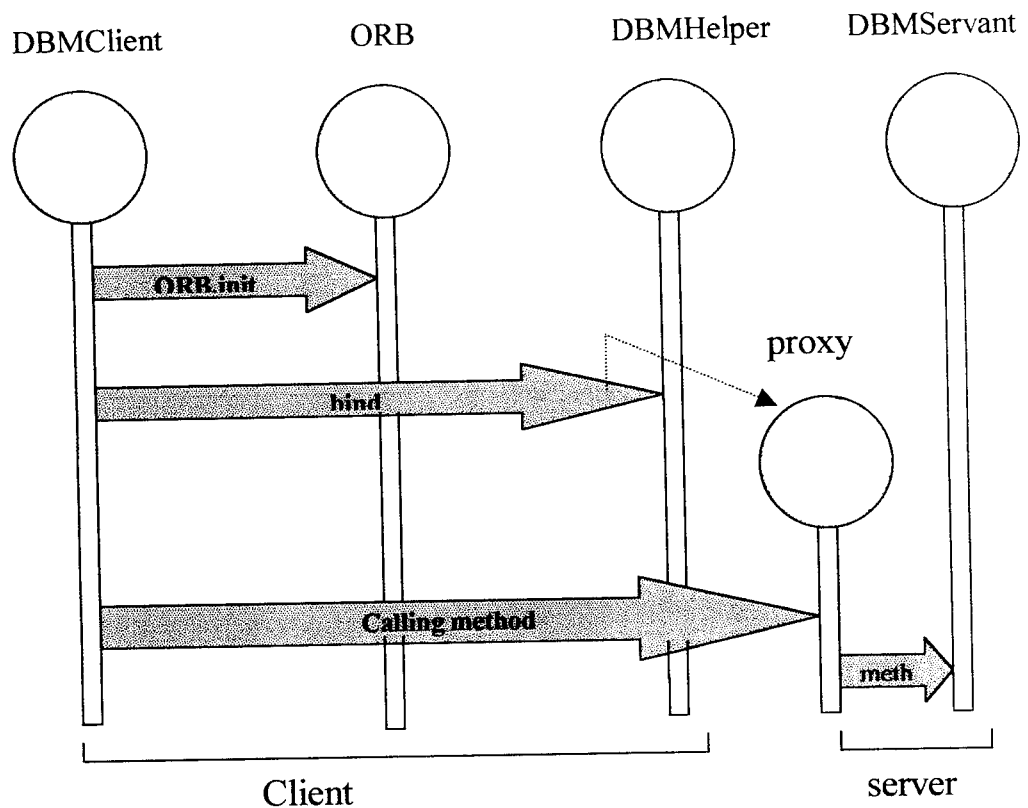
The Java client is explained here and the VC++ client is dealt later.

The Java client is DBMClient. This class provides the main method.

Usage:

The object interaction diagram is shown in figure 4.7. The DBMClient class performs the following functions:

- 1.Initialize the ORB.
- 2.Locate the remote object.
- 3.Invoke the appropriate methods.



6.7 Client side object interaction

To invoke the methods in the server, the bind method of the DBManagerHelper class is used. Now, the methods in the server is called as if calling a local method no matter wherever the method resides in the network.

Related classes:

The DBMClient class has no related classes.

Visibroker osagent

To run the client/server program, visibroker osagent is started first. This osagent provides fault tolerant location services. The osagent must be running on atleast one machine in the LAN environment.

7. VC++ connectivity

7.1 Introduction

One of the main benefits of CORBA is that it lets objects communicate across languages and operating systems. Here a VC++ client is developed and made to talk to the Java counterparts. A VC++ client can invoke the Java objects. This is established using Borland/Visigenic Visibroker for VC++

A VC++ client is also developed to illustrate the reusability of code in this project. A VC++ client can communicate with the database and perform database programming without knowing the interfaces involved in database programming. To train a programmer in database programming in VC++ consumes time. This problem can be alleviated as follows:

7.2 The IDL

The IDL file is the same as for the Java client. Because CORBA IDL is a language-independent language, the `dbm.idl` used in Java can be used again in VC++. This IDL file is mapped to something that C++ clients and servers can understand. This is done using an IDL-to-CPP compiler. The compiler generates four C++ files from the IDL input. The files generated are shown in figure 6.2

The functions performed by the various classes are as follows:

DBManager_s.cpp

The `DBManger_s.cpp` is the server skeleton for the methods of the `DBManager` class. This is the code that unmarshals calls for the `DBManager` object and then invokes the implementation of the object.

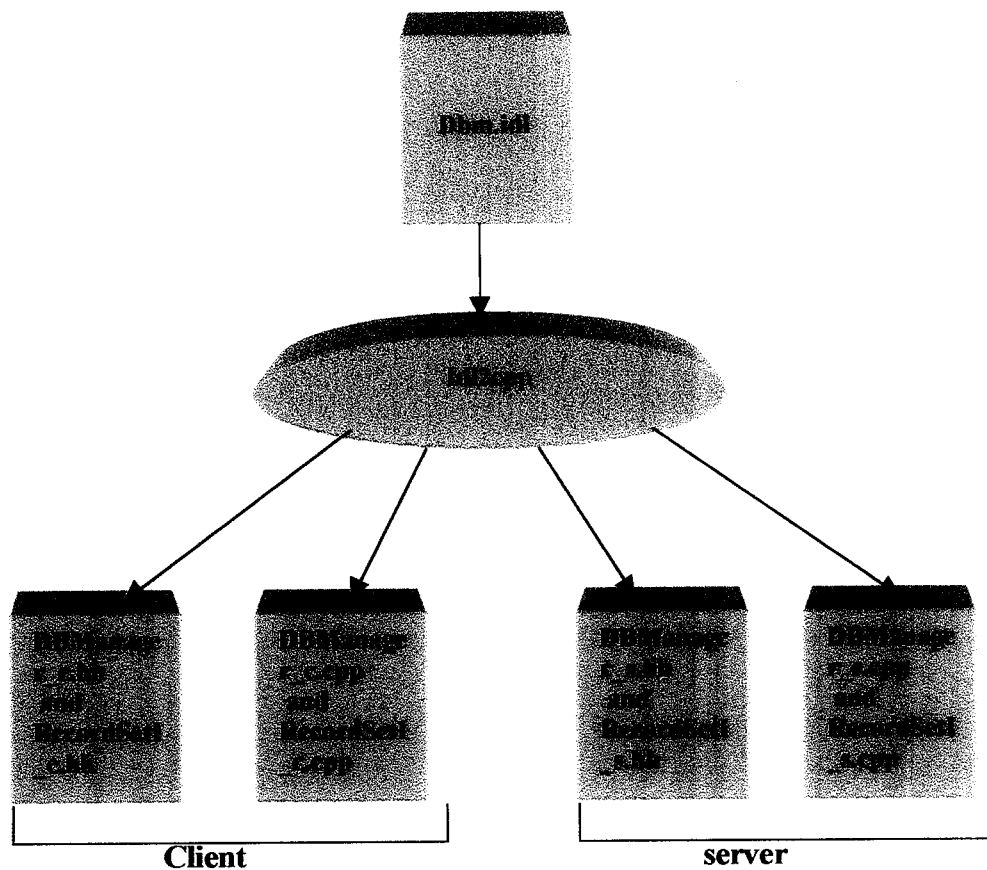


Figure 7.2 C++ files generated by idl2cpp compiler

DBManager_s.hh

DBManager_s.hh is the server header file that includes the class definitions for the server skeletons implemented in DBManager_s.cpp

DBManager_c.cpp

DBManager_c.cpp contains a class called DBManager that serves as a proxy on the client for the DBManager object. It provides stubs and marshaling functions for all the methods defined in the DBManager interface. It also implements a bind method that helps the client locate the DBManager server.

