PIMAQS: Platform Independent Multidatabase Administration and Query System

Joakim Johansson (joakim@acm.org)
Le Gruenwald (gruenwal@cs.ou.edu)
Yumiko Yokoyama(yumiko@ou.edu)

The University of Oklahoma
School of Computer Science
200 Felgar
Norman, OK 73019-6151

Kevin Livingston (livingkm@flyernet.udayton.edu)

The University of Dayton
Computer Science
300 College Park
Dayton, OH 45469-1410

ABSTRACT

In this age of networking, finding good ways to integrate decentralized information has become more important than ever. One way to integrate multiple database systems is through a multidatabase system. This paper describes a platform independent, Web deployed, scalable multidatabase system prototype called PIMAQS, which was implemented using Java, JDBC and Java Foundation Classes. Java is a promising new technology that seems well suited for programs that require database connectivity and graphical user interfaces.

1 INTRODUCTION

As enterprises and organizations incorporate new technologies and geographically diversify their operations, it is more likely that their database management systems (DBMS) consist of many local database management systems (LDBMS) operating in heterogeneous software and hardware environments. Each LDBMS in the enterprise DBMS may be designed and administered independently, maintaining local autonomy. One solution for a unified interface to such DBMS environment is a multidatabase system (MDBS), a software layer that integrates multiple *heterogeneous* and *autonomous* LDBMSs [Silberschatz, 1997]. Another important characteristic in MDBS is *transparency* that is the user of MDBS is unaware of accessing different member database systems and its workings [Leong-Hong, 1982].

In an MDBS environment, schemas of member LDBMSs are integrated into a global schema. Using the global schema, the users of MDBS can access any member LDBMS as if it is a single database. For example, the global schema might contain a table called Employees, which actually refers to several tables with different names in the LDBMSs. Users are able to query the Employee table, and get the joined results from the LDBMSs. The MDBS is responsible for creating the subqueries needed to get the data from LDBMSs, and for handling type conversions and name mappings.

This paper describes Platform Independent Multidatabase Administration and Query System (PIMAQS), a prototype for user-friendly, platform independent MDBS with Web-access capability. PIMAQS is developed in Java, thus providing platform independence wherever a Java runtime environment is available. The user interface can be run as a Java applet in a Web browser; thus, PIMAQs is Internet ready. As the paper will show, the user side is a thin-client program that allows the user to utilize the MDBS.

PIMAQS is an example of how Java and Object-Oriented programming environment can be used to create a customized multidatabase solution rather quickly and cheaply. Java is gaining popularity because of one of its prominent features, that is, platform independence. Java's "Write once, run anywhere" functionality is an appealing quality in a programming language. In the last few years, the network has become more and more critical to operations. The creators of Java recognized this trend and made Java's networking capabilities very strong. In addition to outstanding networking, Java provides database connectivity via JDBC, a set of classes that are part of the standard Java language. JDBC allows Java programs to connect to databases on an Intranet or on the Internet. Finally, Java provides graphical capabilities that make it very suitable for development of GUIs (Graphical User Interfaces). Given these desirable qualities, Java appears to be an ideal language to use for a piece of software like PIMAQS.

The overall design of PIMAQS is described in Section 2. Sections 3 discusses how the major components of the prototype were implemented. Annotated screenshots of PIMAQS are found in Section 4. Impressions and observations from the prototype development process are documented in Section 5. Section 6 contains conclusions about using Java to connect to databases and Java prototyping as well as the future plans for PIMAQS and other related projects.

2 DESIGN

In designing a prototype, first we have made assumptions and boundaries as described in Section 2.1. Based on these boundaries and assumptions, we created a process flow describing the sequence of events that take place in using PIMAQS, which will be discussed in Section 2.2. Finally, the system architecture was created to meet the demands of the process flow and the expected functionality (See Section 2.3).

2.1 Assumptions

Because a complete multidatabase system is very complex, the current prototype of PIMAQS is a specialized version. It provides network transparency to the end-user, but queries must be pre-parsed by a global database administrator (GDBA) who is aware of the global schema. The GDBA is responsible for creating the set of atomic subqueries that constitutes a multidatabase query. As the query base grows, requested queries are more and more likely to already exist in the system. If a user requests a query that already exists, the DBA simply grants the user access to the existing one. End users never have to write SQL statements or know where the data is stored. PIMAQS may best suit small operations where a limited number of non-technical users need access to a relatively static set of queries.

PIMAQS is capable of read-only and update queries. However, because of LDBMS autonomy, it is difficult to guarantee global serialization. Local queries are not scheduled globally, so they may conflict with global subqueries. The current prototype includes an implementation of a *ticket method* that should be able to ensure global serialization [Georgakopoulos, 1993]. The ticket method implementation is currently not tested, so for the purposes of this paper, all queries can be assumed to be read-only and they are not guaranteed to be globally serialized.

2.2 Process flow

At startup, PIMAQS goes through the setup phase as shown in Figure 1. When client applet is loaded, it prompts the user for authentication information, which is sent to the application server. The application server then checks against the support database for the user authentication, and returns the result to the client. Upon successful authentication, the application server extracts the user's configuration information from the support database and passes it on to the client. The client uses this information to configure the graphical user interface (GUI).

