83796

A REAL APPLICATION FOR QUERYING, ORDERING AND MONITORING REAL-TIME DATA ON THE INTERNET

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of
Dokuz Eylül University
In Partial Fulfillment of the Requirements for
the Degree of Master of Science in Computer Engineering, Computer Program

by
Nail Erdem BALAN

February, 1999
İZMİR

83796

M.Sc THESIS EXAMINATION RESULT FORM

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc.Prof.Dr.Alp Kut
(Advisor)

[illegible]

Asst.Prof.Dr.Yalçın ÇEBİ
(Committee Member)

[illegible]

Asst.Prof.Dr.Adil ALPKOÇAK
(Committee Member)

Approved by the
Graduate School of Natural and Applied Sciences

[illegible]

Prof. Dr. Cahit Helvacı
Director

ACKNOWLEDGMENTS

I wish to thank Alp KUT, advisor of this thesis, whose very insightful comments helped me to shape this study. I'm very grateful for his advise and suggestions in all phases of this study.

A special acknowledgment goes to Ali Yildiz for his hardware and software support to my work. I extend a special thanks to Bora Kumova for his feedback and comments.

And I would also like to thank all of the individuals who work to develop the Internet, "Information Superhighway".

Nail Erdem Balan

ABSTRACT

To be competitive in the 1990's, organizations need to access accurate information very quickly, regardless of the data source, platform or location. The growth of the Internet, especially World Wide Web, allows organizations to access any information easily.

Because of incompatible hardware architectures and operating systems, software developers need platform independent tools to solve the compatibility problem and make applications work in a distributed client-server environment. Java is designed to meet the challenges of application development in the context of heterogeneous, network-wide distributed environments. Java applications are portable across multiple platforms and this portability allows the same Java program to run on multiple operating systems and hardware architectures without modification.

The Java Database Connectivity (JDBC) API is a specification by which Java application developers can access many different kinds of computer database systems, regardless of their location and platform. JDBC provides Java programmers a powerful API that is consistent with the rest of the Java language specification.

In this study, we first explained the background information necessary to develop a Java application and then built up a real one. The main functions of our application are ordering product, called "electronic commerce", querying data from a database, and monitoring real time data gathered from production systems. All data was stored in two databases. In order to make connection with control systems and get real time data from production floor, we used a Supervisory Control and Data Acquisition (SCADA) software and sent collected data to SQL server.

ÖZET

1990'lı yıllarda, organizasyonların rekabet edebilmesi için doğru bilgiye, bu bilgi hangi kaynakta, ortamda ve coğrafyada olursa olsun, cok hızlı biçimde ulaşması gereklidir. Internet'in, özellikle de World Wide Web (WWW)'in gelişmesi, organizasyonlara her türlü bilgiye kolayca ulaşma imkanı vermiştir.

Uyumsuz donanim yapıları ve işletim sistemleri nedeniyle, yazılım geliştiriciler uyum problemini çözmek ve uygulamalarını dağıtılmış istemci-sunucu ortamlarında çalıştırabilmek için çalısma ortamından bağımsız araçlara gereksinim duymaktadırlar. Java homojen olmayan dağıtılmış ağ ortamlarında uygulama geliştirme için gerekli ihtiyaçları karşılamak için tasarlanmıştır. Java uygulamaları farklı ortamlar arasında taşınabilir ve bu taşınabilirlik özelliği aynı Java programının farklı işletim sistemlerinde ve farklı donanım mimarilerinde hiçbir değişiklik yapılmadan çalışmasını sağlar.

Java Database Connectivity (JDBC) API, Java uygulama geliştiricilerin yeri ve ortamı ne olursa olsun birçok değişik veri tabanına erişebilmesi için gerekli kurallar bütünüdür. JDBC'nin Java programcılarına sunduğu arabirim Java programlama dilinin özellikleriyle tamamen uyumludur.

Bu çalışmada ilk olarak bir Java uygulaması geliştirebilmek için gerekli temel bilgiler açıklandı ve daha sonra gerçek bir uygulama yapıldı. Uygulamanın temel fonksiyonları, sipariş verme, uzaktaki bir veri tabanından sorgulama ve üretim sahasındaki verilerin gerçek zamanlı olarak gösterilmesi olarak belirlendi.. Bu amaçla kullanılan bütün veriler iki veri tabanında toplandı. Üretim sahasındaki

kontrol sistemleriyle haberleşmeyi sağlamak ve gerçek zamanlı verileri SQL sunucuya göndermek için de bir SCADA yazılımı kullanıldı.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE
INTRODUCTION

1.1. Introduction

Educational and corporate organizations continue to amass large amounts of data, and that data is stored in a Database Management System (DBMS). The DBMS is installed, and it includes many utilities to act upon the database, for administration, application development, and user interaction. As more data appeared, more DBMS's were purchased. Programmers were forced to develop different applications for each DBMS, since each system had different protocols, Application Programming Interfaces (API), and data structures.

To be competitive in the 1990's, organizations need to access accurate information very quickly, regardless of the data source, platform or location. Management put the pressure on in-house developers to invent database clients for each employee's desktop machine that can find all the company's information from one application. (Visigenic, 1996)

The answer to this call was Open Database Connectivity (ODBC). Developed by Microsoft and based on the Call Level Interface specification of the SQL Access Group, ODBC allows users to access data in heterogeneous environments of relational and non-relational databases. Developers, using the ODBC API, can write one application to access data on many vendors' DBMS's on many platforms, including Windows, Macintosh, and UNIX. (Intersolv, 1996)

In 1995, Sun Microsystems released an alpha 3 version of its new object-oriented language, Java. This general purpose programming language became very popular around the Internet, mostly because of its platform-independent bytecode interpretation. A Java application is compiled into bytecodes, then interpreted on a client machine by a Java interpreter for that particular architecture (Gosling & McGilton, 1995).

The distributed, networking nature of Java quickly found the new language paired with client/server database development. In an effort to develop Java applications for use with remote DBMS's, the Java Database Connectivity (JDBC) API was created by Javasoft, a Sun Microsystems operating company.

JDBC is based on the same Call Level Interface that ODBC was written from. JDBC implementations, however, are native Java code, while ODBC implementations are C coded programs. Developing a pure Java database driver is important in maintaining adequate performance and convenience to the Java developer. JDBC has been adopted by most major DBMS vendors, including Oracle, Informix, Imaginary, and Borland.

This study will provide background information about Internet and Java concepts at first. Then it will define ODBC and JDBC, as well explore the issues developers and implementors face in this ``Java-enhanced" world of database connectivity. Furthermore, this study will explain a JDBC client/server system developed by the author. The project's objectives are

- Introduce Internet and Java concepts
- Research the ODBC and JDBC specifications
- Study JDBC development
- Design, describe, and implement a JDBC compliant database system

CHAPTER TWO
THE INTERNET

The Internet was started as an experiment to test networks to try and develop a network that could survive a nuclear attack. While the net has never needed to survive a nuclear blast its design has proven again and again how robust it is. The Internet of today transmits data to almost every country in the world. This data includes everything from personal messages and currency transactions to military data.

By definition the word internet means a collection of networks that are joined together. Thus, the word Internet, with a capital 'I', means 'The collection of networks that are joined together.'

2.1. The History of the Internet

The history of the Internet begins with the RAND group in 1966. Paul Baran was commissioned by the U.S. Air Force to do a study on how it could maintain its command and control over its missiles and bombers, after a nuclear attack. Baran's finished document described several ways to accomplish this task. What he finally proposes is a packet switched network. This network would have no central hub, and no central control center. Instead it would have lines linking varies places together. Packets would be forwarded from place to place until they arrived at the proper destination.(Hardy, 1997) The theory was that if the middle of the country were taken out by a nuclear attack, the coasts could still talk to one another. This would be done by routing traffic around the missing links through Canada, satellites, or possibly by going around the world.

In 1969 what would later become the Internet was founded. It contrasts sharply with today's Internet. The ARPANET network had four machines on it, linked together with a packet switched network. Soon afterward other government agencies became interested in this new network; DoD, NASA, NSF, and the Federal Reserve Board. Because of this new interest and the fact that ARPANET was growing, now 24 nodes in 1972, IPTO began to look at other ways to transmit data other than through a wire. Two projects were launched to settle these needs. The first was the use of satellites for data transmission. The second project was for radio transmitted data. It soon also became apparent that a packet switched radio network, for mobile computing, would be possible.(Norberb & O'Neill, 1992) In 1976 the packet satellite project went into practical use. SATNET, Atlantic packet Satellite network, was born. This network linked the United States with Europe.

Because of the above two projects it soon became apparent that a new protocol for the network was needed. For one the original addressing scheme did not contain enough information to allow for the connection of other large networks. Also, a system had to be found to allow the network to recover if it could not talk to the host that it was trying to communicate with. To try and solve these problems IPTO hired Vinton Cerf who came from Stanford. Cerf and his students had written the first protocol used on ARPANET, NCP. (Norberb & O'Neill, 1992)

In 1973 Cerf and his group developed the protocols that were to be called TCP/IP, Transmission Control Protocol/ Internet Protocol. In 1976 the DoD began to experiment with this new protocol and soon decided to require it for use on ARPANET. January 1983 was the date fixed as when every machine connected to ARPANET had to use this new protocol. Also, at this time is when ARPANET ceased to be and the INTERNET came into being, for practical purposes. ARPANET was not officially taken out of service until 1990. What this means is that the actual 56Kbs lines were taken out of service. But, because the network is designed to route around missing parts no one noticed that they were gone. Those people that were still using the lines received some sort of a new connection to the NSF backbone. As a side note in 1984 the DoD decided to form MILNET out of the ARPANET. This was done to ensure that the military would have a reliable network to use for communication, while the ARPANET could still be used for advanced wide area network testing.

In 1981 two other networks that would play an important role in the INTERNET were founded. The first was BITNET. It was started by IBM and used IBM's RSCS protocol over leased lines that connected the different sites. BITNET is an interesting network in that it stands for "Because its Time Network". The basic principle of BITNET was that it was a store and forward network. This meant that first users could not get a real time connection over a BITNET connection. BITNET was designed to allow E- mail and mailing lists. The second network that was founded in 1981 was CSNET. This network was setup by NSF as a way to allow CS departments to work together. CSNET was designed to work over modems. Its basic purpose was much like BITNET's because it allowed peers to exchange mail. In 1987 both of these networks combined together to form CREN, Corporation for Research and Education Networking. In 1991 CREN discontinued CSNET, since its function had been fulfilled and could now be met by other means. CREN is a totally self-sufficient organization. Its costs are met by its members. BITNET its self is also slowing ceasing to exist. Most of its members are leaving, they can get more functionality out of an INTERNET connection.

In 1983 the University of Wisconsin made an incredible breakthrough in the way the INTERNET worked. They developed the idea of the domain name server. This meant that messages no longer needed to know the exact path, the IP number, to their destination. The domain name server in essence allowed a message to ask for directions along the way to its destination. For instance if you wanted to send a message to "resource.cso.uiuc.edu" the message would first go to the DNS that served "edu" which would then tell it how to get to "uiuc" which would tell the message were it could find "resource.cso". This was a vast improvement over the old system, before one of two things had to happen. Either the sender had to send a message to the IP#, which in the example above would be "128.174.201.130" or the administrator of the mainframe that you were using would have to place an entry in the hosts file that would transform the name into an IP number. This made administration of machines much easier, it also made life easier for the users, since people find it easier to remember names instead of numbers.

In 1984 the National Science Foundation got into the act of networking. The NSF felt that it would be good to setup super computer centers that people could purchase time on for research. Five centers for super computer research were established and each was linked via a backbone to the others. This NSF backbone was also linked to ARPANET.

The years 1991 to 1994 were years of growth and stability for the Internet. No major changes were made to the physical network. The most significant thing that happened was the growth. Many new networks were added to the NSF backbone. Hundreds of thousands of new hosts were added to the INTERNET during this time period. Also, significant new uses of the net were found. CREN developed the technology for the World Wide Web, which quickly became the second most used application on the INTERNET, based on the amount of data transferred.

2.2. The Web

The World Wide Web came into being in 1991, thanks to developer Tim Berners-Lee and others at the European Laboratory for Particle Physics, also known as Conseil Européenne pour la Recherche Nucléaire (CERN). The CERN team created the protocol based on hypertext that makes it possible to connect content on the Web with hyperlinks. Berners-Lee now directs the World Wide Web Consortium (W3C), a group of industry and university representatives that oversees the standards of Web technology. (Microsoft, 1998)

Early on, the Internet was limited to noncommercial uses because its backbone was provided largely by the National Science Foundation, the National Aeronautics and Space Administration, and the U.S. Department of Energy, and funding came from the government. But as independent networks began to spring up, users could access commercial Web sites without using the government-funded network. By the end of 1992, the first commercial online service provider, Delphi, offered full Internet access to its subscribers, and several other providers followed.