DBManager_c.hh

DBManager_c.hh is a client header file that includes declarations and class definitions for the stub implementations in DBManager_c.cpp.

The functions of the classes generated for the RecordSetI is same as that for DBManager interface

7.3 VC++ Implementation

The VC++ implementation of the methods present in the DBManager interface and the RecordSetI interface which corresponds to the DBMServant class and RecordSet class in Java is not necessary. The implementation that is provided once in Java can be used by any other clients. This is the major advantage of this project.i.e., the methods that are implemented in the DBMServant class can be used by the C++ client without any manual code modifications.

7.4 VC++ Client

The VC++ client is very similar to the Java counterpart. . It initializes the ORB,locates the remote object and invokes the appropriate methods. The bind method is used by the client in order to locate the DBManager proxy server.

7.5 VC++ Server

The Java server acts as the server for both Java and VC++. The VC++ Client can communicate with the Java server as shown in figure 6.5

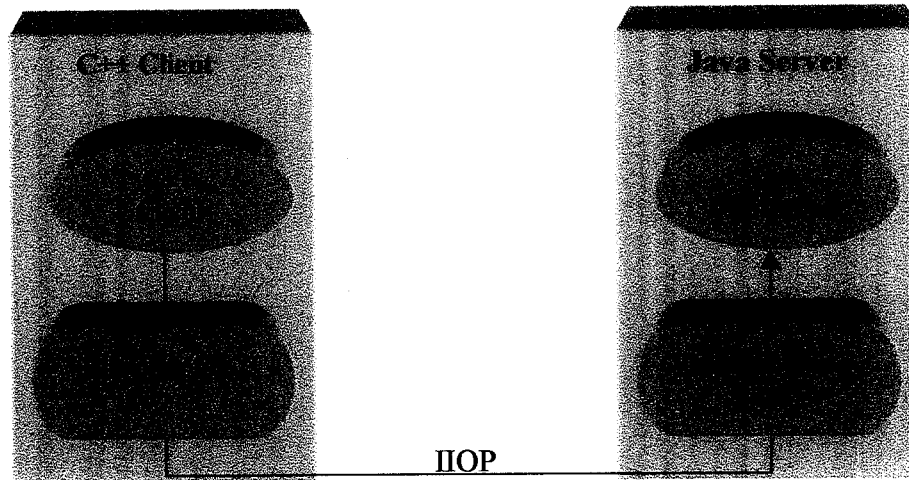


Figure 7.5 C++ client invoking Java Server

The figure7.5 demonstrates how a C++ ORB on the client side can talk to a Java ORB on the server side via the Internet Inter ORB Protocol(IIOP) and illustrates that ORBs allow to transparently invoke objects across languages.In this case VC++ client calls Java server object without requiring any changes to either side.

7.6 Running the client server application

To run the client/server program, first the visibroker OSAgent is started. The OSAgent must be present atleast on one computer in the network and the client must be informed about the location of the OSAgent. After that the server is started. The server program must always be running to service the clients at any time. Finally the client is fired off.

8. Connection Monitoring

8.1 Illustration Of Connection Monitoring

A Connection Monitoring program is developed for the network administrator to monitor the clients that are presently connected to the database. This program displays the connectionId, Data Source Name, Start-time of the connection and End-time of the connection of all the clients that are connected to the database during a particular time interval.

In the DBMServant class, whenever a connection is established in the openConnection method, the start-time at which the connection is established is got and put into the database. Likewise the end-time at which the connection terminates is also got in the closeConnection method and put in the database.

The start-time and end-time are obtained with the help of the TimeStamp class which comes under the java.util package. The arguments passed to TimeStamp are year, month, day, hour, minute, second and millisecond.

The method insertConnectionIntoDB in the DBMServant class adds start-time into the database along with the the connectionId and DSN of that particular connection.

The method UpdateConnectionInDB in the DBMServant class takes the connectionId and end-time as its arguments and inserts the end-time into the database such that the connectionId of the start-time and the end-time are the same.

Given a particular time interval, the class DBMMonitor displays all the clients that were connected during the given time interval. The details

includes the connectionId, DSN, start-time and end-time of the connection with the database.

This connection Monitoring allows the network administrator to have an idea of the various clients that are connected during a particular time interval.



9. System Testing

Software testing is considered as one of the final opportunity for the developer to detect and convert (or) rectify any defects that may be in the software. Software testing is the process of testing a program with the explicit intention of finding errors that are making the program fail.

In short, system testing and quality assurance is a review of the software products and related documentation for completion, correctness, reliability and maintainability.

Software testing is defined as a process by which one detects the defects in the software. The first step in system testing is to prepare a plan that will test all aspects of the system. System testing can be grouped into two levels

1. Unit Testing

2. Integrated Testing

The system here is tested on the top of a real time application namely Employee Management System where the client here is the Java client, the middletier is the ODBC server and the backend being MSAccess.

9.1 Employee Management System

Features Of Employee Management System

The features of the system includes

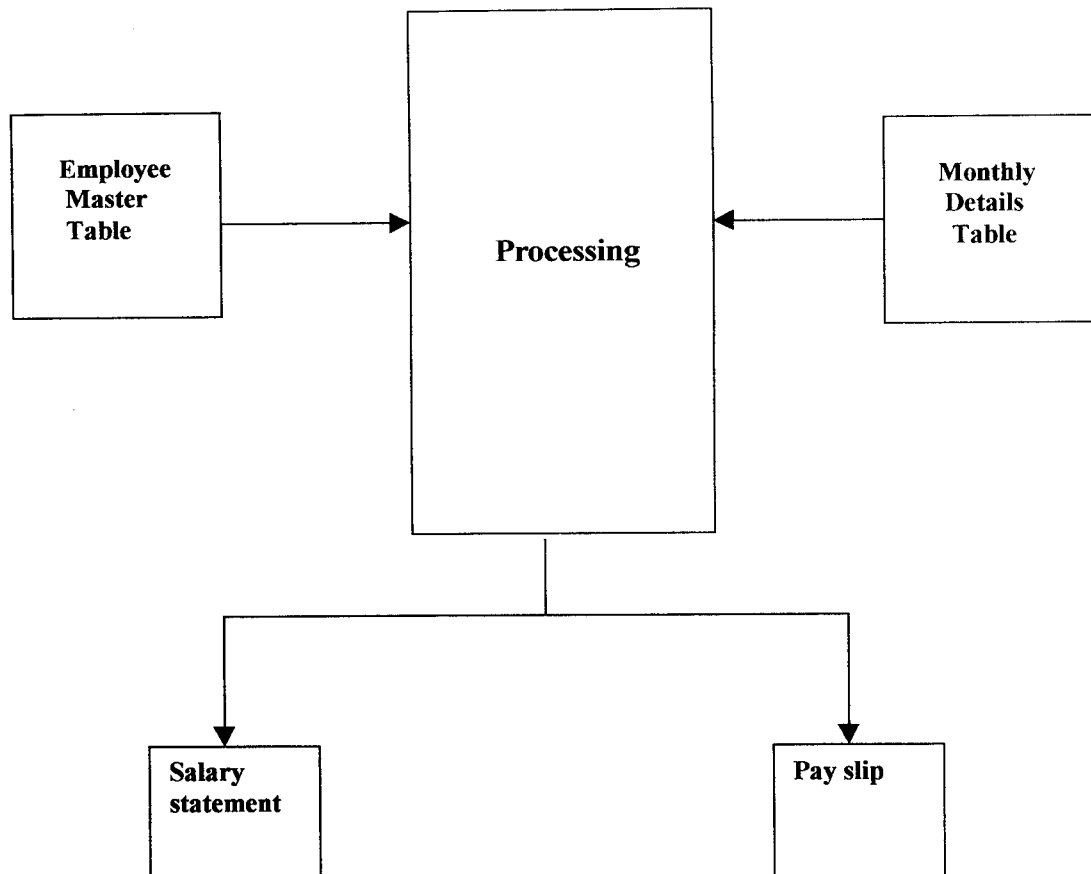
1. giving up-to-date minute status and information about the employees and their monthly salary details.
2. Computerising various operations like
 - Employee details
 - Monthly Salary details
 - Leave details

Employee Management System Overview

The proposed system consists of following features which are desirable in any Employee Management System activities

- Provision for on-line queries and status
- Centralised database
- Provision for on-line data validation
- User friendly, easy to use system for the non technical employees of an organisation
- System with full data security and all recovery facilities.