The query phase as shown in Figure 2 follows the setup phase. By using the client applet, the user initiates a query, say, with query identification, Q. The query Q may require some inputs in forms of variable arguments from the user. The client applet collects them and sends the query's identification Q as well as the user-defined arguments for Q to the application server. From the support database, the application server finds that query Q consists of subqueries q1, q2, …, q*n*. A subquery is an atomic query that is only valid at a given LDBMS. A temporary table that will hold the intermediate data is created in the support database. All subqueries are fetched from the support database and consequently submitted to the respective LBDMS and executed locally at the LDBMS. The data from q1 and q*i* are inserted into the temporary table. Finally, the query to extract the final result, q*n*, is performed on the temporary table in the support database. The resulting data set is sent from the server to the client program. The client program displays the output in a tabular format, and is ready for a next query.

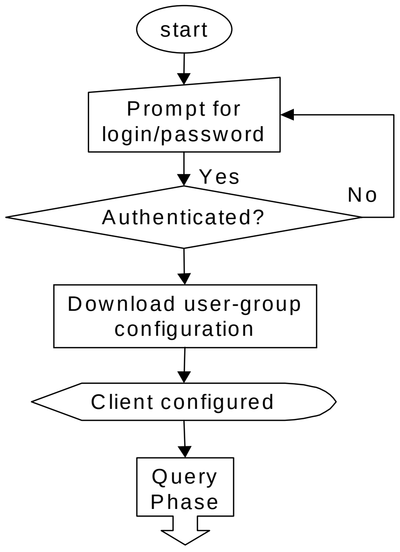


Figure 1: Setup phase process flow

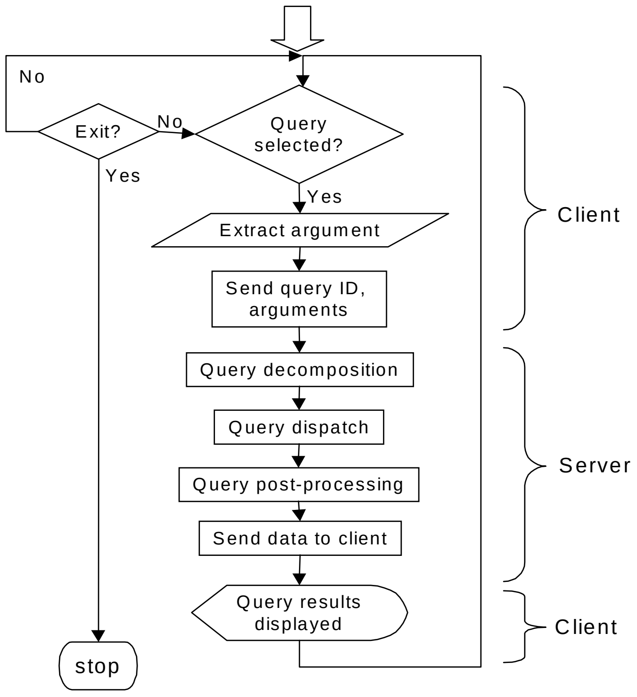


Figure 2: Query phase process flow

2.3 System Architecture

PIMAQS is a three-tier architecture consisting of Client Tier, Application Server Tier, and Local Database System Tier, as shown in Figure 3. The Client Tier is made up of small client programs running on networked computers. The Local Database System Tier is composed of multiple autonomous and heterogeneous databases on the Intranet or Internet. The Application Server Tier consists of a text-based Java server program and a JDBC-accessible support database.

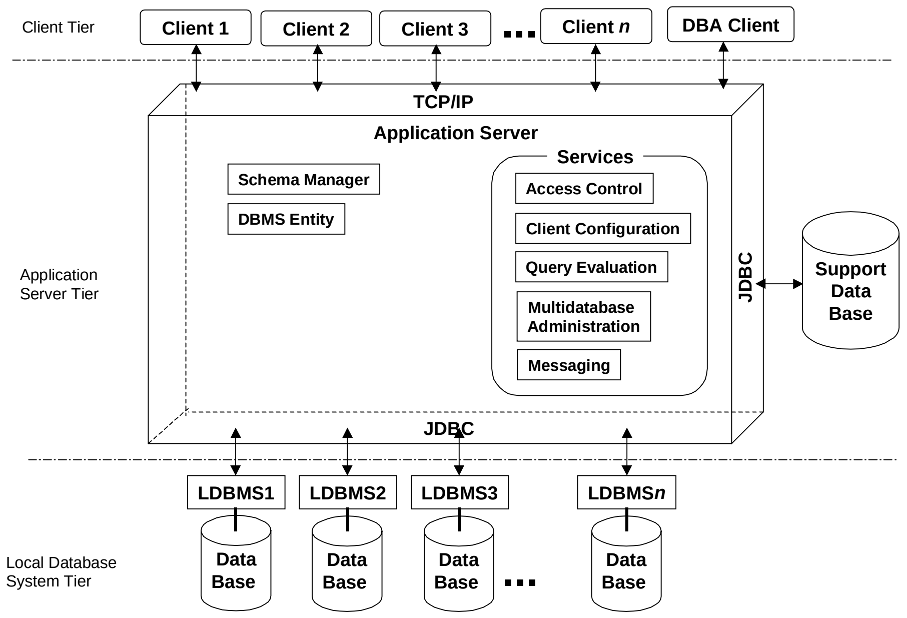


Figure 3 : PIMAQS Architecture

2.3.1 Client Tier

The client programs are Java applets running in Web browsers. They are downloaded over the Web, and the GUI is created based on configuration obtained from the application server tier during the setup phase. The client program is quite simple and small, since most of the data processing is done in the middle tier.