2.3. Who Controls the Internet?

No one authority controls the World Wide Web. Today's Web site authoring tools allow virtually anyone who has access to a computer and the Internet to post a Web site and contribute to the definition of what this medium is and what it can do. But the World Wide Web Consortium (W3C) does oversee the development of Web technology. (Microsoft, 1998)

W3C was formed by Berners-Lee in 1994. An international group of industry and university representatives, W3C promotes the Web by developing common protocols for transmitting information over the Internet. The consortium provides information, reference code, and prototype and sample applications to developers and users. It is hosted by the Massachusetts Institute of Technology's Laboratory for Computer Science in the United States, the Institut National de Recherche en Informatique et en Automatique in Europe, and the Keio University Shonan Fujisawa Campus in Japan. (Kehoe, 1992)

2.4. What Does Internet Do?

People use Internet basically to do four things: mail, discussion groups, long-distance computing, and file transfers. (Sterling, 1993)

Internet mail is "e-mail," electronic mail, faster by several orders of magnitude than the US Mail. Internet mail is somewhat like fax. It's electronic text. E-mail can also send software and certain forms of compressed digital imagery.

The discussion groups, or "newsgroups," are a world of their own. This world of news, debate and argument is generally known as "USENET. " USENET is, in point of fact, quite different from the Internet. USENET is not so much a physical network as a set of social conventions. In any case, at the moment there are some 2,500 separate newsgroups on USENET, and their discussions generate about 7 million words of typed commentary every single day. Naturally there is a vast amount of talk about computers on USENET, but the variety of subjects discussed is enormous, and it's growing larger all the time. USENET also distributes various free electronic journals and publications.

Long-distance computing was an original inspiration for ARPANET and is still a very useful service. Programmers can maintain accounts on distant, powerful computers, run programs there or write their own. Scientists can make use of powerful supercomputers a continent away. Libraries offer their electronic card catalogs for free search. Enormous CD-ROM catalogs are increasingly available through this service. And there are fantastic amounts of free software available.

File transfers allow Internet users to access remote machines and retrieve programs or text. Many Internet computers allow any person to access them anonymously, and to simply copy their public files, free of charge. This is no small deal, since entire books can be transferred through direct Internet access in a matter of minutes. Today, there are over a million such public files available to anyone who asks for them, and many more millions of files are available to people with accounts. Internet file-transfers are becoming a new form of publishing, in which the reader simply electronically copies the work on demand, in any quantity he or she wants, for free. New Internet programs, such as "archie," "gopher," and "WAIS," have been developed to catalog and explore these enormous archives of material.

CHAPTER THREE
JAVA

3.1. What is Java?

Java is two things: a a high-level programming language and a platform. (Javasoft, 1998)

3.1.1. Java Programming Language

Java is a high-level, object-oriented programming language. Java is unusual in that each Java program is both compiled and interpreted. With a compiler, a Java program is translated into an intermediate language called *Java bytecodes*--the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java bytecode instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how this works. (Javasoft, 1998)

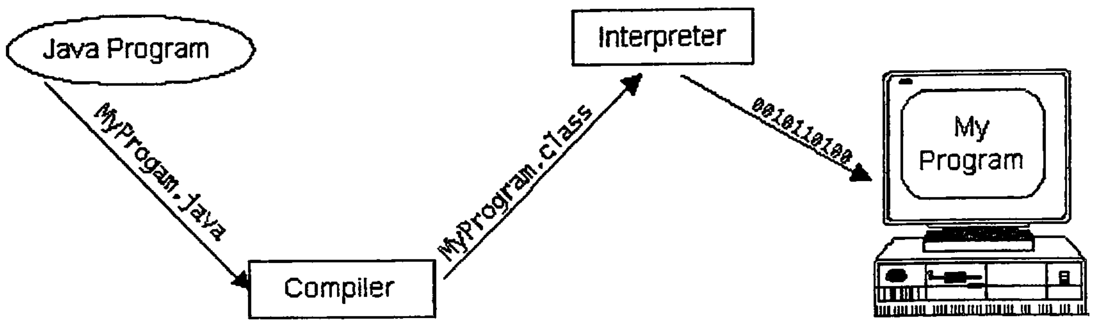


Figure 3-1 Java interpreter and compiler

Java bytecode is like machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware. Java bytecodes help make "write once, run anywhere" possible. The bytecodes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh.

3.1.2. The Java Platform

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components:

- The *Java Virtual Machine* (Java VM). The Java VM is the base for the Java platform and ported onto various hardware-based platforms.
- The *Java Application Programming Interface* (Java API). The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (*packages*) of related components.

The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring Java's performance close to that of native code.

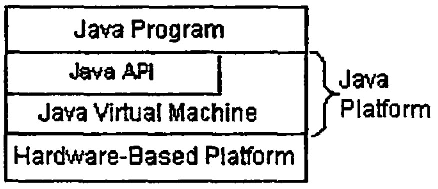


Figure 3-1 Java platform

3.2. Design Goals of Java

Java is designed to meet the challenges of application development in the context of heterogeneous, network-wide distributed environments. Paramount among these challenges is secure delivery of applications that consume the minimum of system resources, can run on any hardware and software platform, and can be extended dynamically. (Gosling&McGilton, 1995, p.12)

The massive growth of the Internet and the World-Wide Web leads developers to a completely new way of looking at development and distribution of software. To live in the world of electronic commerce and distribution, Java must enable the development of secure, high performance, and highly robust applications on multiple platforms in heterogeneous, distributed networks.Operating on multiple platforms in heterogeneous networks invalidates the traditional schemes of binary distribution, release, upgrade, patch, and so on. To overcome these difficulties, Java must be architecture neutral, portable, and dynamically adaptable. (Gosling&McGilton, 1995)

Some of the properties of Java and their respective benefits were explained below. (Gosling&McGilton, 1995)

3.2.1. Simple

Primary characteristics of Java include a simple language that can be programmed without extensive programmer training while familiar to current software practices. The fundamental concepts of Java are grasped quickly; programmers can be productive from the very beginning.

3.2.2. Object Oriented

Java is designed to be object oriented from the ground up. The needs of distributed, client-server based systems coincide with the encapsulated, message-passing paradigms of object-based software. To function within increasingly complex, network-based environments, programming systems must adopt object-oriented concepts. Java provides a clean and efficient object-based development environment.

3.2.3. Familiar

Even though C++ was rejected as an implementation language, keeping Java looking like C++ as far as possible results in Java being a familiar language, while removing the unnecessary complexities of C++. Having Java retain many of the object-oriented features and the "look and feel" of C++ means that programmers can migrate easily to Java and be productive quickly.

3.2.4. Robust

Java is designed for creating highly reliable software. It provides extensive compile-time checking, followed by a second level of run-time checking. The memory management model—no pointers or pointer arithmetic—eliminates entire classes of programming errors. It is possible to develop Java language code with confidence that the system will find many errors quickly.

3.2.5. Secure

Java is designed to operate in distributed environments, which means that security is of paramount importance. With security features designed into the language and run-time system, Java lets developers construct applications that can't be invaded from outside. In the networked environment, applications written in Java are secure from intrusion by unauthorized code attempting to get behind the scenes and create viruses or invade file systems.

3.2.6. Architecture Neutral

Java is designed to support applications that will be deployed into heterogeneous networked environments. In such environments, applications must be capable of executing on a variety of hardware architectures. Within this variety of hardware platforms, applications must execute a variety of operating systems and interoperate with multiple programming language interfaces. To accommodate the diversity of operating environments, the Java compiler generates bytecodes—an architecture neutral intermediate format designed to transport code efficiently to multiple hardware and software platforms. The interpreted nature of Java solves both the binary distribution problem and the version problem; the same Java language byte codes will run on any platform.

3.2.7. Portable

Architecture neutrality is just one part of a truly portable system. The primary benefit of the interpreted byte code approach is that compiled Java language programs are portable to any system on which the Java interpreter and run-time system have been implemented. The architecture-neutral aspect is one major step towards being portable. Java eliminates portability problem by defining standard behavior that will apply to the data types across all platforms. Java specifies the sizes of all its primitive data types and the behavior of arithmetic on them.

The architecture-neutral and portable language environment of Java is known as the Java Virtual Machine. It's the specification of an abstract machine for which Java language compilers can generate code. The Java Virtual Machine is based primarily on the POSIX interface specification—an industry-standard definition of a portable system interface.

3.2.8. High Performance

Performance is always a consideration. Java achieves superior performance by adopting a scheme by which the interpreter can run at full speed without needing to check the run-time environment. The automatic garbage collector runs as a low-priority background thread, ensuring a high probability that memory is available when required, leading to better performance. In general, users perceive that interactive applications respond quickly even though they're interpreted.

3.2.9. Interpreted

The Java interpreter can execute Java bytecodes directly on any machine to which the interpreter and run-time system have been ported. In an interpreted environment such as Java system, the link phase of a program is simple, incremental, and lightweight. You benefit from much faster development cycles—prototyping, experimentation, and rapid development are the normal case, versus the traditional heavyweight compile, link, and test cycles.

3.2.10. Threaded

Modern network-based applications typically need to do several things at the same time. A user working with multi-threaded Java application can perform several tasks concurrently. Java's multithreading capability provides the means to build applications with many concurrent threads of activity.

Multithreading thus results in a high degree of interactivity for the end user. Java supports multithreading at the language level with the addition of sophisticated synchronization primitives: the language library provides the Thread class, and the run-time system provides monitor and condition lock primitives. At the library level, moreover, Java's high-level system libraries have been written to be thread safe: the functionality provided by the libraries is available without conflict to multiple concurrent threads of execution.

3.2.11. Dynamic

While the Java compiler is strict in its compile-time static checking, the language and run-time system are dynamic in their linking stages. Classes are linked only as needed. New code modules can be linked in on demand from a variety of sources, even from sources across a network.

"The result is on-line services that constantly evolve; they can remain innovative and fresh, draw more customers, and spur the growth of electronic commerce on the Internet." (Gosling & McGilton, 1995, pp.12-16)

3.3. What Can Java Do?

The most well-known Java programs are *Java applets*. An applet is a Java program that adheres to certain conventions that allow it to run within a Java-enabled browser. Java is a general-purpose, high-level programming language and a powerful software platform. Using the generous Java API, programmers can write many types of programs. The most common types of programs are probably applets and applications. (Javasoft, 1998)

Another type of Java program is Java application that is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers include Web servers, proxy servers, mail servers, print servers, and boot servers. Another specialized program is a *servlet*. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java servers, configuring or tailoring the server. (Javasoft, 1998)

Java API support all of these kinds of programs with packages of software components that provide a wide range of functionality. The *core API* is the API included in every full implementation of the Java platform. The core API has the following features:

- **The Essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.

- **Applets**: The set of conventions used by Java applets.

- **Networking**: URLs, TCP and UDP sockets, and IP addresses.

- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.

- **Security**: Both low-level and high-level, including electronic signatures, public/private key management, access control, and certificates.

- **Software components**: Known as JavaBeans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's Live Connect.

- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).

- **Java Database Connectivity (JDBC)**: Provides uniform access to a wide range of relational databases. (JDBC is explained in Chapter-4 more detailed)

3.4. Java and the Web

A Web browser that implements the Java run-time system can incorporate Java applets as executable content inside of documents. This means that Web pages can contain not only static hypertext information but also interactive applications. A user can retrieve and use software simply by navigating with the Web browser.

"Formerly static information can be paired with portable software for interpreting and using the information. Instead of just providing some data for a spreadsheet, for example, a Web document might contain a fully functional spreadsheet application embedded within it that allows users to view and manipulate the information." (Niemeyer&Peck, 1996, p.22)

The term applet used to mean a small, subordinate, or embeddable application. Embeddable means that it's designed to be run within the context of a larger system. Most programs are embedded within a computer's operating system. An operating system manages its native applications in a variety of ways: it starts, stops, suspends, and synchronizes applications. Java applets is embedded in and controlled by larger applications, such as Java-enabled Web browser or an applet viewer. (Jamsa, 1996)

A Java applet is compiled Java program, composed of classes like any Java program. While a simple applet may consist of only a single class, most large applets should be broken into many classes. Each class is stored in a separate class file. The class files for an applet are retrieved from the network, as they are needed.

An applet has four-part life cycle. When applet initially loaded by a Web browse, it's asked to initialize itself. The applet is then informed each time it's displayed and each time it's no longer visible to the user. Finally, the applet is told when it's no longer needed, so that it can clean up after itself. During it's lifetime, an applet may start and suspend itself, do work, communicate with other applications, and interact with the Web browser.

3.5. What Applets Can and Can't Do

3.5.1. Security Restrictions

Every browser implements security policies to keep applets from compromising system security. The implementation of the security policies differs from browser to browser. Also, security policies are subject to change. For example, if a browser is developed for use only in trusted environments, then its security policies will likely be much more lax than those described below. (Javasoft, 1998)

Current browsers impose the following restrictions on any applet that is loaded over the network:

- An applet cannot load libraries or define native methods.
- It cannot ordinarily read or write files on the host that's executing it.
- It cannot make network connections except to the host that it came from.
- It cannot start any program on the host that's executing it.
- It cannot read certain system properties.
- Windows that an applet brings up look different than windows that an application brings up.

Each browser has a SecurityManager object that implements its security policies. When a SecurityManager detects a violation, it throws a SecurityException. An applet can catch this SecurityException and react appropriately. (Javasoft, 1998)

3.5.2. Applet Capabilities

The java.applet package provides an API that gives applets some capabilities that applications don't have. For example, applets can play sounds, which other programs can't do yet. (Javasoft, 1998)

Here are some other things that current browers and other applet viewers let applets do:

- Applets can usually make network connections to the host they came from.
- Applets running within a Web browser can easily cause HTML documents to be displayed.
- Applets can invoke public methods of other applets on the same page.
- Applets that are loaded from the local file system (from a directory in the user's CLASSPATH) have none of the restrictions that applets loaded over the network do.
- Although most applets stop running once you leave their page, they don't have to.