Employee Management System Flowchart



Input Design

In input design user oriented inputs are converted to computer based format. Inaccurate inputting of data may lead to flaws during processing. Commonly these mistakes occur at data entry level. Input design objectives in the application are

1. controlling amounts of inputs
2. avoid data errors
3. avoid extra steps
4. keep process simple

Output Design

In output design the emphasis is producing the hard copy of the information or displaying the output.

One of the most important reasons which tempts the system to go for a new system is the output. Hence a predetermination of the output requirements is required before going for the actual system.

The main hard copy reports generated for the Employee Management system are

1. salary statement
2. pay slip

Database design

The most important aspect of building an application system is the design of tables. The data they store must be organised according to user requirements. The important tables constructed are as follows

Employee Master

The Employee Master table captures the details of the employee.

Monthly Details

This table captures the monthly details of the employees such as number of working days, number of days worked, etc.

Testing

The application is designed using Java. All the queries are done using the methods in the RecordSet class. The DBMServant class is again the implementation class and the DBMServer is the server for this application.

10. Conclusion

Middleware Database drivers is successfully developed. The three tiers, client tier, ODBC server tier and the database tier are developed and the connection between the client and the server is established. The client can now communicate with any database on the backend without the database drivers on the client machine. The program is tested using an Employee Management System application and it found working. The VC++ client also communicates with the Java server and performs database manipulations.

The Connection Monitoring program successfully displays the connections established with the ODBC server in a particular time interval. A chart displaying the connections that are maintained with the ODBC server at various time intervals is also drawn.

Scope for further development

For the purpose of Network Administrator, the project can be further developed to monitor parameters like connection response time, determining the performance of the server, etc.

Middleware database can also be extended to work with other clients like Visual Basic client, Power Builder client, etc.

BIBLIOGRAPHY

URLs

[1] <http://www.networkcomputing.com/713/713workODBC.html>

Description of ODBC and JDBC drivers and networking concepts.

[2] <http://www.ftech.co.uk/>

Description of three tier architecture and its advantages.

[3] <http://www.omg.org/>

Description of CORBA.

[4] <http://java.sun.com/products/jdbc/>

Description of JDBC documentation.

[5] <http://www.sei.cmu.edu/str/description/threetier-body.html>

Description of three tier architecture.

Books

[1] Cay S. Horstmann, Gary Cornell, "Core Java2 Volume II Advanced Features" Prentice Hall, pp 203-253, 298-317, 2000

[2] Robert Orfali, Dan Harkey, "Client/server programming with java and CORBA" Shroff Publishers and Distributers Pvt. Ltd., pp 10-16, 27-30, 64-82, 97-111, 521-573, 1998

[3] Patrick Naughton, Herbert Schilt, "The Complete Referece Java2", Tata McGraw Hill Publications, pp 717-780, 1999

/***** IDL *****/

```
module dbm
{
    interface RecordSetI
    {
        boolean first();
        boolean next();
        boolean previous();
        boolean last();
        boolean isFirst();
        boolean isLast();
        string getString(in long col);
    };
    interface DBManager
    {
        long openConnection(in string dsn, in string user, in
string pwd);
        boolean closeConnection(in long connectionId);
        RecordSetI createRecordSet(in long connectionId, in string
query);
        boolean execute(in long connectionId, in string stmt);
    };
};
```

```
import dbm.*;
import org.omg.CORBA.*;
import java.sql.*;
import java.util.*;
/*****
 * CLASS
 *     DBMServant
 * DESCRIPTION
 *     Contains all the interface function implementations, which
helps the user to open and close db connections and as well as to
execute DML/DDL/SQL statements
 * USAGE
 *     Get the reference to this object and invoke OpenConnection
to get the unique connection id. With the help of this connection
id, all other database operations like executing a DDL/DML/SQL can
be done by invoking appropriate methods.
 * RELATED CLASSES
 *     RecordSet
```

```

public class DBMServant extends _DBManagerImplBase
{
    ///
    // Unique Connection Identifier.
    //
    private static int nextConnectionId = 0;

    ///
    // Connections
    //
    private Hashtable connections = new Hashtable();

    public DBMServant(String name) {
        super(name);
    }

    ////////////////////////////////////////////////////////////////////

    // Opens a new database connection.
    //
    // RETURN VALUE:
    // On SUCCESS - Returns Unique connection identifier , which
can be used for further transaction.
    // On FAILURE - It returns 0 (Zero).

    ////////////////////////////////////////////////////////////////////

    public int openConnection(String dsn, String user, String pwd)
    {
        System.out.println(dsn + " " + user + " " + pwd);
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con =
DriverManager.getConnection("jdbc:odbc:" + dsn,user,pwd);
            ///
            // Incrementing the unique connection id for the this
connection.
            //
            nextConnectionId++;

            ///
            // Making the connection entry in the hash table.
            //
            connections.put("" + nextConnectionId, con);

            java.util.Date d = new java.util.Date();

```

```

        Timestamp startTime = new Timestamp(d.getYear(),
d.getMonth(),
                                d.getDate(),
                                d.getHours(),
                                d.getMinutes(),
                                d.getSeconds(), 0);

        InsertConnectionIntoDB(nextConnectionId, dsn,
startTime);

        ///
        // Returning the connection identifier of this
connection
        //
        return nextConnectionId;
    } catch (Exception ex) {
        ex.printStackTrace();
        return 0;
    }
}

////////////////////////////////////
/////
// Closes the connection with the help of the specified unique
// connection identifier.
//
// RETURN VALUE:
// On SUCCESS - true
// On FAILURE - false

////////////////////////////////////
/////
public boolean closeConnection(int connectionId)
{
    boolean rv = false;

    try {
        Connection con = (Connection) connections.get("" +
connectionId);
        con.close();

        connections.remove("" + connectionId);
        rv = true;

        java.util.Date d = new java.util.Date();

```

```

        Timestamp endTime = new Timestamp(d.getYear(),
d.getMonth(),
                                d.getDate(),
                                d.getHours(),
                                d.getMinutes(),
                                d.getSeconds(), 0);

        UpdateConnectionInDB(connectionId, endTime);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return rv;
}

```

```

////////////////////////////////////
/////
// Executes the given query and creates a RecordSet.
//
// RETURN VALUE:
// RecordSet object reference will be returned.