2.3.2 Application Server Tier

The middle tier of PIMAQS hosts the application server. It is the heart of the MDBS, and it does most of the work. The server accepts connections from clients, serves them configuration information and processes queries. The client-server connections use TCP/IP, and the server communicates with the LDBMSs using JDBC and database specific driver classes. The application server also uses JDBC to communicate with the support database system. To ease the burden on the application server, our design uses a support database to store configuration data and to perform database operations. The application server contains three components: Schema Manager, DBMS Entity and Services.

Schema Manager – Because the member databases are autonomous, independent, it is possible that the local schema changes without the MDBS being notified. Therefore the MDBS has a schema manager that is responsible for periodically checking the member databases for schema updates, modifications and availability. If the local schema changes, the global schema must be updated to reflect the changes. Changes could be that tables are dropped or renamed, attributes are added to, or removed from, a table, or that the database is completely offline. These changes may make some queries unavailable. A query is unavailable if one or more subqueries can not be executed.

DBMS Entity – This is an encapsulation of a JDBC connection between the application server and a DBMS.

Services – The application server offers the following services to clients:

- **Access Control**: authenticates users; authentication is required for access to PIMAQS.
- **Client Configuration**: sends the configuration information the client in the setup phase.
- **Query Evaluation**: evaluates queries requested by clients
- **Multidatabase Administration**: provides administrative functions to DBA users
- **Messaging**: provides basic messaging capabilities to users

2.3.3 Local Database System Tier

Any JDBC compliant DBMS system should be able to be inserted into the Local Database System Tier. For example, student databases at the University of Oklahoma and Oklahoma State University are examples of autonomous and heterogeneous local databases that theoretically could be integrated into a MDBS using PIMAQS.

3 IMPLEMENTATION

Prototype development requires access to hardware and software resources. The implementation of PIMAQS did not put high demands on hardware resources, but rather it relied on existing Java technologies for its power. More specifically, to access the databases we used JDBC, a standard Java package with industrial strength database connectivity. Java's multithreading capabilities were used in creating an application server program that could handle multiple clients simultaneously. Java Foundation Classes (JFC) with its Swing package is a recent addition to standard Java that we used to create the graphical user interface for the client program [Andrews, 1999]. The client program and the application server within the three-tier architecture were written entirely in Java. We used a support database to provide bookkeeping functions and persistent storage in the application server tier. The implementation also required us to create sample databases to be used in the local database system tier.

3.1 Resources

Development and testing were done on hardware that would probably be considered average today in terms of performance. The University of Oklahoma Engineering Computer Network and the Computer Science department provided the DBMSs used for development and testing. All other software used was downloaded free of charge from their respective vendors. Figure 4 summarizes the hardware, software and DBMSs used.

Hardware	**Software**	**DBMSs**
PII 266MHz, 64Mb, WinNT 4.0 PII 400MHz, 128Mb, Win98 UltraSPARC 20, Solaris 7	JDK 1.2.2, Win98/NT JDK 1.1.6, Solaris Java Plug-in 1.2.2 Java Plug-in 1.2 HTML Converter Microsoft Personal Web Server	Oracle 8.0.4 MySQL 9.29

Figure 4: Hardware and Software used to create PIMAQS prototype

3.2 Database connectivity

Java's database connectivity package JDBC was introduced in 1996 [Graham, 1997]. It has been a standard JDK component since version 1.1. The current version of JDBC is v2.0. JDBC provides classes that allow Java programs to query databases across an Intranet or the Internet. To be able to query a database using JDBC, a DBMS-specific driver is required on the client machine. Most DBMS vendors have made "pure Java" drivers available, and that means that the driver can be downloaded together with the Java applet itself. This type of driver is especially well suited for Web deployment since no driver installation is required on the client machine.

3.3 Multithreading

Java supports multithreading, a feature that is extremely useful in a server program which is likely to be handing several connections simultaneously. As clients connect to the server, new threads can be created to handle the new connection. That means that a client does not have to wait for the current transaction to be finished before the server can process the new request. Java's Thread class has all the methods needed to create multithreaded applications that are not overly complex to understand. We based our server on an example found in O'Reilly's "Java By Example In a Nutshell", and none of the small tests that we ran revealed any performance problems [Flanagan, 1997]. In fact, the simple server program was very robust and the threading appeared to work well.