CHAPTER FOUR
JDBC

4.1. Open Database Connectivity (ODBC)

In an effort to standardize an interface to DBMS's, Microsoft created Open Database Connectivity (ODBC), based on the X/Open definitions of SQL CLI (Call Level Interface). ODBC is an API in which application developers can code their programs using ODBC function calls, and each DBMS vendor can provide an ODBC driver for their specific DBMS. An application written for the ODBC API can be used to access any DBMS, given the appropriate ODBC drivers (Visigenic, 1996).

ODBC alleviates the need for independent software vendors and corporate developers to learn multiple API's. ODBC now provides a universal data access interface. Application developers can allow an application to concurrently access, view, and modify data from multiple, diverse databases.

ODBC is a specification to which developers write either

- An ODBC-enabled ``front-end" or ``client" desktop application, also known as an ``ODBC Client." This is the application that the computer-user sees on the computer screen, or

- An ODBC Driver for a ``back-end" or ``server" DBMS (Database Management System). This is the DBMS application that resides on a computer that is used to store data for access by several users. This application is a standards consortium not what is loaded on the end user's computer. This server application is usually more robust than the client application. The

ODBC Driver resides between the ODBC Client and the DBMS; however, it is loaded on the front-end computer.

To use ODBC, the following three components are required. (Hamann, 1996)

ODBC CLIENT an ODBC-enabled front-end (also called ODBC client) - Examples Microsoft Access, an application created with Access, or ODBC enabled applications from other vendors (such as Lotus)

ODBC DRIVER an ODBC Driver for the ODBC Server. Any ODBC client can access any DBMS for which there is an ODBC Driver.

DBMS SERVER a back-end or server DBMS - Examples SQL Server, Oracle, AS/400, Access, or any DBMS for which an ODBC driver exists.

4.2. ODBC Over a Network

Some DBMS vendors provide a transport mechanism for client applications to access the database server over a network. MiniSQL is built around a network paradigm, as the database engine is a daemon 12 to be accessed locally via a UNIX domain socket or remotely via a TCP socket. Oracle provides developers with SQLNet, a set of libraries to facilitate data transfers over a TCP/IP network.

A three-tier architecture can be used to develop ODBC clients in a TCP/IP network. The client application is written to the ODBC specifications, and compiled with the ODBC and DBMS transport libraries. The client binary is now equipped to communicate with a DBMS server remotely, and the source code is portable among other DBMS's.

4.3. Java Database Connectivity (JDBC)

JDBC is a Java API for executing SQL statements. It consists of a set of classes and interfaces written in the Java programming language. JDBC provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API. (Sun Microsystems, 1997)

Using JDBC API, it is easy to send SQL statements to virtually any relational database and not necessary to write special programs to access different databases. The combination of Java and JDBC lets a programmer write it once and run it anywhere.

JDBC makes it possible to do three things:

- establish a connection with a database
- send SQL statements
- process the results.

JDBC is a "low-level" interface, which means that it is used to invoke SQL commands directly. It works very well in this capacity and is easier to use than other database connectivity APIs, but it was designed also to be a base upon which to build higher-level interfaces and tools. A higher-level interface is "user-friendly," using a more understandable or more convenient API that is translated behind the scenes into a low-level interface such as JDBC.

Two kinds of higher-level APIs were developed on top of JDBC:

- An embedded SQL for Java. DBMSs implement SQL, a language designed specifically for use with databases. JDBC requires that the SQL statements be passed as Strings to Java methods. An embedded SQL preprocessor allows a programmer to instead mix SQL statements directly with Java: for example, a Java variable can be used in a SQL statement to receive or provide SQL values. The embedded SQL preprocessor then translates this Java/SQL mix into Java with JDBC calls.

- A direct mapping of relational database tables to Java classes. In this "object/relational" mapping, each row of the table becomes an instance of that class, and each column value corresponds to an attribute of that instance. Programmers can then operate directly on Java objects; the required SQL calls to fetch and store data are automatically generated "beneath the covers." More sophisticated mappings are also provided, for example, where rows of multiple tables are combined in a Java class.

4.4. JDBC versus ODBC

Microsoft's ODBC (Open DataBase Connectivity) API is the most widely used programming interface for accessing relational databases. It offers the ability to connect to almost all databases on almost all platforms. ODBC can be used from Java without JDBC, but accessing a database from Java is best done ODBC with the help of JDBC in the form of the JDBC-ODBC Bridge. (Sun Microsystems, 1997)

Advantages of ODBC with JDBC:

1. ODBC is not appropriate for direct use from Java because it uses a C interface. Calls from Java to native C code have a number of drawbacks in the security, implementation, robustness, and automatic portability of applications.

2. A literal translation of the ODBC C API into a Java API would not be desirable. For example, Java has no pointers, and ODBC makes copious use of them, including the notoriously error-prone generic pointer "void *".

3. ODBC is hard to learn. It mixes simple and advanced features together, and it has complex options even for simple queries. JDBC, on the other hand, was designed to keep simple things simple while allowing more advanced capabilities where required.

4. A Java API like JDBC is needed in order to enable a "pure Java" solution. When ODBC is used, the ODBC driver manager and drivers must be manually installed on every client machine. When the JDBC driver is written completely in Java, however, JDBC code is automatically installable, portable, and secure on all Java platforms from network computers to mainframes.

The JDBC API is a natural Java interface to the basic SQL abstractions and concepts. It builds on ODBC rather than starting from scratch. JDBC retains the basic design features of ODBC, and both interfaces are based on the X/Open SQL CLI (Call Level Interface).

4.5. ODBC Compatibility

In an effort to bridge the initial gap between existing ODBC applications and the new JDBC specifications, Intersolv released a JDBC-ODBC Bridge. Java applications can be written using the JDBC API, but the API calls are filtered through the JDBC-ODBC bridge and converted into ODBC function calls. The remote DBMS (with the appropriate ODBC drivers) receives these calls as they do from any other ODBC client, and dutifully execute them. (Hamann, 1996)

4.6. JDBC Components

JDBC product consists of three components :

- **The JDBC driver manager :** The JDBC driver manager is the backbone of the JDBC architecture. Its primary function is to connect Java applications to the correct JDBC driver and then get out of the way.

- **The JDBC driver test suite :** The JDBC driver test suite provides some confidence that JDBC drivers will run a program. Only drivers that pass the JDBC driver test suite can be designated JDBC COMPLIANT.

- **The JDBC-ODBC bridge :** The JDBC-ODBC bridge allows ODBC drivers to be used as JDBC drivers. It was implemented as a way to get JDBC off the ground quickly, and long term will provide a way to access some of the less popular DBMSs if JDBC drivers are not implemented for them.

4.7. JDBC Driver Types

The JDBC drivers fit into one of four categories (Sun Microsystems, 1997):

4.7.1. JDBC-ODBC bridge plus ODBC driver:

The JavaSoft bridge product provides JDBC access via ODBC drivers. ODBC binary code, and in many cases database client code, must be loaded on each client machine that uses this driver. As a result, this kind of driver is most appropriate on a corporate network where client installations are not a major problem, or for application server code written in Java in a three-tier architecture.

4.7.2. Native-API partly-Java driver:

This kind of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.

4.7.3. JDBC-Net pure Java driver:

This driver translates JDBC calls into a DBMS- independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect its pure Java clients to many different databases. The specific protocol used depends on the vendor. In general, this is the most flexible JDBC alternative. It is likely that all vendors of this solution will provide products suitable for Intranet use. In order for these products to also support Internet access, they must handle the additional requirements for security, access through firewalls, and so on, that the Web imposes. Several vendors are adding JDBC drivers to their existing database middleware products.

4.7.4. Native-protocol pure Java driver:

This kind of driver converts JDBC calls into the network protocol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Intranet access. Since many of these protocols are proprietary, the database vendors themselves will be the primary source, and several database vendors have these in progress.

Driver categories 1 and 2 are interim solutions where direct pure Java drivers are not yet available. There are possible variations on categories 1 and 2 that require a connector, but these are generally less desirable solutions. Driver categories 3 and 4 will be the preferred way to access databases from JDBC. Categories 3 and 4 offer all the advantages of Java, including automatic installation and automatic download the JDBC driver with an applet that uses it.

DRIVER CATEGORY	ALL JAVA?	NET PROTOCOL
1 – JDBC-ODBC Bridge	No	Direct
2 – Native API as basis	No	Direct
3 – JDBC-Net	Yes	Requires Connector
4 – Native protocol as basis	Yes	Direct

Table 4-1 JDBC driver categories and their properties

4.8. Current JDBC Development

Several companies are pouring a great deal of personnel and resources into Java development. Some of these companies are small Internet firms, banking the company's future on the success of Java. Others, like Javasoft, are backed by large corporations seeking to dominate a new market.

4.8.1. Javasoft

Sun Microsystems released the Java language specifications an an alpha product in June 1995. Since then, Javasoft released two or three beta specifications, and finally, in January 1996, announced the release and availability of the Java 1.0 API, a production-quality release. Javasoft also develops Java compilers for Solaris and Windows 95/NT, to promote the use of the Java. This software is available from their WWW server, free of charge.

Javasoft continues to refine and improve the Java language with new specifications for certain functions.

JDBC JDBC allows Java applets to communicate with a wide range of database systems.

JavaBeans JavaBeans provides a platform-independent, portable component model and a rational security model. JavaBeans will work with ActiveX/COM, OpenDoc, and LiveConnect.

IDL Currently in its alpha 2.0 release, Java IDL provides a way for transparently connecting Java clients to network servers using the industry standard IDL Interface Definition language.

JECF Java Electronic Commerce Framework is a secure, extensible framework for conducting business on the Internet.

Java Workshop Currently in its beta release, Java Workshop is a commercial product that provides programmers a comprehensive environment for Java application development.

RMI and Object Serialization Object Serialization allows programs to serialize objects into a stream of bytes that can be later used to build equivalent objects. Remote Method Invocation (RMI) lets you create Java objects whose methods can be invoked from another virtual machine, analogous to a remote procedure call (RPC).

4.8.2. WebLogic

WebLogic, Inc., founded in 1995 in San Francisco, provides software for rapid development of products to build enterprise-wide, commercial applications. WebLogic's software products are written exclusively in Java.

WebLogic has focused on database access, and has developed the following products:

jdbcKona The jdbcKona products are a set of JDBC drivers for a variety of environments. The native drivers are based on vendor-supplied database libraries rather than on Microsoft's ODBC, and offer the performance of native vendor libraries as well as the platform neutrality of JDBC.

htmlKona htmlKona is a set of Java objects for programmatically generating complex HTML documents and constructing dynamic applications using CGI or one of the new Java-enhanced HTTP servers, like Netscape's Enterprise Server and JavaSoft's Jeeves. htmlKona is useful in an interactive HTTP/CGI environment or with Java-enabled HTTP servers, as well as for the periodic generation of static HTML pages.

eventKona/T3 **eventKona/T3**, WebLogic's event server, provides high-level event handling services between applications that generate user-defined events and applications that have registered an interest in the events. eventKona/T3 features include server-based Java evaluation of events as they are submitted to WebLogic's event server, and the ability to perform arbitrary actions based on the events.

4.8.3. Visigenic

Visigenic specializes in application middleware that provides developers access to heterogeneous databases, and enables multi-tier distributed applications for the Internet, intranet, and enterprise environments.

Their new product, OpenChannel, intends to streamline and simplify the design of a client/server database system. Visigenic's solution fits OpenChannel modules on both the client and server to negotiate the transmission of data in a standard form.

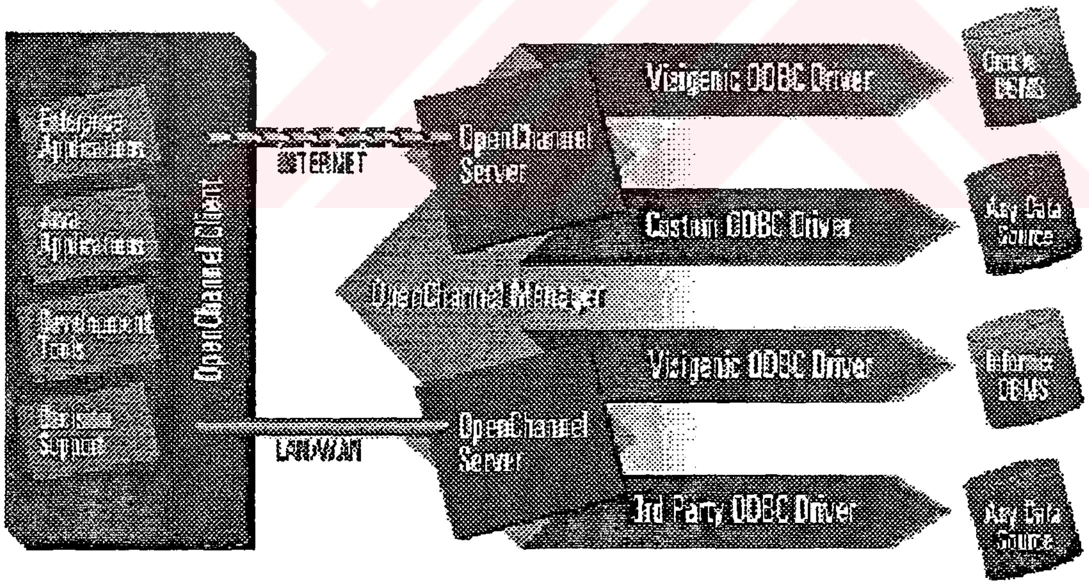


Figure 4-1 Visigenic's OpenChannel Architecture

4.9. SQL Conformance

Database systems support a wide range of SQL syntax and semantics, and they are not consistent with each other on more advanced functionality such as outer joins and stored procedures. Hopefully, the portion of SQL that is truly standard will expand to include more and more functionality. In the meantime, Javasoft takes the following position:

- JDBC allows any query string to be passed through to an underlying DBMS driver, so an application may use as much SQL functionality as desired at the risk of receiving an error on some DBMSs. In fact, an application query need not even be SQL, or it may be a specialized derivative of SQL, e.g. for document or image queries, designed for specific DBMSs.