```

```

////////////////////////////////////
/////
public RecordSetI createRecordSet(int connectionId, String
query)
{
    RecordSetI recordSet = null;
    try {
        Connection con = (Connection) connections.get("" +
connectionId);
        Statement stmt = con.createStatement();

        ResultSet rs = stmt.executeQuery(query);

        recordSet = new RecordSet(rs);

        ORB orb = ORB.init();
        BOA boa = orb.BOA_init();
        boa.obj_is_ready(recordSet);

    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return recordSet;
}

```

```

////////////////////////////////////
/////
    // Executes the given DML/DDDL.
    //
    // RETURN VALUE:
    // On SUCCESS - true
    // On FAILURE - false

////////////////////////////////////
/////
public boolean execute(int connectionId, String sql)
{
    boolean rv = false;

    try {
        Connection con = (Connection) connections.get("" +
connectionId);
        Statement stmt = con.createStatement();

        System.out.println(sql);

        stmt.execute(sql);

        rv = true;
    } catch (Exception ex) {
        System.out.println("ERROR: " + ex);
//        ex.printStackTrace();
    }

    return rv;
}

public void InsertConnectionIntoDB(int connectionId, String
dsn,
                                Timestamp startTime)
{
    try {
        String st = startTime.toString();

        String SQL = "INSERT INTO Connections VALUES (" +
                    connectionId + ", '" + dsn + "', '" +
                    st + "', '" + st + "')";

        System.out.println(SQL);
    }
}

```

```

        Connection tcon =
DriverManager.getConnection("jdbc:odbc:dbmadmin",
                            "admin", "");
        Statement tstmt = tcon.createStatement();
        tstmt.execute(SQL);

        tstmt.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public void UpdateConnectionInDB(int connectionId, Timestamp
endTime)
{
    try {
        String st = endTime.toString();

        String SQL = "UPDATE Connections SET EndTime = '" +
endTime +
                            "' WHERE connectionId = " +
connectionId;

        System.out.println(SQL);

        Connection tcon =
DriverManager.getConnection("jdbc:odbc:dbmadmin",
                            "admin", "");
        Statement tstmt = tcon.createStatement();
        tstmt.execute(SQL);

        tstmt.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}

```

```

*****
import org.omg.CORBA.*;
import java.sql.*;
import java.util.*;
import java.io.*;
import dbm.*;
/*****/
* CLASS

```

```

*   RecordSet
* DESCRIPTION
*   Contains all the interface function implementations, which
*   helps the user to navigate through the record set.
* USAGE
*   Get the reference to this object and invoke appropriate
*   navigation methods.
* RELATED CLASSES
*   none

```

```

/*****

```

```

public class RecordSet extends _RecordSetIIImplBase
{

```

```

    ///
    // Reference to ResultSet.
    //
    private java.sql.ResultSet rs;

```

```

    ///
    // Constructor with initializer.
    //
    public RecordSet(ResultSet rs) {
        this.rs = rs;
    }

```

```

    ///
    // Moves the pointer to the first record.
    //
    public boolean first()
    {
        boolean rv = false;
        try {
            return rs.first();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return rv;
    }

```

```

    ///
    // Moves the pointer to the next record.
    //
    public boolean next()
    {
        boolean rv = false;
        try {

```

```

        return rs.next();
    } catch (Exception ex)    {
        ex.printStackTrace();
    }
    return rv;
}

///  

// Moves the pointer to the previous record.  

///  

public boolean previous()
{
    boolean rv = false;

    try {
        rv = rs.previous();
    } catch (Exception ex)    {
        ex.printStackTrace();
    }
    return rv;
}

///  

// Moves the pointer to the last record.  

///  

public boolean last()
{
    boolean rv = false;
    try {
        rv = rs.last();
    } catch (Exception ex)    {
        ex.printStackTrace();
    }
    return rv;
}

///  

// Checks whether the record pointer is in first record  

///  

public boolean isFirst()
{
    boolean rv = false;
    try {
        rv = rs.isFirst();
    } catch (Exception ex)    {
        ex.printStackTrace();
    }
}

```

```

        return rv;
    }

    ///
    // Checks whether the record pointer is in last record
    //
    public boolean isLast()
    {
        boolean rv = false;
        try {
            rv = rs.isLast();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return rv;
    }

    ///
    // Returns the value of the specified column of the current
row.
    //
    public String getString(int col) {
        String rv = "";
        try {
            rv = rs.getString(col);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return rv;
    }
}

*****
import org.omg.CORBA.*;

public class DBMServer {
    public static void main(String[] args) {
        ORB orb = ORB.init();
        BOA boa = orb.BOA_init();
        DBMServant dbm = new DBMServant("dbm");
        boa.obj_is_ready(dbm);
        System.out.println("Server ready");
        boa.impl_is_ready();
    }
}
*****

```



```

import org.omg.CORBA.*;
import java.awt.*;
import java.awt.event.*;
import dbm.*;

public class DBMClient extends Frame implements ActionListener {
    private TextField txtRollno;
    private TextField txtName;
    private TextField txtAge;

    private DBManager dbm;

    private String    dsn;
    private String    user;
    private String    pwd;

    private int       conId;

    private ORB orb;

    public DBMClient(String title, String[] args) {
        super(title);

        dsn = args[0];
        user = args[1];
        pwd = args[2];

        try {
            orb = ORB.init();

            dbm = DBManagerHelper.bind(orb, "dbm");
            conId = dbm.openConnection(dsn, user, pwd);
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                dispose();
                try {
                    dbm.closeConnection(conId);
                    dbm = null;
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        });
    }
}

```

```

txtRollno = new TextField();
txtName   = new TextField();
txtAge    = new TextField();

Button b1 = new Button("Insert");
Button b2 = new Button("Update");
Button b3 = new Button("Delete");
Button b4 = new Button("Search");

setLayout(new GridLayout(5, 2));
add(new Label("Roll Number: "));
add(txtRollno);
add(new Label("Name: "));
add(txtName);
add(new Label("Age: "));
add(txtAge);

add(b1);
add(b2);
add(b3);
add(b4);

b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);

pack();
}

public void actionPerformed(ActionEvent ae) {
    String cmd = ae.getActionCommand();

    try {
        if (cmd.equalsIgnoreCase("Search")) {
            String SQL = "SELECT * FROM Biodata WHERE RollNo =
" +
                        txtRollno.getText();
            RecordSetI rs = dbm.createRecordSet(conId, SQL);
            if (rs.next()) {
                txtName.setText(rs.getString(2));
                txtAge.setText(rs.getString(3));
            } else {
                txtName.setText("Not Found");
                txtAge.setText("Not Found");
            }
        }
    }
}

```

```

    } else if (cmd.equalsIgnoreCase("Update"))    {
        String SQL = "Update Biodata SET Name='" +
            txtName.getText() + "', Age=" +
            txtAge.getText();
        System.out.println(SQL);
        dbm.execute(conId, SQL);
    } else if (cmd.equalsIgnoreCase("Insert"))  {
        String SQL = "Insert INTO Biodata VALUES(" +
            txtRollno.getText() + ", '" +
            txtName.getText() + "', " +
            txtAge.getText() + ")";
        System.out.println(SQL);
        dbm.execute(conId, SQL);
    } else if (cmd.equalsIgnoreCase("Delete"))  {
        String SQL = "Delete from Biodata " +
            " WHERE Rollno = " +
txtRollno.getText();
        System.out.println(SQL);
        dbm.execute(conId, SQL);
    }
} catch (Exception ex) {
    ex.printStackTrace();
}
}

public static void main(String[] args) {
    if (args.length == 3) {
        (new DBMClient("DB Manager Demo", args)).show();
    } else {
        System.out.println("Usage: java DBMClient dsn user
pwd");
    }
}
}