3.4 Java Foundation Classes (JFCs) and Swing

The Swing components, which are a part of JFC (Java Foundation Classes), proved very easy to work with [Andrews, 1999]. With Swing, Sun introduced a very appealing Look-And-Feel framework, which makes it extremely easy to change the surface features of the graphical user interface. For example, switching between a Windows, Java (“Metal”), and Motif look is as simple as calling a single method. It provides many useful classes like tabbed panes, tables, sliders etc. Swing also makes it very easy to add “Tool Tips” to GUI components. Tool Tips are the little help boxes that are displayed when the mouse pointer hovers over a GUI component. Because data from a database is frequently displayed in a tabular format, Swing’s JTable class proved invaluable. Sorting a table of data by an arbitrary column was easily added by using Sun’s Swing examples. In other words, Swing makes it quite easy to add the GUI functionality that we all have grown accustomed to in many Windows-based applications.

3.5 Client Tier implementation

The client program is a Java applet running in a Web browser. We used the Java Foundation Classes (JFC) when developing the GUI. The graphics package in JFC is called Swing. Swing provides several new powerful GUI components that work well, look good, and are easy to use. In particular, the JTabbedPane and JTable classes were extremely useful in creating our simple client interface. Tabbed panes make it very easy to quickly switch between interface views, similar to using the tabs in a notebook. The JTable class represents a table of data, similar to the Microsoft Excel spreadsheet style. Sun’s TableSorter class was used to add sort-by-column functionality to the tables.

When the applet is loaded in the browser, it establishes a TCP/IP connection to the application server running on the same physical machine as the Web server. Once it is connected the user enters a login and password in a Login Dialog box. The authentication is then sent to the application server for verification. On successful login, the client requests configuration information from the server. The configuration data is used to create dynamically labeled and structured Query Panels, Messaging Panels etc. A panel can be thought of as a blank sheet to which GUI components can be added. The panels are added to a tabbed pane for convenient random access. The output is tabular data, and to enable the user to easily compare the results from multiple queries, new window is created for output tables. Refer to Section 4 section for screenshots of the client interface and output tables.

Login Dialog Box – The login dialog consists of a JTextField, a JPassword field, and a JButton.

Query Panels – The query panels are displayed to the user inside the MDB Client applet. Each panel displays the queries that are available for a workgroup. If the user is a member of three workgroups, he or she will be given three Query Panel tabs (identified by workgroup name) in the client’s tabbed pane. The user issues queries by pressing buttons in the panel.

Query Entries – A query entry consists of a button used to execute the query, a textual description and possible text entry boxes that hold variables. The QueryEntry class was implemented as a subclass of JPanel. QueryEntries are added to the query panels in the same way a regular JButton or JTextField are. When the button is pressed, the getParameter() method returns a string containing the query id and the variables, if any.

Messaging Panel – The messaging panel is created as a panel in the tabbed pane during the setup phase. Users select the recipient using a JComboBox component. JComboBoxes lets the user select from a list of options, but only one value can be selected at a time. With this simple messaging facility, users can exchange text messages, make announcements, make requests for additional queries, ask questions etc. When the client program is in the setup stage, the application server checks for messages left on the support server for that user. If messages exist on the server, they are forwarded to the client. The user sends messages by selecting the recipient (e.g. the DBA, a group, or a normal user), typing in a message in the text area and then pressing the “send” button.

3.6 Application Server Tier implementation

The main components in the application server are the Schema Manager and DBMS Entities. DBMS Entities are objects that are created and discarded on-demand, whereas the Schema Manager is a thread that keeps running until the application is terminated. The access control and client configuration services were implemented in a common class. Query evaluation, multidatabase administration and messaging services are separate entities in the prototype server.

Schema Manager – The Schema Manager is a thread in the server that establishes a JDBC connection to the LDBMSs evey *n* seconds to verify that they are online. It also checks whether all the tables and attributes used in the subqueries are available. If a subquery can not be executed, neither can the query in its entirety. The query is therefore marked as unavailable in the support database, and the query entry is deactivated, "grayed out", in the client graphical user interface (GUI).

DBMS Entity – JDBC connections between the application server and databases are encapsulated in DBMSEntity class objects. When DBMSEntity objects are created using the database URL (Uniform Resource Locator), login and password, the appropriate "pure Java" (Type 4) JDBC driver is registered with Java using the Driver Manager class. The connect() method establishes a connection to the LDBMS. An example of a simple query procedure using a DBMS Entity is shown below.

```
// Create a JDBC connection to a LDBMS and connect it
DBMSEntity db = new DBMSEntity(url, login, pass);
db.connect();
Connection connection = db.getConnection();

// Create a Statement for the Connection
Statement statement = connection.createStatement();

// Execute the SQL query and put the results in a ResultSet
ResultSet results = statement.executeQuery(sqlString);
```

Support Database

PIMAQS uses a JDBC accessible Oracle database for its support database. It is used extensively by the application server for authentication, data integration, and client configuration. The entities in the prototype database are users, groups, queries, subqueries and databases.