- In order to pass JDBC compliance tests and to be called ``JDBC COMPLIANT", Javasoft requires that a driver support at least ANSI SQL92 Entry Level. This gives applications that want wide portability a guaranteed least common denominator. ANSI SQL-2 Entry Level is reasonably powerful and is reasonably widely supported today.

4.10. Abstract Interfaces

The JDBC API is expressed as a series of abstract Java interfaces that allow an application programmer to open connections to particular databases, execute SQL statements, and process the results. (Sun Microsystems, 1997)

The critical interfaces are

- **java.sql.DriverManager**, which handles the loading of DBMS drivers and provides support for creating new database connections.

- **java.sql.Connection**, which represents a connection to a particular database.

- **java.sql.Statement**, which acts as a container for executing an SQL statement on a given connection. Two sub-types are

- ♦ **java.sql.PreparedStatement**, for executing a pre-compiled SQL statement
- ♦ **java.sql.CallableStatement**, for executing a call to database stored procedure

- **java.sql.ResultSet**, which controls access to the row results of a given statement.

4.11. Security with JDBC

As with other Java applets, there are two scenarios a JDBC developer should consider (Hamann, 1996) :

Trusted Code A trusted Java program includes Java applications and applets from ``friendly" sources. A ``friendly" source could be your company's Information Services server or your professor's server. The applet can be signed with a cryptographic key to ensure its integrity during its download.

Untrusted Code Untrusted code is Java applets found across the Internet. Untrusted applets are not allowed access to the client's local devices, such as its filesystem. Furthermore, untrusted applets are only allowed to open network connections back to the server from which it was downloaded. An untrusted JDBC applet should avoid making any automatic or implicit use of local credentials when making connections to remote database servers.

4.12. Multi-Tier JDBC Systems

As a result of the complexities of serving executable content over an insecure network, developers have struggled with system architectures to provide efficient and convenient access to databases, while still maintaining system security and stability. In providing database access through JDBC, either from a stand-alone application or a World Wide Web applet, three major paradigms are generally followed one-tier, two-tier, or three-tier systems.

4.12.1. One Tier Systems

The one-tier system is only seen when the JDBC driver is completely written in Java code.

4.12.1.1. Standalone Application

The Java client code, the JDBC Driver Manager, and the JDBC Driver(s) are all contained in one tier, on the client machine.The database client may then connect to any host on the network and beginaccessing information in the remote database.

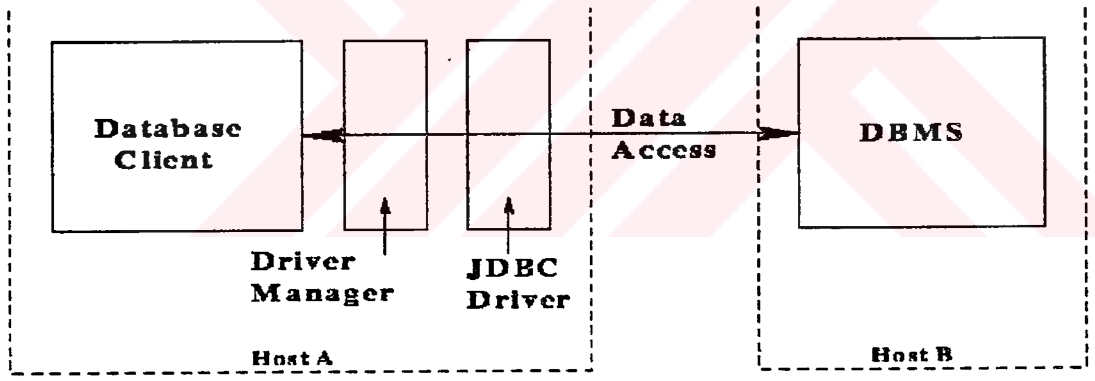


Figure 4-1 One-Tier Standalone Application

4.12.1.2. World Wide Web Applet

The Java client code, the JDBC Driver Manager, and the JDBC Driver(s) are all downloaded from the WWW server, onto the client machine. The database client may then connect to a remote database on the same host from which the classes were loaded from (the WWW server) and begin accessing information.

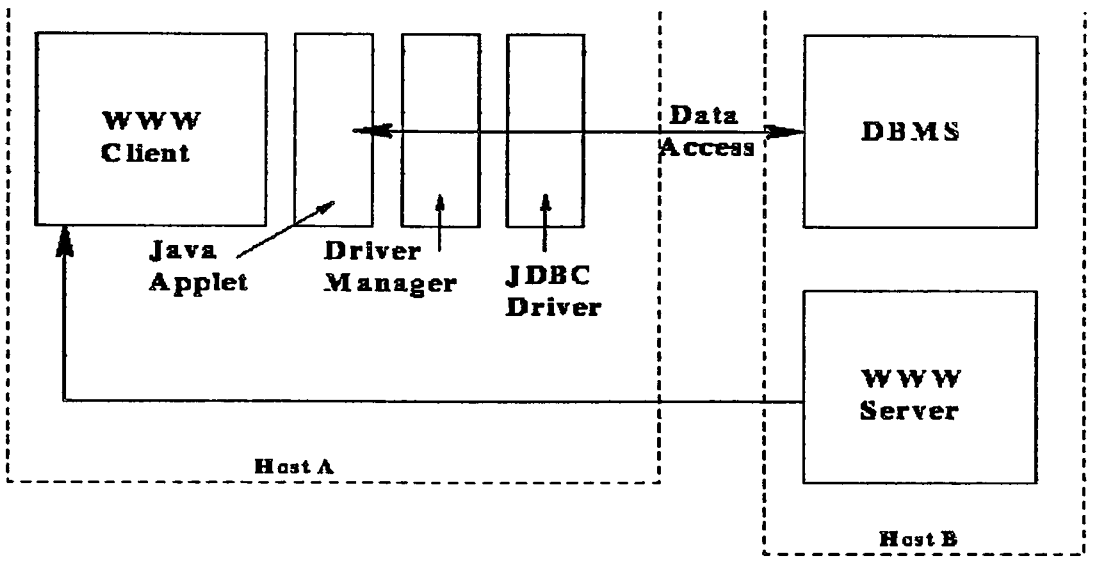


Figure 4-1 One-Tier World Wide Applet

4.12.2. Two Tier Systems

The two-tier system is applicable when the JDBC driver requires a native code library to translate JDBC functions into the DBMS's specific query language. WebLogic's JDBC driver for Oracle 7 uses a native code library, and therefore requires a two or three tier system.

4.12.2.1. Standalone Application

The Java client code, the JDBC Driver Manager, and the JDBC Driver(s) are all contained in one tier, on the client machine. In addition, the JDBC Driver (written in Java) requires a native code library to translate its JDBC functions into a language specific to one DBMS. This native-code library, specific to one DBMS and one operating system, is contained in the second tier. The database client may connect to any host on the network, and begin accessing information, via the native code library, in the remote database.

4.12.2.2. World Wide Web Applet

This architecture is not possible for WWW applets. The Java client code, the JDBC Driver Manager, and the JDBC Driver(s) can be downloaded from the network, but the required native code library cannot be retreived and used on the client machine. The library is platform-specific, and is not subject to the normal safety checks that the Java compiler and runtime engine impose on Java applets.

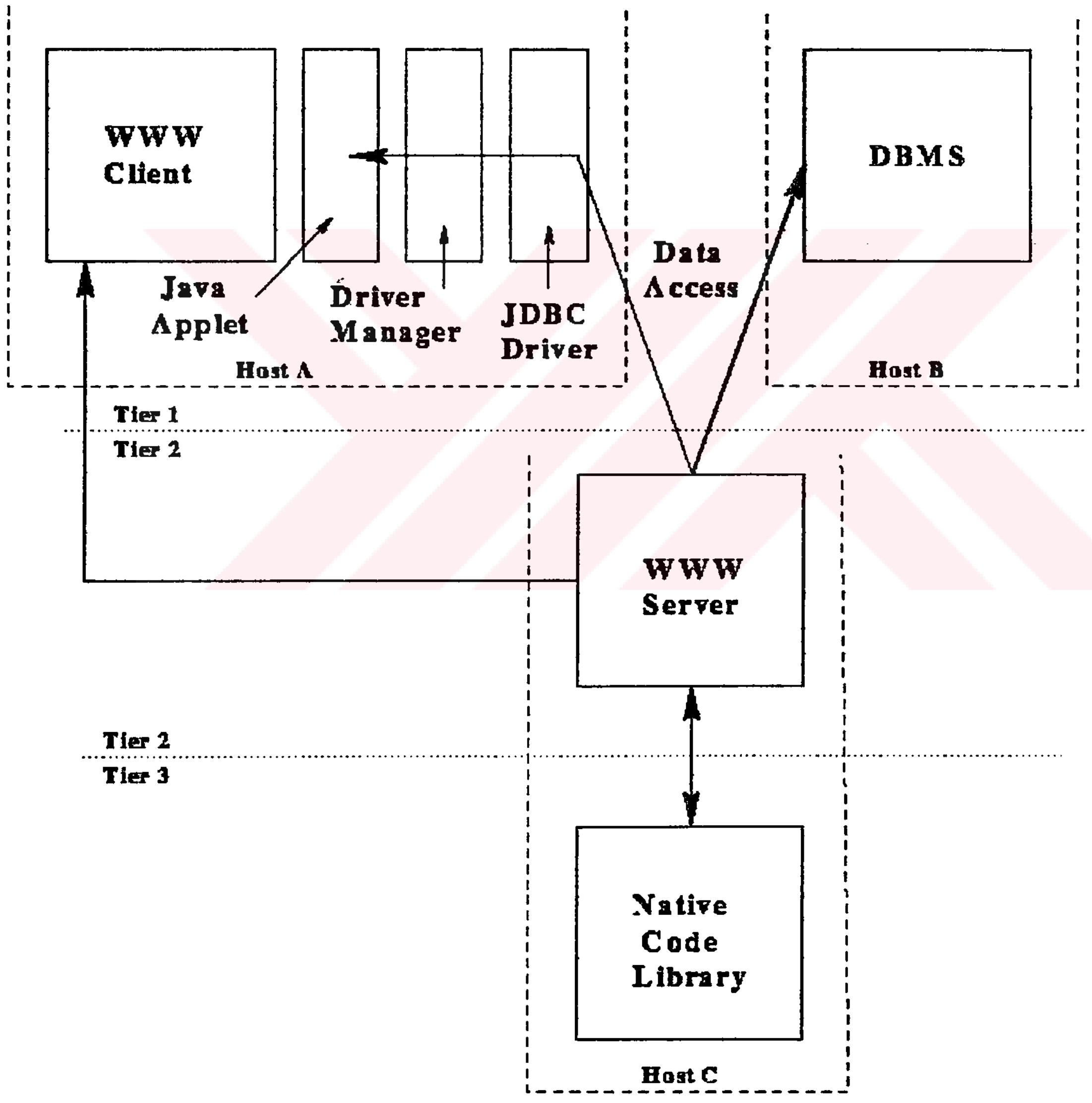


Figure 4-1 Two-Tier Standalone Application

4.12.3. Three Tier Systems

Three-tier systems are being developed by companies like WebLogic and Visigenic because of their support for database access from WWW applets. The JDBC drivers typically require a native code library for translations.

4.12.3.1. Standalone Application

The Java client code, the JDBC Driver Manager, and the JDBC Driver(s) are all contained in one tier, on the client machine.The native code library is housed on a second host, and serves as a gateway to a third tier, the DBMS. The database client may instantiate a ``session" with the native code library gateway, which in turn connects to the remote database system. As the client access data, the gateway negotiates the flow of information to and from the database system.

This allows a centralized point of administration for systems personnel, simplifying upgrades and protocol changes. It also creates a bottleneck situation if the gateway to flooded with requests simultaneously.

4.12.3.2. World Wide Web Applet

The Java client code, the JDBC Driver Manager, and the JDBC Driver(s) are downloaded from the host server to the client machine running a Java-enabled WWW browser (e.g. Netscape Navigator 2.x or later). The native code library is housed on that same host server, and serves as a gateway to a third tier, the DBMS. The database client may instantiate a ``session" with the native code library gateway, which in turn connects to the remote database system. As the client access data, the gateway negotiates the flow of information to and from the database system.

This architecture works around the browser policy restrictions of connecting only to the applet's ``home" server. The web server from which the applet was retrieved serves as a proxy gateway, by which applets can in fact communicate with any remote hosts the proxy allows. Once again, this architecture offers a centralized point of authentication and access control for systems personnel, rather than trying to maintain the access control restrictions in the code of many software programs distributed throughout an enterprise.