```

```

package dbm;
abstract public class _DBManagerImplBase extends
org.omg.CORBA.portable.Skeleton implements dbm.DBManager {
    protected _DBManagerImplBase(java.lang.String name) {
        super(name);
    }
    protected _DBManagerImplBase() {
    }
    public java.lang.String[] _ids() {
        return __ids;
    }
    private static java.lang.String[] __ids = {
        "IDL:dbm/DBManager:1.0"
    };
    public org.omg.CORBA.portable.MethodPointer[] _methods() {
        org.omg.CORBA.portable.MethodPointer[] methods = {
            new org.omg.CORBA.portable.MethodPointer("openConnection",
0, 0),
            new org.omg.CORBA.portable.MethodPointer("closeConnection",
0, 1),
            new org.omg.CORBA.portable.MethodPointer("createRecordSet",
0, 2),
            new org.omg.CORBA.portable.MethodPointer("execute", 0, 3),
        };
        return methods;
    }

    public boolean _execute(org.omg.CORBA.portable.MethodPointer
method, org.omg.CORBA.portable.InputStream input,
org.omg.CORBA.portable.OutputStream output) {
        switch(method.interface_id) {
            case 0: {
                return dbm._DBManagerImplBase._execute(this,
method.method_id, input, output);
            }
        }
        throw new org.omg.CORBA.MARSHAL();
    }

    public static boolean _execute(dbm.DBManager _self, int
_method_id, org.omg.CORBA.portable.InputStream _input,
org.omg.CORBA.portable.OutputStream _output) {
        switch(_method_id) {
            case 0: {
                java.lang.String dsn;
                dsn = _input.read_string();
                java.lang.String user;

```

```

        user = _input.read_string();
        java.lang.String pwd;
        pwd = _input.read_string();
        int _result = _self.openConnection(dsn,user,pwd);
        _output.write_long(_result);
        return false;
    }
    case 1: {
        int connectionId;
        connectionId = _input.read_long();
        boolean _result = _self.closeConnection(connectionId);
        _output.write_boolean(_result);
        return false;
    }
    case 2: {
        int connectionId;
        connectionId = _input.read_long();
        java.lang.String query;
        query = _input.read_string();
        dbm.RecordSetI _result =
        _self.createRecordSet(connectionId,query);
        dbm.RecordSetIHelper.write(_output, _result);
        return false;
    }
    case 3: {
        int connectionId;
        connectionId = _input.read_long();
        java.lang.String stmt;
        stmt = _input.read_string();
        boolean _result = _self.execute(connectionId,stmt);
        _output.write_boolean(_result);
        return false;
    }
    }
    throw new org.omg.CORBA.MARSHAL();
}
}

```

```

*****
package dbm;
public class _example_DBManager extends dbm._DBManagerImplBase {
    public _example_DBManager(java.lang.String name) {
        super(name);
    }
    public _example_DBManager() {
        super();
    }
}

```

```

public int openConnection(
    java.lang.String dsn,
    java.lang.String user,
    java.lang.String pwd
) {
    // implement operation...
    return 0;
}
public boolean closeConnection(
    int connectionId
) {
    // implement operation...
    return false;
}
public dbm.RecordSetI createRecordSet(
    int connectionId,
    java.lang.String query
) {
    // implement operation...
    return null;
}
public boolean execute(
    int connectionId,
    java.lang.String stmt
) {
    // implement operation...
    return false;
}
}

```

```

package dbm;
public class _st_DBManager extends
org.omg.CORBA.portable.ObjectImpl implements dbm.DBManager {
    public java.lang.String[] _ids() {
        return __ids;
    }
    private static java.lang.String[] __ids = {
        "IDL:dbm/DBManager:1.0"
    };
    public int openConnection(
        java.lang.String dsn,
        java.lang.String user,
        java.lang.String pwd
    )

```



```

    {
        try {
            org.omg.CORBA.portable.OutputStream _output =
this._request("openConnection", true);
            _output.write_string(dsn);
            _output.write_string(user);
            _output.write_string(pwd);
            org.omg.CORBA.portable.InputStream _input =
this._invoke(_output, null);
            int _result;
            _result = _input.read_long();
            return _result;
        }
        catch(org.omg.CORBA.TRANSIENT _exception) {
            return openConnection(
                dsn,
                user,
                pwd
            );
        }
    }
    public boolean closeConnection(
        int connectionId
    ) {
        try {
            org.omg.CORBA.portable.OutputStream _output =
this._request("closeConnection", true);
            _output.write_long(connectionId);
            org.omg.CORBA.portable.InputStream _input =
this._invoke(_output, null);
            boolean _result;
            _result = _input.read_boolean();
            return _result;
        }
        catch(org.omg.CORBA.TRANSIENT _exception) {
            return closeConnection(
                connectionId
            );
        }
    }
    public dbm.RecordSetI createRecordSet(
        int connectionId,
        java.lang.String query
    )
    {
        try {

```

```

        org.omg.CORBA.portable.OutputStream _output =
this._request("createRecordSet", true);
        _output.write_long(connectionId);
        _output.write_string(query);
        org.omg.CORBA.portable.InputStream _input =
this._invoke(_output, null);
        dbm.RecordSetI _result;
        _result = dbm.RecordSetIHelper.read(_input);
        return _result;
    }
    catch(org.omg.CORBA.TRANSIENT _exception) {
        return createRecordSet(
            connectionId,
            query
        );
    }
}
public boolean execute(
    int connectionId,
    java.lang.String stmt
) {
    try {
        org.omg.CORBA.portable.OutputStream _output =
this._request("execute", true);
        _output.write_long(connectionId);
        _output.write_string(stmt);
        org.omg.CORBA.portable.InputStream _input =
this._invoke(_output, null);
        boolean _result;
        _result = _input.read_boolean();
        return _result;
    }
    catch(org.omg.CORBA.TRANSIENT _exception) {
        return execute(
            connectionId,
            stmt
        );
    }
}
}
}

```

```

package dbm;
public class _tie_DBManager extends dbm._DBManagerImplBase {
private dbm.DBManagerOperations _delegate;
public _tie_DBManager(dbm.DBManagerOperations delegate,
java.lang.String name)

```



```

{
    super(name);
    this._delegate = delegate;
}
public _tie_DBManager(dbm.DBManagerOperations delegate) {
    this._delegate = delegate;
}
public dbm.DBManagerOperations _delegate() {
    return this._delegate;
}
public int openConnection(
    java.lang.String dsn,
    java.lang.String user,
    java.lang.String pwd
) {
    return this._delegate.openConnection(
        dsn,
        user,
        pwd
    );
}
public boolean closeConnection(
    int connectionId
) {
    return this._delegate.closeConnection(
        connectionId
    );
}
public dbm.RecordSetI createRecordSet(
    int connectionId,
    java.lang.String query
) {
    return this._delegate.createRecordSet(
        connectionId,
        query
    );
}
public boolean execute(
    int connectionId,
    java.lang.String stmt
) {
    return this._delegate.execute(
        connectionId,
        stmt);
}
}

```

```

package dbm;
public interface DBManager extends org.omg.CORBA.Object {
    public int openConnection(
        java.lang.String dsn,
        java.lang.String user,
        java.lang.String pwd
    );
    public boolean closeConnection(
        int connectionId
    );
    public dbm.RecordSetI createRecordSet(
        int connectionId,
        java.lang.String query
    );
    public boolean execute(
        int connectionId,
        java.lang.String stmt
    );
}

*****
* package dbm;
abstract public class DBManagerHelper {
    public static dbm.DBManager narrow(org.omg.CORBA.Object object)
    {
        return narrow(object, false);
    }
    private static dbm.DBManager narrow(org.omg.CORBA.Object
object, boolean is_a)
    {
        if(object == null) {
            return null;
        }
        if(object instanceof dbm.DBManager) {
            return (dbm.DBManager) object;
        }
        if(is_a || object._is_a(id())) {
            dbm.DBManager result = new dbm._st_DBManager();
            ((org.omg.CORBA.portable.ObjectImpl) result)._set_delegate
                (((org.omg.CORBA.portable.ObjectImpl)
object)._get_delegate());
            return result;
        }
        return null;
    }
    public static dbm.DBManager bind(org.omg.CORBA.ORB orb) {
        return bind(orb, null, null, null);
    }
}