Users of the system are uniquely identified by a login; other attributes are password, user ID, name and description. They are members of one or more groups. **Groups** are identified by a group id. Each group also has a name and a description. An example is a group with ID 100, name "dba" and description "Database Administrator group". Groups have access to a set of queries. **Queries** have unique query IDs, a description of the query, number of subqueries and arguments, and a flag that indicates whether the query is executable or not. The queries correspond to a set of subqueries. **Subqueries** are identified by subquery IDs. Subqueries also have an SQL query string. Subqueries are atomic and are only directed at a particular database. Finally, d**atabases** are identified by a database ID. Database tuples also contain a URL, a type and a login/password pair. Figure 5 shows an ER diagram for the support database.

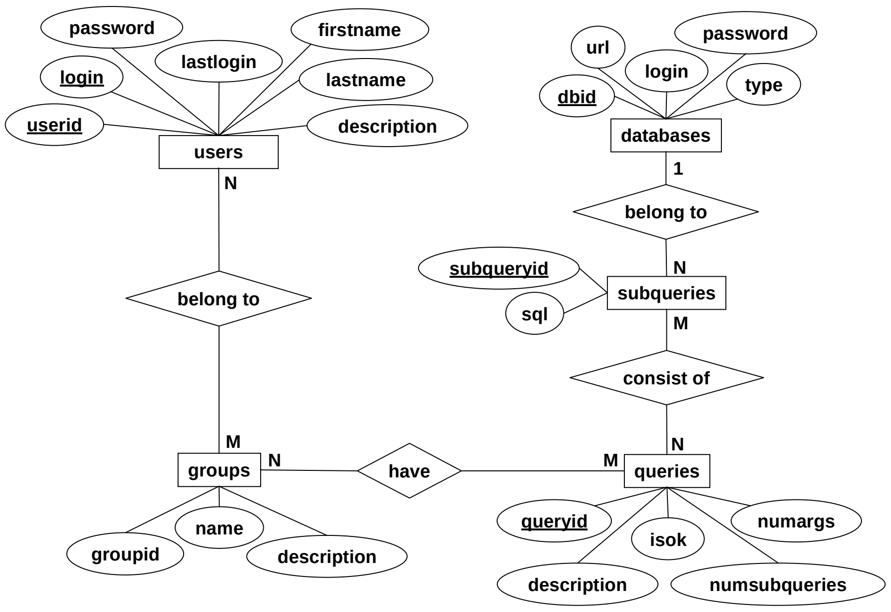


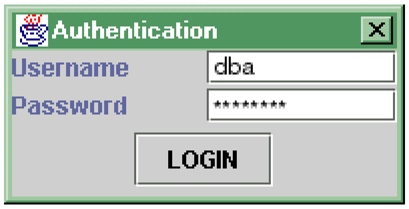
Figure 5: Support Database System ER Diagram

Local Database System Tier

We used three databases to test the system. Two databases were located on an Oracle 8.0.4 system, and one was a MySQL database. Each database contained three to four tables and some test data. A real-world scenario might include the entire DBMS in the multidatabase, but our prototype only included specific databases in each DBMS.

4 RESULTS

The following screen captures of the client program show the authentication dialog box, a query panel, a messaging panel and an output window. To save screen real estate, PIMAQS was run as an application rather than an applet for the purpose of obtaining screen captures. The difference is minor; instead of letting the browser act as the container; a frame is created that performs the same function.



Username dba
Password ********

Figure 6: User authentication dialog box. The user must authenticate using the authentication dialog box before he or she is allowed to continue using PIMAQS. If the login and password are incorrect, the user must try again.