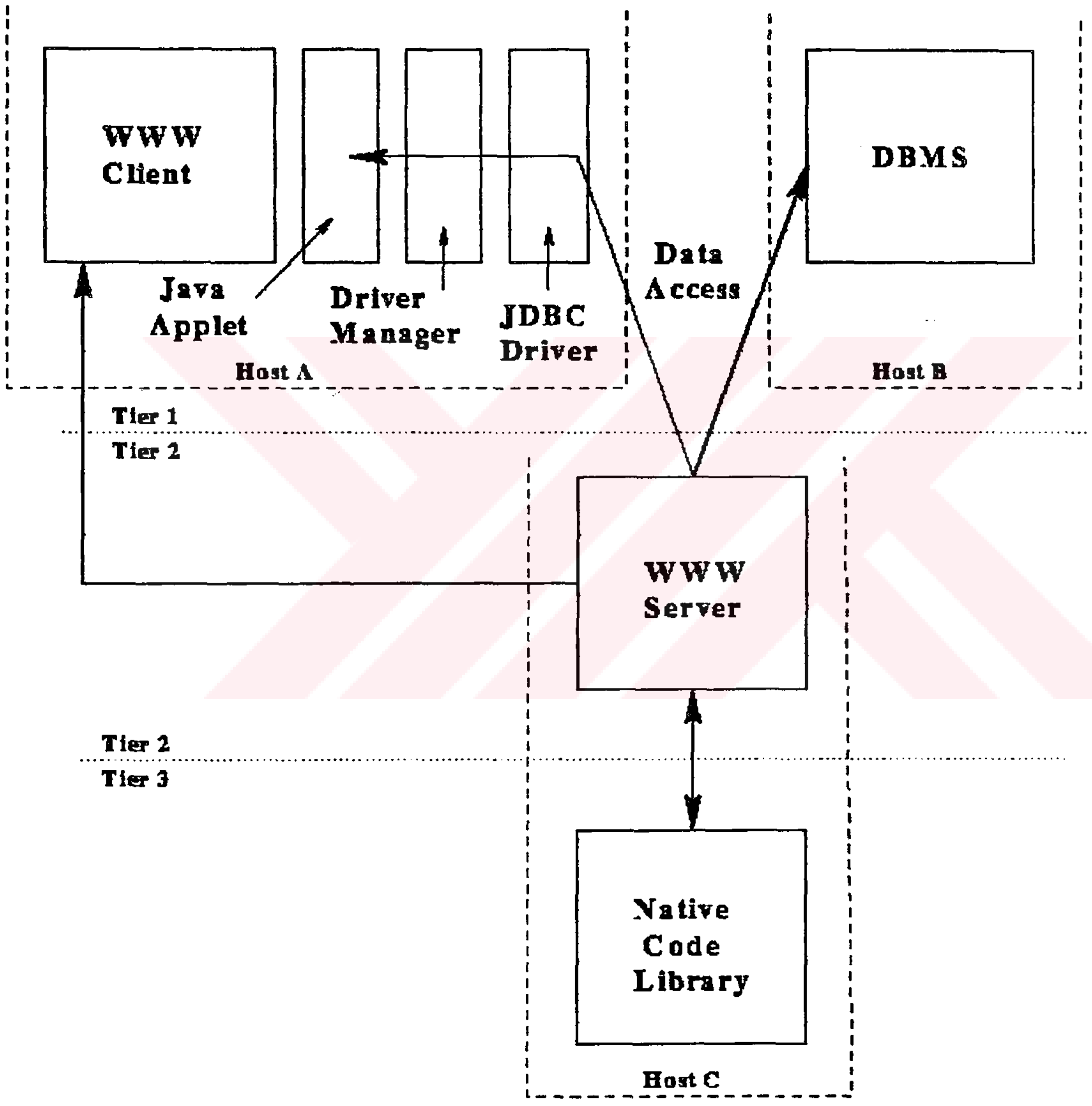


Figure 4-1 Three-Tier World Wide Web Applet

CHAPTER FIVE
A REAL APPLICATION

5.1. Description of the Project

I developed a real application, which enable authorised people to send or receive data stored in remote databases via Internet.

The project consists of three parts. The first part is for ordering via Internet, namely "Electronic Commerce". Second part of the project lets someone to query the main database of the company, so that he can get information about products, orders etc. The last part is about monitoring real-time data of the production system. The real-time data are temperature, pressure, or speed of the machines at the production floor.

Since Java is more suitable to internet applications, I chose Java environment to develop the project. To manage the project and create java applets, Borland **JBuilder** IDE was used. Borland **InterBase** was chosen as an SQL server. JDBC connection between applets and database is made with Borland **DataGateway**.

In order to make connection with control systems and get real time data from production floor, I used **FIX** Supervisory Control and Data Acquisition (SCADA) software and sent collected data to SQL server.

Before describing the project more detailed, development tools and other software will be explained in the next section.

5.2. Development Tools and Servers

5.2.1. Borland Jbuilder IDE

Borland JBuilder 2 is a family of highly productive, visual development tools for creating high-performance, platform-independent applications using the Java programming language created by Sun Microsystems. The JBuilder 2 scalable, component-based environment is designed for all levels of "Information Network" development projects, ranging from applets and applications that require networked database connectivity to client/server and enterprise-wide, distributed multi-tier computing solutions. The JBuilder 2 open environment supports 100% Pure Java, JavaBeans, Enterprise JavaBeans, Servlets, JDK 1.1, JDK 1.2, JFC/Swing, RMI, CORBA, JDBC, ODBC, and all major corporate database servers. JBuilder 2 will also provide developers with a flexible open architecture to incorporate new JDKs, third-party tools, add-ins, and JavaBean components.

JBuilder was architected to generate only 100% Pure Java code. All applications, applets, or JavaBeans created with JBuilder can work on any platform running a Java Virtual Machine: Windows, Unix, Mac, AS/400, mainframes, etc. They will also work in any browser supporting JDK 1.1.

JBuilder 2 offers the fastest rebuild/recompile speeds by using Smart Dependency Checking, which results in fewer unnecessary compiles of interdependent source files, and thus accelerates the edit/recompile cycle. When compiling, instead of deciding whether to recompile a source file based only on the time stamp of the file, JBuilder analyzes the nature of the changes made to source files. A source file is recompiled only if it uses (or depends on) a particular element that has changed within another source file. There are two commands for compiling: the Make command, and the Build command. The Make command compiles only the files that have changed. Make is the recommended command. The Build command compiles all the files.

JBuilder also includes a graphical debugger with all the features professional developers need to easily find and fix bugs. For faster development cycles, this professional debugger includes breakpoint setting, multi-thread support, single step through the code, etc.

Unique "Two-Way Tool" lets developers work simultaneously in the visual designers and pure Java code by switching between the Design and Source tabs, so they get all the benefits of visual programming without sacrificing the control of working in code.

The JBuilder compiler has full support for the Java language, including inner classes and JAR files. Developers can compile from within the IDE, or from the command line.

JBuilder supports the pure JDBC API, JavaSoft database connectivity specification. JDBC is the ultimate all-Java industry standard API for accessing and manipulating database data. JBuilder database applications can connect to any database that has a JDBC driver. All-Java based drivers can be loaded from the server or locally. The advantages to using a driver entirely written in Java is that it can be downloaded as part of an applet and is cross-platform. This is the preferred environment for Pure Java portable solutions. JBuilder also supports ODBC. If the connection to the database server is through an ODBC driver, use the JavaSoft (tm) JDBC-ODBC bridge software integrated within JBuilder.

5.2.1.1. The DataExpress Architecture

The JBuilder DataExpress architecture, in combination with the industry-standard JavaSoft JDBC call level API, provides powerful, vendor-independent support for Oracle, Sybase, Informix, Interbase, DB2, MS SQL server, Paradox, dBase, FoxPro, Access and other popular databases.

The DataExpress approach to data access and update of data sources like JDBC data sources can be explained as three phases:

1. A subset of data from a data source such as a SQL server is fetched into a DataSet component. This phase is called **"providing."**
2. The data in the DataSet can be freely navigated and edited entirely on the client machine without further communication with the original data source. All edits to the DataSet are all transparently recorded.
3. All of the recorded changes to a DataSet can be saved back to a data source such as a SQL server. This process is called **"resolving."** There is sophisticated built-in reconciliation technology to deal with potential edit conflicts.

The advantages of this approach can be summarized as follows.

- The personalities/semantics of data access and update for various data sources can be largely isolated to two clean points: providing (load the data) and resolution (save the changes to the data). For example, if data is accessed directly through a SQL server cursor—many issues come up: Is record locking supported? Are bidirectional cursors supported? What kind of transactional support is needed? What index ordering is supported? Where do posted records go—to the end, or where they are inserted, or in index order? How is data cached across transaction boundaries? Scroll bars like to know how many rows there are and which row is currently being scrolled to.

- Most SQL servers are highly tuned for the high volume of short running, small row set transactions that are common in an OLTP application. Because the DataExpress approach is "set" oriented, it is ideally suited to this environment. For providing, a transaction is only open long enough to fetch the result set. For resolving, the transaction stays open only long enough to save the changes.

- DataExpress encourages thin client application solutions when used with all-Java JDBC drivers. No special driver installation and registry settings are required with an all Java JDBC driver. This allows applications built with DataExpress technology to be run as an application or as an applet from with in a Web browser.

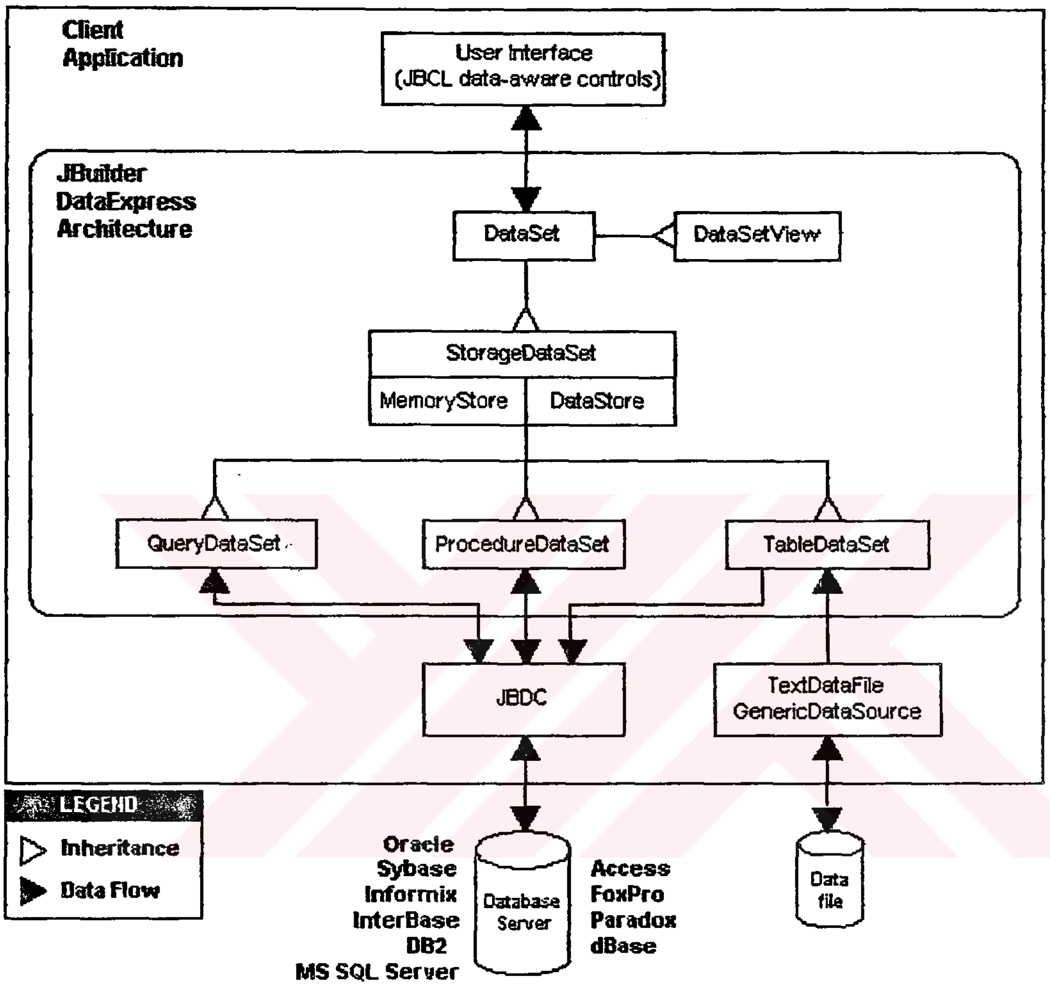


Figure 5-1 The DataExpress Architecture

- The DataExpress approach is also well-suited to application partitioning. DataSets allow for a deferred update model. This is because providing and resolving are done in separate transactions with an arbitrary amount of time for editing and processing in between the providing/resolving operations. DataSets are also well suited for streaming as parameters to remote methods. It is a rich data structure of manageable size that contains structure, data, and edit state.

- As more support for application partitioning is introduced in the DataExpress architecture, even thinner client software will be possible. The primary reason for this is that providing/resolving and business rule logic can be partitioned to another tier.

- DataSets on the client machine can be sorted and filtered independently of any indexes normally associated with the original data source.