```

```

    }
    public static dbm.DBManager bind(org.omg.CORBA.ORB orb,
java.lang.String name)
    {
        return bind(orb, name, null, null);
    }
    public static dbm.DBManager bind(org.omg.CORBA.ORB orb,
java.lang.String name, java.lang.String host,
org.omg.CORBA.BindOptions options) {
        return narrow(orb.bind(id(), name, host, options), true);
    }
    private static org.omg.CORBA.ORB _orb() {
        return org.omg.CORBA.ORB.init();
    }
    public static dbm.DBManager
read(org.omg.CORBA.portable.InputStream _input) {
        return dbm.DBManagerHelper.narrow(_input.read_Object(),
true);
    }
    public static void write(org.omg.CORBA.portable.OutputStream
_output, dbm.DBManager value) {
        _output.write_Object(value);
    }
    public static void insert(org.omg.CORBA.Any any, dbm.DBManager
value) {
        org.omg.CORBA.portable.OutputStream output =
any.create_output_stream();
        write(output, value);
        any.read_value(output.create_input_stream(), type());
    }
    public static dbm.DBManager extract(org.omg.CORBA.Any any) {
        if(!any.type().equal(type())) {
            throw new org.omg.CORBA.BAD_TYPECODE();
        }
        return read(any.create_input_stream());
    }
    private static org.omg.CORBA.TypeCode _type;
    public static org.omg.CORBA.TypeCode type() {
        if(_type == null) {
            _type = _orb().create_interface_tc(id(), "DBManager");
        }
        return _type;
    }
    public static java.lang.String id() {
        return "IDL:dbm/DBManager:1.0";
    }
}

```

```

package dbm;
final public class DBManagerHolder implements
org.omg.CORBA.portable.Streamable {
    public dbm.DBManager value;
    public DBManagerHolder() {
    }
    public DBManagerHolder(dbm.DBManager value) {
        this.value = value;
    }
    public void _read(org.omg.CORBA.portable.InputStream input) {
        value = DBManagerHelper.read(input);
    }
    public void _write(org.omg.CORBA.portable.OutputStream output)
    {
        DBManagerHelper.write(output, value);
    }
    public org.omg.CORBA.TypeCode _type() {
        return DBManagerHelper.type();
    }
}

*****
* package dbm;
public interface DBManagerOperations {
    public int openConnection(
        java.lang.String dsn,
        java.lang.String user,
        java.lang.String pwd
    );
    public boolean closeConnection(
        int connectionId
    );
    public dbm.RecordSetI createRecordSet(
        int connectionId,
        java.lang.String query
    );
    public boolean execute(
        int connectionId,
        java.lang.String stmt
    );
}

```

```

package dbm;
abstract public class _RecordSetIImplBase extends
org.omg.CORBA.portable.Skeleton implements dbm.RecordSetI {
    protected _RecordSetIImplBase(java.lang.String name) {
        super(name);
    }
    protected _RecordSetIImplBase() {
    }
    public java.lang.String[] _ids() {
        return __ids;
    }
    private static java.lang.String[] __ids = {
        "IDL:dbm/RecordSetI:1.0"
    };
    public org.omg.CORBA.portable.MethodPointer[] _methods() {
        org.omg.CORBA.portable.MethodPointer[] methods = {
            new org.omg.CORBA.portable.MethodPointer("next", 0, 0),
            new org.omg.CORBA.portable.MethodPointer("getString", 0,
1),
        };
        return methods;
    }
    public boolean _execute(org.omg.CORBA.portable.MethodPointer
method, org.omg.CORBA.portable.InputStream input,
org.omg.CORBA.portable.OutputStream output) {
        switch(method.interface_id) {
            case 0: {
                return dbm._RecordSetIImplBase._execute(this,
method.method_id, input, output);
            }
        }
        throw new org.omg.CORBA.MARSHAL();
    }
    public static boolean _execute(dbm.RecordSetI _self, int
_method_id, org.omg.CORBA.portable.InputStream _input,
org.omg.CORBA.portable.OutputStream _output) {
        switch(_method_id) {
            case 0: {
                boolean _result = _self.next();
                _output.write_boolean(_result);
                return false;
            }
            case 1: {
                int col;
                col = _input.read_long();
                java.lang.String _result = _self.getString(col);
                _output.write_string(_result);
            }
        }
    }
}

```

```

        return false;
    }
}
throw new org.omg.CORBA.MARSHAL();
}
}

```

```

package dbm;
public class _example_RecordSetI extends dbm._RecordSetIImplBase
{
    public _example_RecordSetI(java.lang.String name) {
        super(name);
    }
    public _example_RecordSetI() {
        super();
    }
    public boolean next() {
        // implement operation...
        return false;
    }
    public java.lang.String getString(
        int col
    ) {
        // implement operation...
        return null;
    }
}

```

```

package dbm;
public class _st_RecordSetI extends
org.omg.CORBA.portable.ObjectImpl implements dbm.RecordSetI {
    public java.lang.String[] _ids() {
        return __ids;
    }
    private static java.lang.String[] __ids = {
        "IDL:dbm/RecordSetI:1.0"
    };
    public boolean next() {
        try {
            org.omg.CORBA.portable.OutputStream _output =
this._request("next", true);
            org.omg.CORBA.portable.InputStream _input =
this._invoke(_output, null);
            boolean _result;
            _result = _input.read_boolean();

```

```

        return _result;
    }
    catch(org.omg.CORBA.TRANSIENT _exception) {
        return next();
    }
}
public java.lang.String getString(
    int col
) {
    try {
        org.omg.CORBA.portable.OutputStream _output =
this._request("getString", true);
        _output.write_long(col);
        org.omg.CORBA.portable.InputStream _input =
this._invoke(_output, null);
        java.lang.String _result;
        _result = _input.read_string();
        return _result;
    }
    catch(org.omg.CORBA.TRANSIENT _exception) {
        return getString(
            col
        );
    }
}
}
}

```

```

package dbm;
public class _tie_RecordSetI extends dbm._RecordSetIImplBase {
    private dbm.RecordSetIOperations _delegate;
    public _tie_RecordSetI(dbm.RecordSetIOperations delegate,
java.lang.String name) {
        super(name);
        this._delegate = delegate;
    }
    public _tie_RecordSetI(dbm.RecordSetIOperations delegate) {
        this._delegate = delegate;
    }
    public dbm.RecordSetIOperations _delegate() {
        return this._delegate;
    }
    public boolean next() {
        return this._delegate.next(
        );
    }
    public java.lang.String getString(