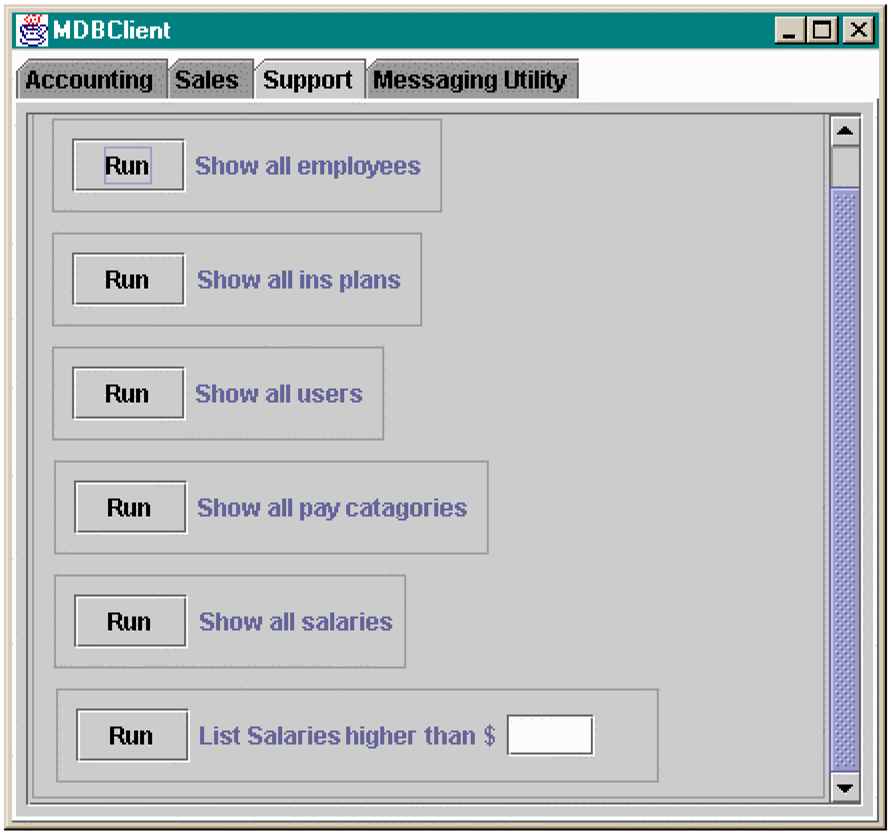
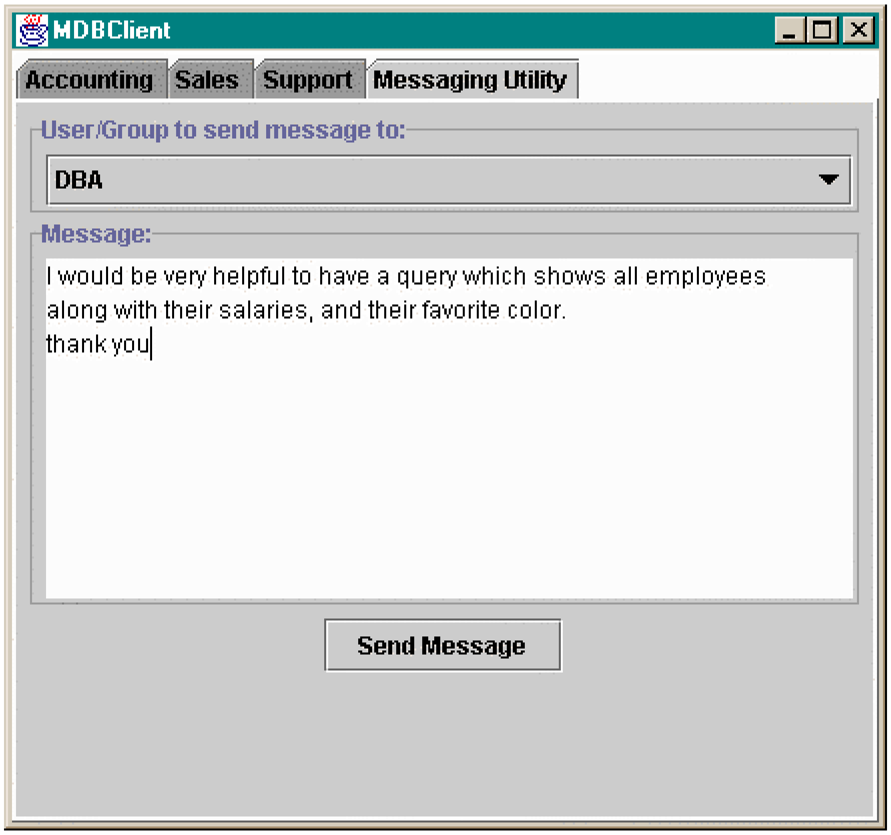
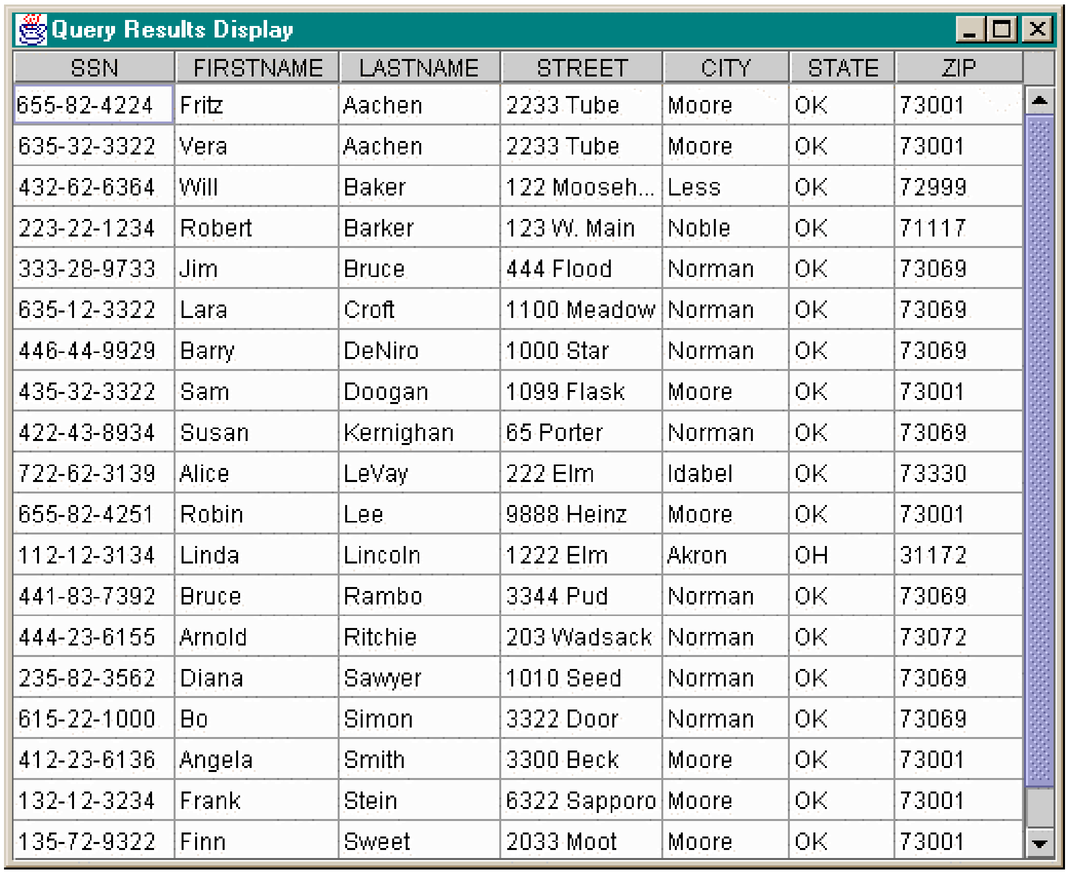