- The data collected from the data source is a single consistent snapshot which is independent of changes which occur between the providing and the resolving of the DataSet. This effectively removes the traditional refresh problems associated with trying to reconcile data changes coming from other users while the data is being edited on a client machine.

The DataExpress approach is one of the most practical approaches for developing thin client/server-oriented applications on the Internet across a broad variety of data sources

5.2.1.2. The DataSet class hierarchy.

The DataSet class hierarchy is the centerpiece of the DataExpress architecture. The DataSet class hierarchy is not a set of implementable interfaces. DataSets provide a rich level of functionality and semantics for data access and update. The intent of the DataExpress architecture is to provide a powerful, performant, and tested set of components that are ready for use and customization. Customization is achieved by property settings, event handlers, and implementation of smaller, more focused interfaces such as Resolver and DataFile.

The DataSet class hierarchy is not a set of implementable interfaces. DataSets provide a rich level of functionality and semantics for data access and update. If interfaces existed for these APIs, the implementor would have the burden of supporting the rich functionality and semantics of the DataSet APIs. The leaf nodes of this class hierarchy are instantiable components. DataSet and StorageDataSet are abstract classes.

DataSet : An abstract class. A large amount of the public API for all DataSets is surfaced in this class. All navigation, data access, and update APIs for a DataSet are surfaced in this class. Support for master detail relationships, row ordering, and row filtering are surfaced in this class. All of our data, aware JBCL controls have a DataSet property. This means a GridControl can have its DataSet property set to the various extensions of DataSet: DataSetView, QueryDataSet, ProcedureDataSet, and TableDataSet.

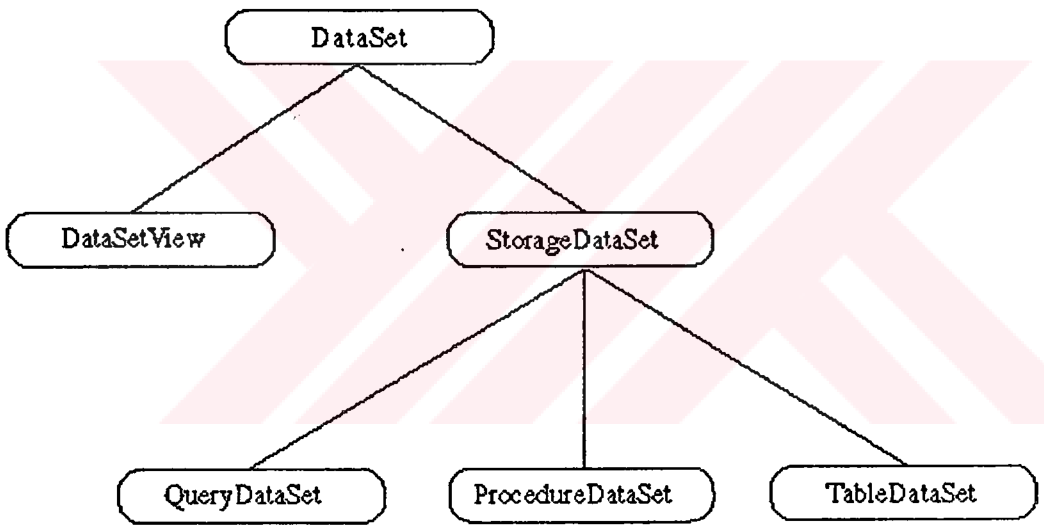


Figure 5-1 The DataSet class hierarchy

StorageDataSet : An abstract class. The StorageDataSet manages the storage of DataSet data,indexes used to maintain varying views of the data and persistent Column state. The current release provides efficient in-memory storage for data. The architecture also lends itself to plugging in persistent DataStores as well. All structural APIs (add/delete/change/move column) are surfaced in this class. Since StorageDataSets manage the data, it is where all row updates, inserts, and deletes are automatically recorded. Since all changes to the StorageDataSet are tracked, we know exactly what needs to be done to save (resolve) these changes back to the data source during a resolution operation.

DataSetView : This component can be used to provide independent navigation (a cursor) with a row ordering and filtering different than that used by the base DataSet. To use this, component DataSetView has a StorageDataSet property that must be set. This component can also be used when multiple controls need to dynamically switch to a new DataSet. The controls can all be wired to the same DataSetView. To force them all to view a new DataSet, the DataSetView StorageDataSet property can be changed.

QueryDataSet : This is a JDBC-specific DataSet. It manages a JDBC provider of data. The data to be provided is specified in a query property. The query property is a SQL statement.

ProcedureDataSet : This is a JDBC-specific DataSet. It manages a JDBC provider of data. The data to be provided is provided with a procedure property. The procedure property is a stored procedure.

TableDataSet : This is a generic DataSet component without anya built-in provider mechanism. Even though it has no default provider, it can be used to resolve its changes back to a data source. TableDataSets Columns and data can be added through DataSet methods or by importing data with a DataFile component like TextDataFile.

5.2.2. Borland DataGateway

Borland DataGateway for Java provides developers a multi-tier, fast, and reliable database connectivity solution adhering to the industry standard, JDBC. It uses the Borland Database Engine (BDE) and SQL Links, which supply drivers for many databases to access both local and remote data.

DataGateway is a network-protocol/all-Java driver. The DataGateway Client is written entirely in Java and this provides the following benefits:

1. Complete cross-platform support: Clients can connect from any platform supported by a Java Virtual Machine (JDK 1.1 or greater).
2. Zero-Configuration/Zero-Install Client: The Client is fully downloadable and requires no separate installation or configuration.

Borland DataGateway is a collection of JDBC drivers that allow Java applications and applets on any platform to access the following data sources:

Desktop databases

DBASE, Paradox, Microsoft Access, FoxPro

Client/Server databases

InterBase, Informix, Oracle, Sybase, Microsoft SQL Server, IBM DB/2

Other data sources

ODBC drivers (version 3 or 2.1)

DataGateway consists of the following parts:

Part	**Description**
DataGateway Client	Communicates with the DataGateway Server. It is written in Java and resides on the Java application/applet machine. The Client can exist on any machine that supports Java.
DataGateway Server	Manages the transfer of information and calls between the Client and the Bridge. Exists on Windows 95 and Windows NT machines only.

DataGateway Bridge	Translates the Java calls that come from a Java application/applet to BDE calls and BDE calls to Java. Exists on Windows 95 and Windows NT machines only.
BDE and SQL Links	Provides access to multiple data sources through a consistent API and native drivers.

Table 5-1 DataGateway components

The DataGateway Client, written completely in Java, runs on all hardware platforms that support Java and resides with the Java application or applet on the client machine. The client connects to the DataGateway Server on the server machine using TCP/IP protocol. The DataGateway Bridge connects to both local and remote data sources through BDE and SQL Links.

Tier 1: DataGateway Client

A Java application or applet makes JDBC calls through the DataGateway Client, and the Client connects to a Windows 95 or Windows NT server through the TCP/IP protocol.

Tier 2: DataGateway Server

The server has the DataGateway Server, the DataGateway Bridge, and the Borland Database Engine (BDE) installed, as well as valid BDE aliases for the data sources requested by the Java application or applet. The DataGateway Server manages the Client requests and passes the information on to the Bridge. The Bridge converts the JDBC calls to BDE calls, and then passes these calls to the BDE.

The BDE then queries the data source specified by the Java application or applet. If the data source is a local database such as dBASE or Paradox, the BDE uses its local drivers to retrieve the data from the database.

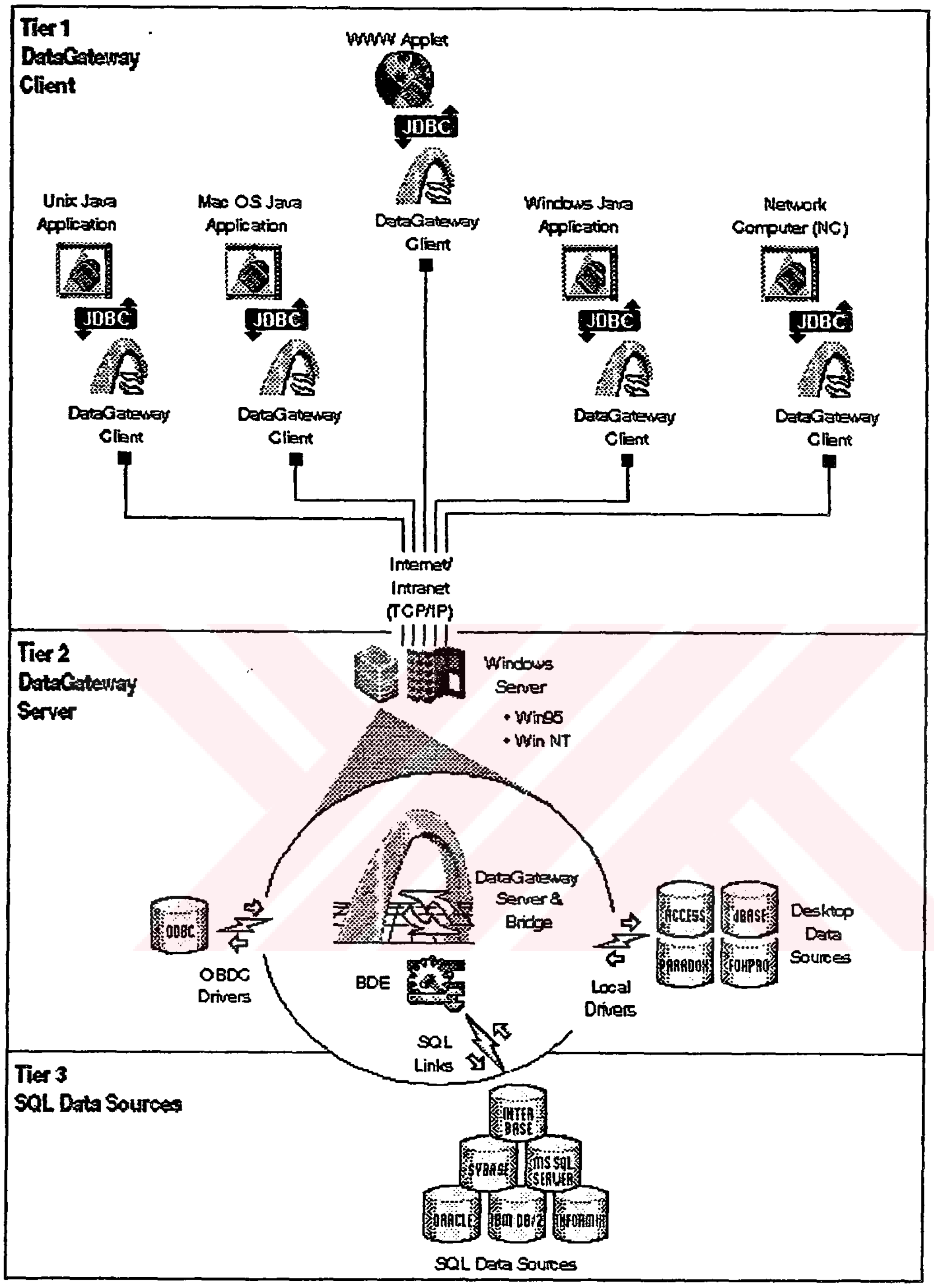


Figure 5-1 DataGateway structure

Tier 3: SQL data sources

If the data source specified by the Java application or applet is an SQL database, the BDE uses SQL Links to connect to the specified database and retrieves the data. The data is passed back to the Bridge for conversion back to JDBC, and is then returned via the Server to the Client. The Client then passes the returned data back to the Java application or applet.

DataGateway ban be configured and used in three ways:

Client on client machine, Server and Bridge on a server machine:

Here the DataGateway client resides on the various client virtual machines, and the DataGateway Server, Bridge, BDE and SQL Links reside on a Windows 95/NT server machine.

Client, Server, and Bridge all on one machine:

Here the DataGateway Client, Server, Bridge, BDE and SQL links, and application all reside and run on one machine.

Bridge only:

It is possible to use the DataGateway Bridge without the DataGateway Client and Server when the Java application resides on the same machine as the BDE and SQL Links.

5.2.3. Borland InterClient

Borland's InterClient allows developers to create Java applications and applets that communicate with an InterBase database over a WAN or LAN. The JDBC API is the framework for the InterClient API. The JDBC API is a Java application programming interface to SQL databases that was developed by Sun Microsystems.

As an all-Java JDBC driver, InterClient enables platform-independent, client/server development for the Internet and corporate Intranets. The advantage of an all-Java driver versus a native-code driver is that you can deploy InterClient-based applets without having to manually load platform-specific JDBC drivers on each client system (the Web servers automatically download the InterClient classes along with the applets). Therefore, there's no need to manage local native database libraries, which simplifies administration and maintenance of customer applications. As part of a Java applet, InterClient can be dynamically updated, further reducing the cost of application deployment and maintenance.