```

```

        .int col
    ) {
        return this._delegate.getString(
            col
        );
    }
}
*****
package dbm;
public interface RecordSetI extends org.omg.CORBA.Object {
    public boolean next();
    public java.lang.String getString(
        int col
    );
}
}
*****
package dbm;
abstract public class RecordSetIHelper {
    public static dbm.RecordSetI narrow(org.omg.CORBA.Object
object) {
        return narrow(object, false);
    }
    private static dbm.RecordSetI narrow(org.omg.CORBA.Object
object, boolean is_a) {
        if(object == null) {
            return null;
        }
        if(object instanceof dbm.RecordSetI) {
            return (dbm.RecordSetI) object;
        }
        if(is_a || object._is_a(id())) {
            dbm.RecordSetI result = new dbm._st_RecordSetI();
            ((org.omg.CORBA.portable.ObjectImpl) result)._set_delegate
                (((org.omg.CORBA.portable.ObjectImpl)
object)._get_delegate());
            return result;
        }
        return null;
    }
    public static dbm.RecordSetI bind(org.omg.CORBA.ORB orb) {
        return bind(orb, null, null, null);
    }
    public static dbm.RecordSetI bind(org.omg.CORBA.ORB orb,
java.lang.String name) {
        return bind(orb, name, null, null);
    }
}

```



```

    public static dbm.RecordSetI bind(org.omg.CORBA.ORB orb,
    java.lang.String name, java.lang.String host,
    org.omg.CORBA.BindOptions options) {
        return narrow(orb.bind(id(), name, host, options), true);
    }
    private static org.omg.CORBA.ORB _orb() {
        return org.omg.CORBA.ORB.init();
    }
    public static dbm.RecordSetI
    read(org.omg.CORBA.portable.InputStream _input) {
        return dbm.RecordSetIHelper.narrow(_input.read_Object(),
    true);
    }
    public static void write(org.omg.CORBA.portable.OutputStream
    _output, dbm.RecordSetI value) {
        _output.write_Object(value);
    }
    public static void insert(org.omg.CORBA.Any any,
    dbm.RecordSetI value) {
        org.omg.CORBA.portable.OutputStream output =
    any.create_output_stream();
        write(output, value);
        any.read_value(output.create_input_stream(), type());
    }
    public static dbm.RecordSetI extract(org.omg.CORBA.Any any) {
        if(!any.type().equal(type())) {
            throw new org.omg.CORBA.BAD_TYPECODE();
        }
        return read(any.create_input_stream());
    }
    private static org.omg.CORBA.TypeCode _type;
    public static org.omg.CORBA.TypeCode type() {
        if(_type == null) {
            _type = _orb().create_interface_tc(id(), "RecordSetI");
        }
        return _type;
    }
    public static java.lang.String id() {
        return "IDL:dbm/RecordSetI:1.0";
    }
}
}
*****
package dbm;
final public class RecordSetIHolder implements
org.omg.CORBA.portable.Streamable {
    public dbm.RecordSetI value;
    public RecordSetIHolder() {

```

```

}
public RecordSetIHolder(dbm.RecordSetI value) {
    this.value = value;
}
public void _read(org.omg.CORBA.portable.InputStream input) {
    value = RecordSetIHelper.read(input);
}
public void _write(org.omg.CORBA.portable.OutputStream output)
{
    RecordSetIHelper.write(output, value);
}
public org.omg.CORBA.TypeCode _type() {
    return RecordSetIHelper.type();
}
}
*****
package dbm;
public interface RecordSetIOperations {
    public boolean next();
    public java.lang.String getString(
        int col
    );
}

```

```

/***** VC++ Connectivity *****/
#include <DBM_c.h>
#include <iostream.h>
#include <conio.h>
void main(int argc, char* argv[])
{
    if (argc >= 4)
    {
        try
        {
            CORBA::ORB_ptr orb = CORBA::ORB_init(argc, argv);
            dbm::DBManager_ptr t_dbm = dbm::DBManager::_bind("dbm");
            CORBA::Long t_conID = t_dbm->openConnection(
                argv[1], argv[2], argv[3]);
            char str[100];
            cout << "Enter any Database Stmt : " << endl;
            gets(str);
            t_dbm->execute(t_conID, str);
            cout << "Enter a Query : " << endl;
            gets(str);
            dbm::RecordSet_ptr rs = t_dbm->createRecordSet(t_conID, str);
            while (rs->next())
            {
                cout << rs->getString(1) << " " << rs->getString(2) << rs->
                    getString(3) << endl;
            }
            catch (CORBA::SystemException& ex)
            {
                cout << ex << endl;
            }
            else
            {
                cout << "VCClient dsn username password" << endl;
            }
        }
    }
}

```

```
/****** Connection Monitoring *****/
```

```
import java.awt.*;  
import java.awt.event.*;  
import java.sql.*;
```

```
public class DBMMonitor extends Frame implements ActionListener {  
    private TextField txtST;  
    private TextField txtET;  
    private TextArea ta;  
    private Button btnList;  
    private Button btnExit;  
    public DBMMonitor() {  
        super("DBM Monitor");  
        addWindowListener(new WindowAdapter() {  
            public void windowClosing(WindowEvent we) {  
                dispose();  
            }  
        });  
        txtST = new TextField("yyyy-mm-dd hh:mm:ss.ns");  
        txtET = new TextField("yyyy-mm-dd hh:mm:ss.ns");  
        ta = new TextArea();  
  
        btnList = new Button("List");  
        btnExit = new Button("Exit");  
  
        btnList.addActionListener(this);  
        btnExit.addActionListener(this);  
  
        Panel p1 = new Panel();  
        p1.setLayout(new GridLayout(1, 4));  
        p1.add(new Label("Start Time"));  
        p1.add(txtST);  
        p1.add(new Label("End Time"));  
        p1.add(txtET);  
  
        Panel p2 = new Panel();  
        p2.setLayout(new GridLayout(1, 2));  
        p2.add(btnList);  
        p2.add(btnExit);  
  
        add(p1, "North");  
        add(ta);  
        add(p2, "South");  
        pack();  
    }  
}
```

```

public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == btnExit)
        dispose();

    if (ae.getSource() == btnList) {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(
                "jdbc:odbc:dbmadmin", "admin",
                "");

            Statement stmt = con.createStatement();

            String SQL = "SELECT * FROM Connections";
            ResultSet rs = stmt.executeQuery(SQL);

            Timestamp st = Timestamp.valueOf(txtST.getText());
            Timestamp et = Timestamp.valueOf(txtET.getText());

            ta.setText("conID --- DSN --- Time \n");
            while (rs.next()) {
                int conID = rs.getInt(1);
                String dsn = rs.getString(2);
                Timestamp ts =
Timestamp.valueOf(rs.getString(3));

                if ( (ts.after(st) && ts.before(et)) ||
                    ts.equals(st) || ts.equals(et)) {
                    ta.append(conID + " --- " + dsn + " --- "
+ ts.toString() + "\n");
                }
            }
            rs.close();
            stmt.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

public static void main(String[] args) {
    (new DBMMonitor()).show();
}
}

```

```
/****** Connection Monitoring Chart *****/
```

```
import com.objectplanet.chart102.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
```

```
public class FreqInputScreen extends Frame implements
ActionListener {
```

```
    private TextField f1;
    private TextField f2;
    private TextField f3;
    private TextField f4;
    private TextField f5;
    private TextField f6;
    private TextField f7;
    private TextField f8;
    private TextField f9;
    private TextField f10;
```

```
    Button btnChart;
    Button btnExit;
```

```
    private double[] noc = new double[9];
    private String barlabels[] = new String[9];
```

```
    public FreqInputScreen()
```

```
    {
        super("Connection Monitoring Chart Entryr");
```

```
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                dispose();
            }
        });
```

```
        f1 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f2 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f3 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f4 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f5 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f6 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f7 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f8 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f9 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
        f10 = new TextField("yyyy-mm-dd hh:mm:ss.ns");
```

```

setLayout(new GridLayout(6, 2));
add(f1);
add(f2);
add(f3);
add(f4);
add(f5);
add(f6);
add(f7);
add(f8);
add(f9);
add(f10);

btnChart = new Button("Show Chart");
btnExit  = new Button("Exit");

btnChart.addActionListener(this);
btnExit.addActionListener(this);

add(btnChart);
add(btnExit);

pack();
}

public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == btnExit)
        dispose();

    if (ae.getSource() == btnChart) {
        noc[0] = GetNoOfConnections(f1.getText(), f2.getText());
        noc[1] = GetNoOfConnections(f2.getText(), f3.getText());
        noc[2] = GetNoOfConnections(f3.getText(), f4.getText());
        noc[3] = GetNoOfConnections(f4.getText(), f5.getText());
        noc[4] = GetNoOfConnections(f5.getText(), f6.getText());
        noc[5] = GetNoOfConnections(f6.getText(), f7.getText());
        noc[6] = GetNoOfConnections(f7.getText(), f8.getText());
        noc[7] = GetNoOfConnections(f8.getText(), f9.getText());
        noc[8] = GetNoOfConnections(f9.getText(),
f10.getText());

        barlabels[0] = "f1";
        barlabels[1] = "f2";
        barlabels[2] = "f3";
        barlabels[3] = "f4";
        barlabels[4] = "f5";
    }
}