Figure 7: Sample view of post-configuration query panel. The configuration information that is stored in the support database determines what panels are created and what queries are made available in the query panels at configuration time. One query panel is created for each group that the user is a member of. In this case, the "Support" group's query panel is visible. The last query in the panel requires input from the user before it can be executed.



I would be very helpful to have a query which shows all employees along with their salaries, and their favorite color.
thank you

Figure 8: A Messaging Utility panel. All users automatically get the Messaging Utility panel. With this tool, the user can send simple text messages to other users or groups. The DBA is also a user; here we can see a message for the DBA requesting that a particular query is added.



SSN	FIRSTNAME	LASTNAME	STREET	CITY	STATE	ZIP
655-82-4224	Fritz	Aachen	2233 Tube	Moore	OK	73001
635-32-3322	Vera	Aachen	2233 Tube	Moore	OK	73001
432-62-6364	Will	Baker	122 Mooseh...	Less	OK	72999
223-22-1234	Robert	Barker	123 W. Main	Noble	OK	71117
333-28-9733	Jim	Bruce	444 Flood	Norman	OK	73069
635-12-3322	Lara	Croft	1100 Meadow	Norman	OK	73069
446-44-9929	Barry	DeNiro	1000 Star	Norman	OK	73069
435-32-3322	Sam	Doogan	1099 Flask	Moore	OK	73001
422-43-8934	Susan	Kernighan	65 Porter	Norman	OK	73069
722-62-3139	Alice	LeVay	222 Elm	Idabel	OK	73330
655-82-4251	Robin	Lee	9888 Heinz	Moore	OK	73001
112-12-3134	Linda	Lincoln	1222 Elm	Akron	OH	31172
441-83-7392	Bruce	Rambo	3344 Pud	Norman	OK	73069
444-23-6155	Arnold	Ritchie	203 Wadsack	Norman	OK	73072
235-82-3562	Diana	Sawyer	1010 Seed	Norman	OK	73069
615-22-1000	Bo	Simon	3322 Door	Norman	OK	73069
412-23-6136	Angela	Smith	3300 Beck	Moore	OK	73001
132-12-3234	Frank	Stein	6322 Sapporo	Moore	OK	73001
135-72-9322	Finn	Sweet	2033 Moot	Moore	OK	73001

Figure 9: An output window with the data sorted by LASTNAME column. The query results are presented in like this. Although this picture can not illustrate it, the columns can be reordered via a simple click-and-drag motion. The user can also sort the records by clicking on the column headings. This picture really demonstrates the power of Swing.

5 OBSERVATIONS

Prototyping is exciting and rewarding, but often frustrating along the way. This section discusses some problems that we encountered and how we solved or avoided them.

Plug-in required for to run applets that use Java 1.1 features

Even the most recent web browsers do not yet support all standard Java 1.1+ features. To be able to run an applet that uses JFC/Swing or the Java 1.1 event-handling model, a one-time installation of a browser plug-in is required. The Web page that loads the applet also has to be processed using a conversion program. The converted Web page instructs the browser to use the Java 1.1 compliant virtual machine when running the applet. Sun's Java homepage provides both programs, free of charge. Future browsers are likely to be Java 1.1 compliant - the plug-in represents an acceptable interim solution to this problem.

Applet security requires that the application server resides on the same machine as the Web server

Malicious Java applets could, if it was not for Java's security features, exploit the good faith of people who run them to destroy, modify, or steal data stored on the local machine.

Applets are therefore not allowed to access the local file system, or to create network connections to sites other than the Web server that the applet was downloaded from. Because PIMAQS's application server runs on the same machine as the Web server, this Java security feature does not hinder it. Although this paper does not discuss it, Java applets can be given more freedom if they go through a digital signing process. Signed applets can be trusted, since the person running it can be certain that the signer is who they claim to be. VeriSign is one such digital signature provider.