InterClient allows Java applets and applications to:

- Open and maintain a high-performance, direct connection to an InterBase database server
- Bypass resource-intensive, stateless Web server access methods
- Allow higher throughput speeds and reduced Web server traffic

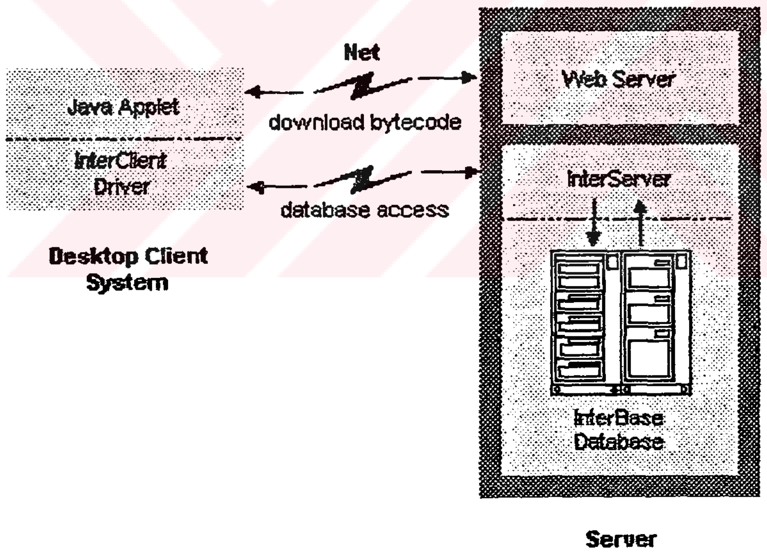


Figure 5-1 Borland InterClient architecture

The InterClient product consists of two major pieces:

1. A client-side Java package, called InterClient, containing a library of Java classes that implement most of the JDBC API and a set of extensions to the JDBC API.

This package interacts with the JDBC Driver Manager to allow client-side Java applications and applets to interact with InterBase databases.

2. A server-side driver, called InterServer. This server-side middleware serves as a translator between the InterClient-based clients and the InterBase database server.

InterClient Communication :

InterClient is a driver for managing interactions between a Java applet or application and an InterBase database server. On a client system, InterClient works with the JDBC Driver Manager to handle client requests through the JDBC API. To access an InterBase database, InterClient communicates via a TCP/IP connection with an InterServer translator that runs on the same system as the InterBase database server. InterServer forwards InterClient requests to the InterBase server and passes back the results to the InterClient process on the client machine.

5.2.4. Borland InterBase

InterBase is a relational database management system (RDBMS) that provides rapid transaction processing and data sharing in a single- or multi-user environment.

InterBase is a server technology that offers transparent support across heterogeneous networks. InterBase runs on Windows 95, Windows NT, Novell NetWare, and many implementations of the UNIX operating systems.

In addition, InterBase Server includes a driver for the Open Database Connectivity standard (ODBC) that enables ODBC client applications to share data with InterBase servers.

InterBase offers all the benefits of a fully relational DBMS. The following table lists some of the key InterBase features:

Feature	Description
Network protocol support	Support for both Microsoft NetBEUI/Named Pipes and TCP/IP for communication to clients. InterBase Server only.
SQL-92 entry-level conformance	ANSI standard SQL, available through an Interactive SQL tool and Borland desktop applications.
Simultaneous access to multiple DBs	One application can access many databases at the same time.
Multi-generational architecture	Server maintains older versions of records (as needed) so that transactions can see a consistent view of data.
Query optimization	Server optimizes queries automatically, or user may manually specify query plan.
Declarative referential integrity	Automatic enforcement of cross-table relationships (between FOREIGN and PRIMARY KEYs.)
Stored procedures	Programmatic elements in the database for advanced queries and data manipulation actions.
Triggers	Self-contained program modules that are activated when data in a specific table is inserted, updated, or deleted.

Updatable views	Views can reflect data changes as they occur.
Concurrent multiple application access	One application reading a table does not necessarily block others from it.
Automatic two-phase commit	Multi-database transactions check that changes to all databases happen before committing. (InterBase Server only)
Server Manager	Windows tool for database backup, restoration, maintenance, and security.
Windows ISQL	Interactive data definition and query tool for Windows
Comdiag	InterBase communications diagnostic tool.

5.2.5. Intellution FIX

FIX software is industrial automation software. Industrial automation software provides real-time data to plant personnel and other software applications throughout a plant. This real-time data presentation is the key to more efficient use of resources and personnel. The core software performs the basic functions that allow specific applications to perform their assigned tasks. The two most basic functions are data acquisition and data management.

Data Acquisition

Data acquisition is the ability to retrieve data from the plant floor and to process that data into a usable form. Data can also be written to the plant floor, thereby establishing the critical two-way link that control and application software require.

FIX software requires no proprietary hardware to acquire data. It communicates directly with the I/O devices already in place through a software interface called an I/O driver.

Data Management

Once data is acquired, it is manipulated and channeled according to the requests of software applications. This process is known as data management.

Since all FIX platforms has the intrinsic ability to communicate with nodes running on other platforms, plant managers can tie the entire plant together. The core FIX software manages the cross-platform data requests seamlessly.

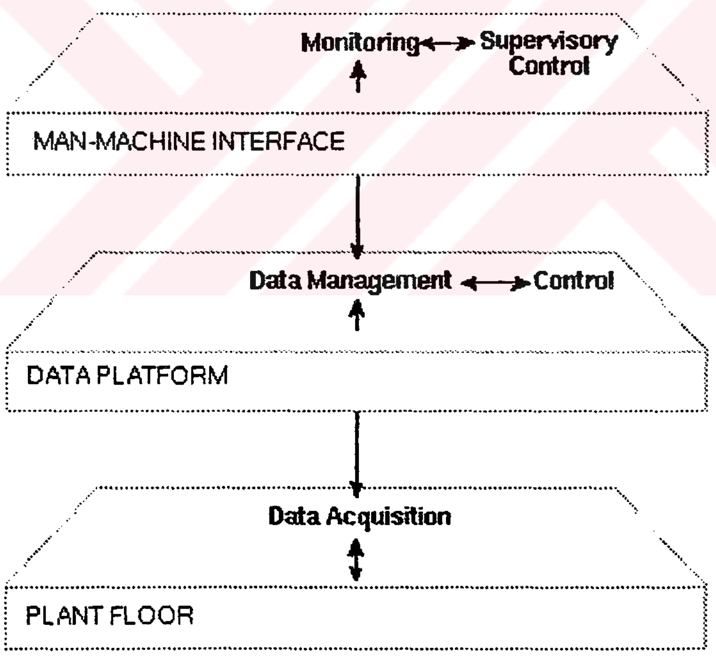


Figure 5-1 Intellution FIX basic functions

The basic functions of data acquisition and management provide the basis for all the industrial automation tasks, for example:

Monitoring

Monitoring is the ability to display real-time plant-floor data to operators. Powerful numeric, text, and graphical formats are available to make data more accessible.

Supervisory Control

Supervisory control is the ability to monitor real-time data coupled with the ability of operators to change set points and other key values directly from the computer.

Alarming

Whether operators are working from a monitoring station or a supervisory control station, they need the ability to immediately recognize exceptional events within the process. Alarming is the ability to recognize exceptional events and immediately report those events.

Control

Control is the ability to automatically apply algorithms that adjust process values and thereby maintain those values within set limits. Control goes one step beyond supervisory control by removing the need for human interaction. Database Definitions

5.3. Components of the Project

5.3.1. Databases

There are two databases in the project, LOCAL and REMOTE. The first database, LOCAL contains data about products, customers and orders, while other database, REMOTE contains real-time data coming from SCADA software. LOCAL database resides in the main server machine in which Gateway server, web server and SQL server run. REMOTE database is served by another SQL server that runs on the remote server machine located in the production floor.

<u>**LOCAL database contains tables below.**</u>

- Table **CUSTOMER** contains customers data.

```
/* Table: CUSTOMER, Owner: SYSDBA */
CREATE TABLE CUSTOMER (ID INTEGER NOT NULL,
        FIRSTNAME VARCHAR(25),
        MI CHAR(1),
        LASTNAME VARCHAR(40),
        PHONE VARCHAR(15),
        FAX VARCHAR(15),
        EMAIL VARCHAR(128),
        ADDR1 VARCHAR(30),
        ADDR2 VARCHAR(30),
        CITY VARCHAR(30),
        STATE VARCHAR(15),
        POSTALCODE VARCHAR(12),
        COUNTRY VARCHAR(20),
        SHIPNAME VARCHAR(60),
        SHIPADDR1 VARCHAR(30),
        SHIPADDR2 VARCHAR(30),
        SHIPCITY VARCHAR(30),
        SHIPSTATE VARCHAR(15),
        SHIPPOSTALCODE VARCHAR(12),
        SHIPCOUNTRY VARCHAR(20),
CONSTRAINT PK_CUSTOMER PRIMARY KEY (ID));
```

- Table **L_PRODUCTCATEGORY** is a lookup table and provide data for a choiceControl component.

```
/* Table: L_PRODUCTCATEGORY, Owner: SYSDBA */
CREATE TABLE L_PRODUCTCATEGORY (CATEGORY VARCHAR(25) NOT
NULL,
CONSTRAINT PK_CONPRODCAT PRIMARY KEY (CATEGORY));
```

- Table **L_STATUS** is another lookup table and provide data related to order status for a choiceControl component.

```
/* Table: L_STATUS, Owner: SYSDBA */
CREATE TABLE L_STATUS (STATUS VARCHAR(15) NOT NULL,
CONSTRAINT PK_STATUS PRIMARY KEY (STATUS));
```

- Table **ORDERITEM** contains data of each ordered item.

```
/* Table: ORDERITEM, Owner: SYSDBA */
CREATE TABLE ORDERITEM (ORDERID INTEGER NOT NULL,
        PRODUCTID INTEGER NOT NULL,
        QTY INTEGER,
        SALEPRICE DOUBLE PRECISION,
CONSTRAINT PK_ORDERITEM PRIMARY KEY (ORDERID, PRODUCTID));
```

- Table **ORDERS** stores information of orders.

```
/* Table: ORDERS, Owner: SYSDBA */
CREATE TABLE ORDERS (ID INTEGER NOT NULL,
        CUSTOMERID INTEGER NOT NULL,
        ORDERDATE DATE,
        STATUS VARCHAR(15),
        SHIPDATE DATE,
        ORDERTRACKNUM VARCHAR(20),
        CUSTOMERPONUM VARCHAR(10),
        AMTPAID DOUBLE PRECISION,
        SHIPNAME VARCHAR(60),
        SHIPADDR1 VARCHAR(30),
        SHIPADDR2 VARCHAR(30),
        SHIPCITY VARCHAR(30),
        SHIPSTATE VARCHAR(15),
        SHIPPOSTALCODE VARCHAR(12),
        SHIPCOUNTRY VARCHAR(20),
        BILLADDR1 VARCHAR(30),
        BILLADDR2 VARCHAR(30),
        BILLCITY VARCHAR(30),
          BILLSTATE VARCHAR(15),
          BILLPOSTALCODE VARCHAR(12),
          BILLCOUNTRY VARCHAR(20),
CONSTRAINT PK_ORDER PRIMARY KEY (ID));
```

- Table **PRODUCT** stores information of products.

```
/* Table: PRODUCT, Owner: SYSDBA */
CREATE TABLE PRODUCT (ID INTEGER NOT NULL,
          ISACTIVE SMALLINT,
          NAME VARCHAR(255),
          CATEGORY VARCHAR(25),
          BASEPRICE DOUBLE PRECISION,
          DISCOUNTPCT DOUBLE PRECISION,
          STOCKQTY INTEGER,
          MINREORDERQTY INTEGER,
CONSTRAINT PK_PRODUCT PRIMARY KEY (ID));
```

<u>REMOTE database contains tables below.</u>

- Table **ALARM** contains data about alarms occurred on the production floor, eg. High temperature, low pressure, broken resistants.

```
/* Table: ALARM, Owner: SYSDBA */
CREATE TABLE ALARM (LINENO VARCHAR(3),
          ALARMCODE VARCHAR(4),
          ALARMNAME VARCHAR(40),
          START DATE,
          END DATE);
```

- Table **SQLERR** is a SCADA software specific table and contains errors

```
/* Table: SQLERR, Owner: SYSDBA */
CREATE TABLE SQLERR (TD DATE,
          NODE VARCHAR(8),
          TAG VARCHAR(30),
        SQLNAME VARCHAR(8),
        FIX_ERR VARCHAR(100),
        SQL_ERR VARCHAR(250),
        PROG_ERR VARCHAR(100));
```

- Table **SQLLIB** is another SCADA software specific table and identify SQL commands for inserting alarms to ALARM table

```
/* Table: SQLLIB, Owner: SYSDBA */
CREATE TABLE SQLLIB (SQLNAME VARCHAR(8) NOT NULL,
        SQLCMD VARCHAR(200),
CONSTRAINT SQLLIBPRIMARYKEY1 PRIMARY KEY (SQLNAME));
```

- Table **REALVALUE** contains real-time data

```
/* Table: REALVALUE, Owner: SYSDBA */
CREATE TABLE REALVALUE (LINENO VARCHAR (3),
           ITEMNAME VARCHAR (20),
           VALUE DOUBLE PRECISION);
```

5.3.2. Java applets

DataModule1

DataModule1 is an implementation of the DataModule interface. The DataModule is a non visual container for components, especially DataBroker components (i.e. Database and DataSet components). This centralizes related database components into one module, allowing a separation of business rule logic and application logic.

A big advantage DataModules allow for easier reuse or sharing of components between multiple applets. This class employs the Singleton pattern to ensure that the class has only one instance. Accessing the singleton instance is made by calling the static method getDataModule. This method instantiates the DataModule for the first caller and returns this same instance to any successive callers:

DataModule1 dm = DataModule1.getDataModule();

The DataSets are contained and initialized here in this DataModule. Any master-detail relationships between datasets are defined here too. All persistent columns for the various datasets are defined and initialized here also.