```

```

        barlabels[5] = "f6";
        barlabels[6] = "f7";
        barlabels[7] = "f8";
        barlabels[8] = "f9";

        ConnChart cc = new ConnChart();
        cc.show();
    }
}

private long GetNoOfConnections(String startTime,
                                String endTime) {
    int noc = 0;
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection(
            "jdbc:odbc:dbadmin", "admin", "");

        Statement stmt = con.createStatement();

        String SQL = "SELECT * FROM Connections ";
        ResultSet rs = stmt.executeQuery(SQL);

        Timestamp st = Timestamp.valueOf(startTime);
        Timestamp et = Timestamp.valueOf(endTime);

        while (rs.next())
        {
            Timestamp ts = Timestamp.valueOf(rs.getString(3));

            if ( (ts.after(st) && ts.before(et)) ||
                ts.equals(st) || ts.equals(et) )
            {
                noc++;
            }
        }

        rs.close();
        stmt.close();
        con.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    System.out.println(noc);
    return noc;
}

```



```

private class ConnChart extends Frame {
public ConnChart() {
    BarChart chart = new BarChart();

    chart.setSampleCount(noc.length);
    chart.setSampleValues(0, noc);

    chart.setBarLabels(barlabels);
    chart.setBarLabelsOn(true);

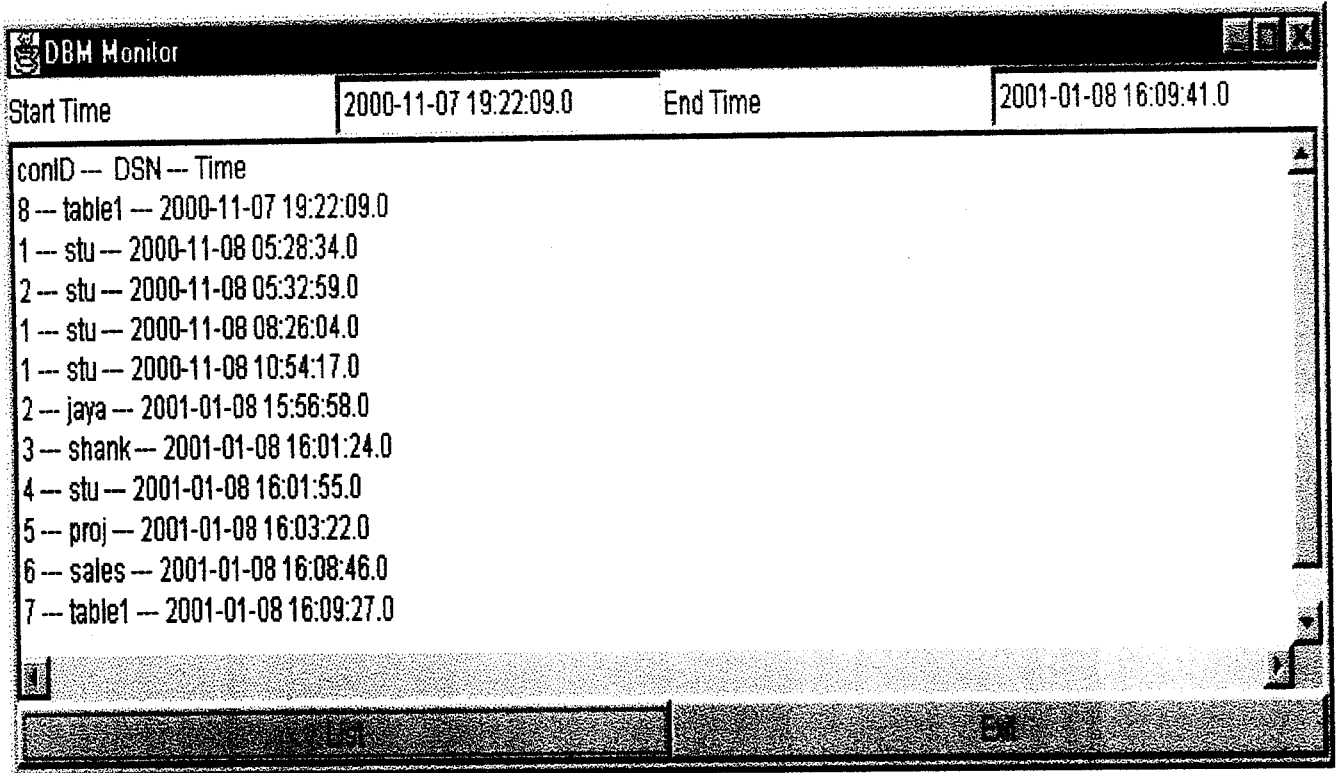
    chart.setLabel("rangeAxisLabel", "No of Connections");
    chart.setLabel("sampleAxisLabel", "Time");

    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent we) {
            dispose();
        }
    });
    add(chart);
    pack();
}
}

public static void main(String[] args) {
    FreqInputScreen fis = new FreqInputScreen();
    fis.show();
}
}

```

Connection Monitoring



The screenshot shows a window titled "DBM Monitor" with a list of connections. The window has a title bar with standard minimize, maximize, and close buttons. Below the title bar, there are two fields: "Start Time" and "End Time". The "Start Time" field contains "2000-11-07 19:22:09.0" and the "End Time" field contains "2001-01-08 16:09:41.0". The main area of the window contains a list of connections, each with a "conID", a "DSN", and a "Time". The list is as follows:

conID	DSN	Time
8	table1	2000-11-07 19:22:09.0
1	stu	2000-11-08 05:28:34.0
2	stu	2000-11-08 05:32:59.0
1	stu	2000-11-08 08:26:04.0
1	stu	2000-11-08 10:54:17.0
2	jaya	2001-01-08 15:56:58.0
3	shank	2001-01-08 16:01:24.0
4	stu	2001-01-08 16:01:55.0
5	proj	2001-01-08 16:03:22.0
6	sales	2001-01-08 16:08:46.0
7	table1	2001-01-08 16:09:27.0

At the bottom of the window, there are two buttons: "DB" and "EM".

Connection Monitoring Time Interval

Connection Monitoring Time Interval	
2000-11-07 08:18:06.0	2000-11-07 15:56:42.0
2000-11-07 19:12:24.0	2000-11-08 19:20:09.0
2000-11-09 20:50:14.0	2001-01-08 17:29:34.0
2001-01-08 17:56:58.0	2001-01-08 21:09:24.0
2001-01-09 15:24:14.0	2001-01-09 16:09:24.0
Show Chart	Exit

Connection Monitoring Bar Chart