JDBC ResultSet limitations

Prior to the release of JDBC 2.0, the ResultSet was arguably very limited in what it could do. The only way to extract the data was to go through it sequentially with the next() method:

```
while( result.next() )
{  // Get the first value as a String
   String data = result.getString(1);
}
```

That is how we did it, and it worked well for the things we wanted to do in PIMAQS. However, this shortcoming has been addressed, and since JDBC 2.0 it is possible to find the current cursor position, to move backwards and to move to any absolute or relative position in the ResultSet. Although that was a great improvement, some facility to join two ResultSets would have been quite useful as well.

JDBC Driver Inconsistencies

Our test environment only used two different types of DBMSs; Oracle and MySQL. Although this worked well, the instructions on how to load the JDBC driver for these two systems varied. The Oracle driver used the DriverManager.registerDriver method, whereas the MySQL driver added the driver class to the system Properties hash table. This is a minor issue, but it nonetheless has implications for implementations because not all drivers can be loaded the same way. We suspect that this problem will go away in the near future, if it has not already done so in more recent driver releases. In PIMAQS, this problem was resolved by creating a specialized DBMSEntity class for MySQL databases.

6 CONCLUSIONS AND FUTURE WORK

Based on our experiences with PIMAQS, we have concluded that Java is a viable technology for this type of project. Not only is it quicker to prototype database applications using Java, but it is also less frustrating, more accurate and easier to use than C or C++. Shortcomings in the massively popular C and C++ were purposely addressed in the development of Java [Graham, 1997]. Garbage collection was added to resolve the memory management problems, and the error prone pointers of C/C++ were redone in Java. Because Java in an interpreted language, performance may still be lagging behind C and C++, but more tests will be required to ascertain whether PIMAQS suffers from a speed problem.

Object Oriented programming can in many cases greatly facilitate software development. It encourages modular design, encapsulation and information hiding, all three important concepts. Code reuse is key, and well-written Java classes are likely to be useful again and again. A system like PIMAQS could probably be implemented in a procedural language like C, using ProC or something else for the database connectivity and a Web interface via Common Gateway Interface (CGI). In fact, similar systems almost certainly already exist. With a language like Java available, future systems are likely to be more efficient, easier to create and update, more user friendly and cheaper than before.

The current PIMAQS prototype makes extensive use of the support database. Some data that is accessed very frequently, for example the queries, could probably be accessed more efficiently in a data structure within the application server. Another alternative is to store some of the frequently accessed data in local files on the application server machine. We would like to investigate whether either of these modifications renders substantial performance increases over the current implementation. Because the network connection between the application server and the support server might be the fastest link in the whole system, it is possible that the current solution is not significantly slower.

Binary Large Objects (BLOBS) is a datatype currently supported by many DBMSs. During the development of PIMAQS, the question whether serialized Java objects could be compressed and stored in a database came up. If it does work, we should be possible to “freeze” a client interface, or any other Java object, compress it using Java’s built-in zip compression, and then store it as a BLOB in the support database. When the user returns, his or her personalized client is downloaded and restored to the exact state it was stored in last time. Some DBMSs apparently have support for storing Java objects at this time; our proposed solution would not require such a feature, just support for the BLOB data type.

A combination of Java, JDBC and the new Java Media Framework (JMF) 2.0 might be a great way to query and display multimedia data. JMF has support for most multimedia data formats, including the popular mp3 format. It would be interesting to prototype a version of PIMAQS that is able to handle both audio and video.

Finally, we recognize that the user might want to use the data once it has been extracted from the MDBS. New versions of the client GUI would have support for cut-and-paste operations, or at least some other facility to import data into another application like MS Excel. JFC does provide the framework needed to implement it in the future.

7 REFERENCES

Silberschatz, A., H. F. Korth, and S. Sudarshan, Chapter 18, Database System Concepts, 3rd Ed., Boston: WCB/McGraw-Hill, 1997.

[Leong-Hong, 1982] Leong-Hong, B., and B. K. Plagman, Data Dictionary/Directory Systems Administration, Implementation and Usage, John Wiley & Sons, Inc., 1982

[Georgakopoulos, 1993] Georgakopoulos, D., Rusinkiewicz, M., and Sheth, A. P., “Using Tickets to Enforce the Serializability of Multidatabase Transactions”, IEEE Transactions On Knowledge and Data Engineering, 1993, Vol XX, No. Y, December 1993.
[
Andrews,1999] Andrews, Mark. “All About JFC and Swing”, http://java.sun.com/products/jfc/tsc/what_is_swing/what_is_swing.html, Aug 1999.

[Graham, 1997] Hamilton, Graham, Rick Cattell, Maydene Fisher. JDBC Database Access with Java Tutorial and Annotated Reference, Addison-Wesley/ Sun Microsystems, Inc., 1997.

[Flanagan, 1997] Flanagan, David. Java Examples in a Nutshell : A Tutorial Companion to Java in a Nutshell, O’Reilly & Associates, Inc., 1997.

Web Links

JDK, Java plugin, Java Documentation:	http://java.sun.com
Oracle JDBC Drivers:	http://technet.oracle.com/software/download.htm
MySQL JDBC Drivers:	http://www.worldserver.com/mm.mysql/
Microsoft Personal Webserver	http://www.microsoft.com