Client-side business logic is implemented in the data module to centralize a common response by datasets to validation checks and database exceptions.

To access a dataset contained in the data module, use the getter methods for the various DataSets in the data module. These methods instantiate the DataSets for the first caller and returns the same instances to any successive callers, allowing mulitple applets to share the same dataset

```
DataModule1 dm = DataModule1.getDataModule();
QueryDataSet customerDataSet = dm.getCustomerDataSet();
```

MainApplet

MainApplet is the starting point of the project. This class implements a button menu to navigate to other functional areas of the project. During instantiation of this class, an attempt to make the database connection is made.

Pressing "Orders" button displays two choices that are "New Order" and "Find Order". If "New Order" is selected "New Order" page will be shown. If "Find Order" is selected "Find Order" page will be shown.

Pressing "Customers" button displays two choices that are "New Customer" and "Find Customer". If "New Customer" is selected "New Customer" page will be shown. If "Find Customer" is selected "Find Customer" page will be shown.

Pressing "Alarms", "RealValues", or "Products" buttons shows related pages.

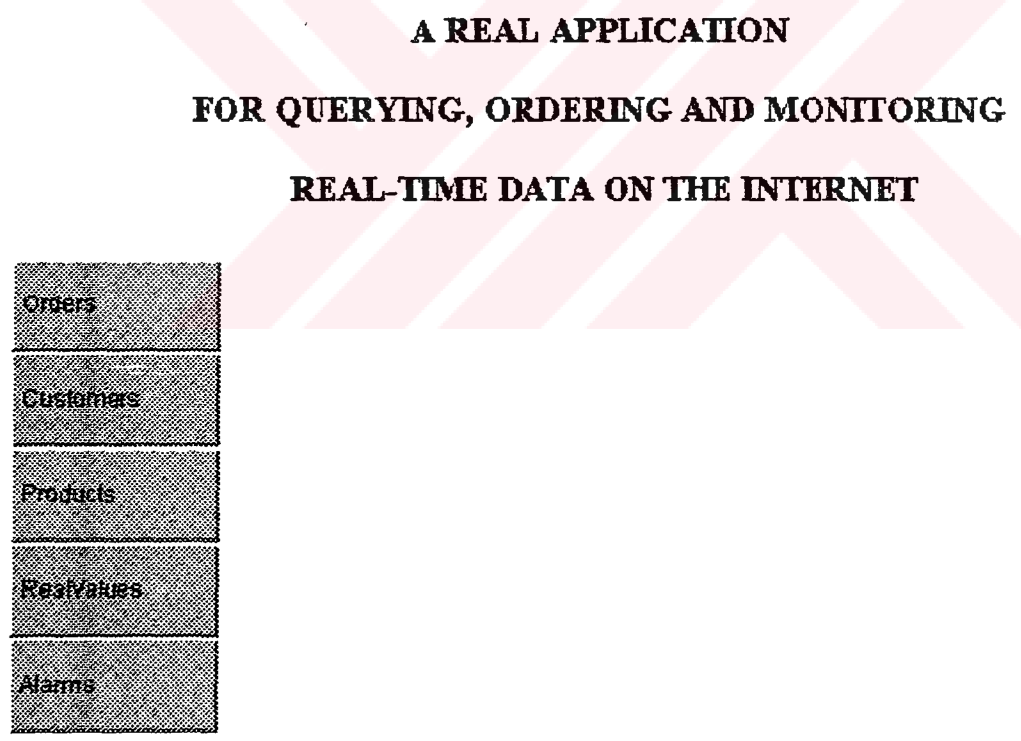
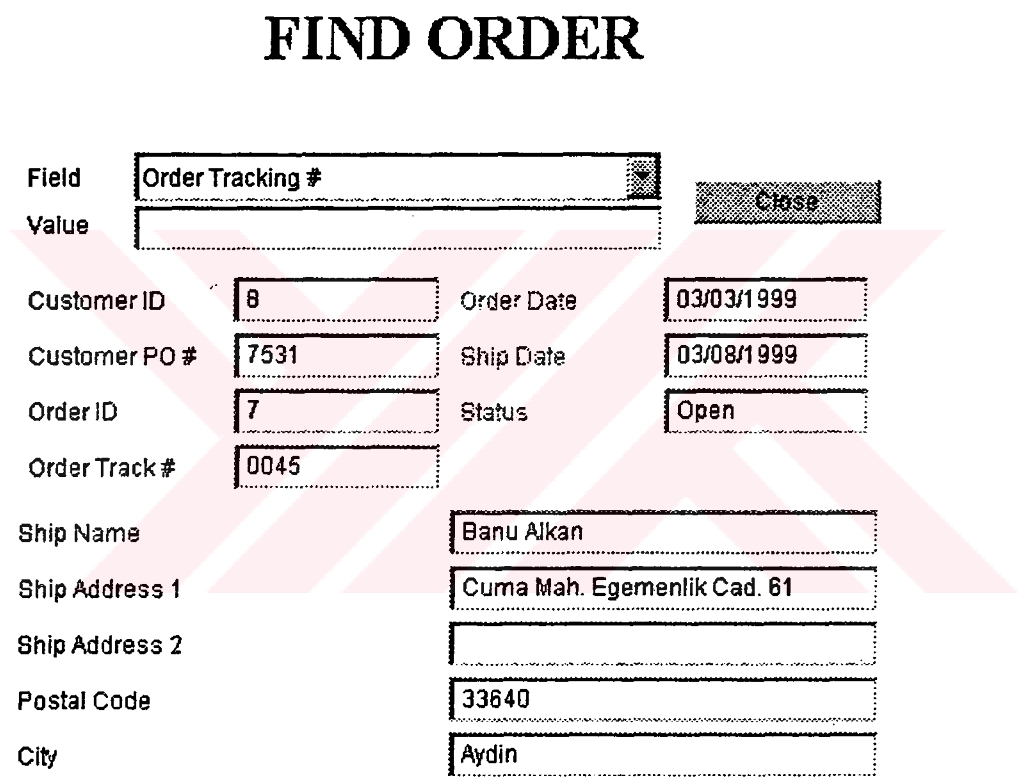

A REAL APPLICATION
FOR QUERYING, ORDERING AND MONITORING
REAL-TIME DATA ON THE INTERNET
Orders
Customers
Products
RealValues
Alarms

Figure 5-1 Screen view of MainApplet

FindOrderApplet

This find form allows users to lookup orders by searching on a value for a single column in the order dataset. "Field" choicebox is used to select search column, e.g. Order Tracking #, PO#, ID. By entering value into "Value" textbox related data is shown automatically. By pressing CLOSE button, it is possible to go to the main menu.



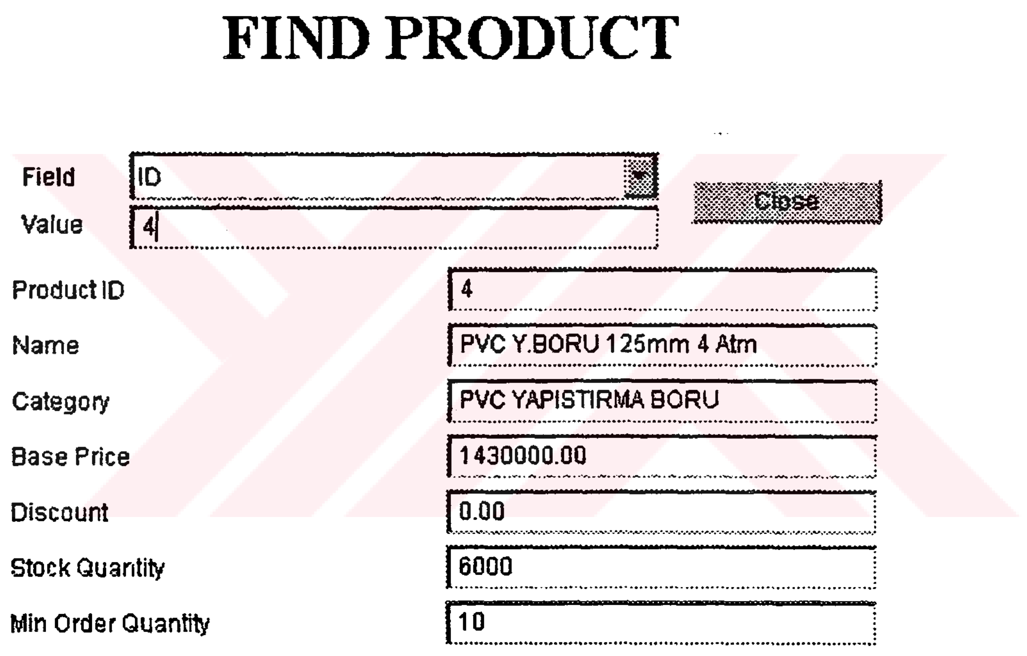
FIND ORDER

Field	Order Tracking #		
Value			
Customer ID	8	Order Date	03/03/1999
Customer PO #	7531	Ship Date	03/08/1999
Order ID	7	Status	Open
Order Track #	0045		
Ship Name	Banu Alkan		
Ship Address 1	Cuma Mah. Egemenlik Cad. 61		
Ship Address 2			
Postal Code	33640		
City	Aydin		

Figure 5-2 Screen view of FindOrderApplet

FindProductApplet

This find forms allows users to lookup products by searching on a value for a single column in the product dataset. "Field" choicebox is used to select search column, e.g. Name, Category, ID. By entering value into "Value" textbox related data is shown automatically. If there are more than one product with the same property, down arrow key can be used to show other customers. By pressing CLOSE button, it is possible to go to the main menu.



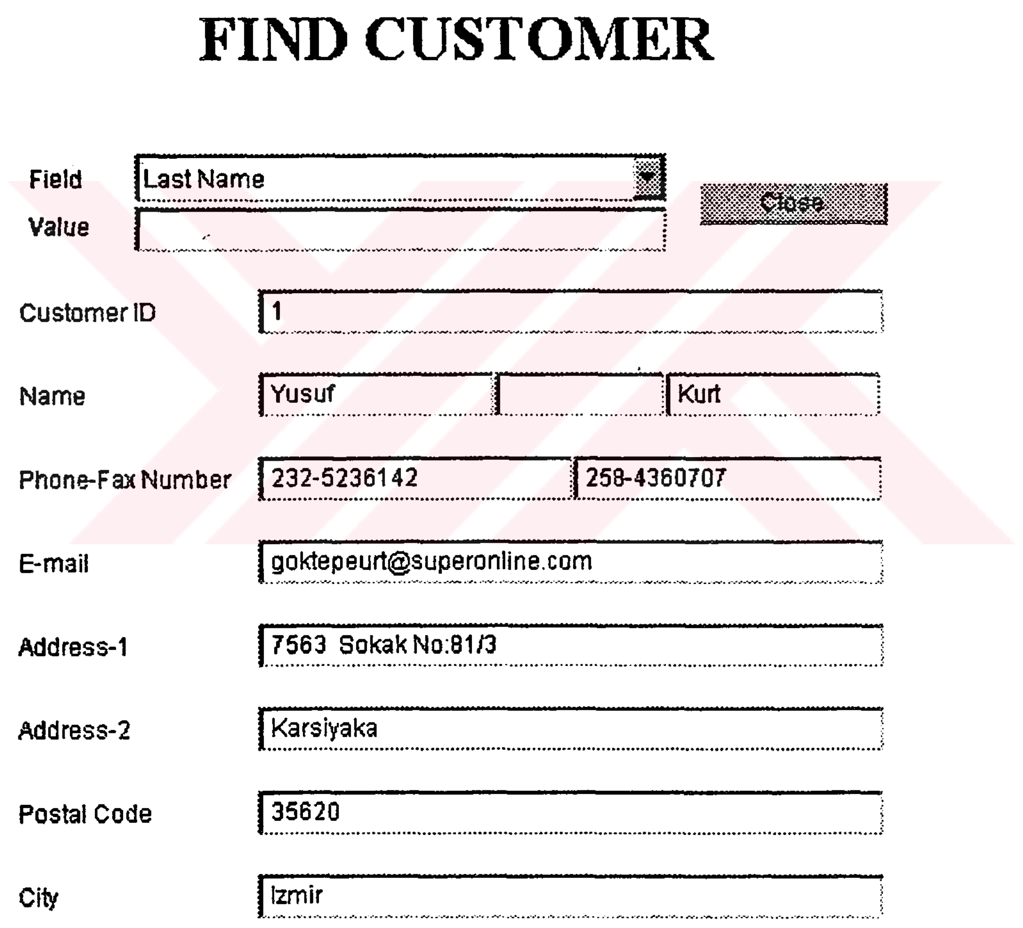
FIND PRODUCT

Field	ID
Value	4
Product ID	4
Name	PVC Y.BORU 125mm 4 Atm
Category	PVC YAPISTIRMA BORU
Base Price	1430000.00
Discount	0.00
Stock Quantity	6000
Min Order Quantity	10

Figure 5-3 Screen view of FindProductApplet

FindCustomerApplet

This find form allows users to lookup customers by searching on a value for a single column in the customer dataset. "Field" choicebox is used to select search column, e.g. LastName, FirstName, ID. By entering value into "Value" textbox related data is shown automatically. If there are more than one customer with the same property, down arrow key can be used to show other customers. By pressing CLOSE button, it is possible to go to the main menu.



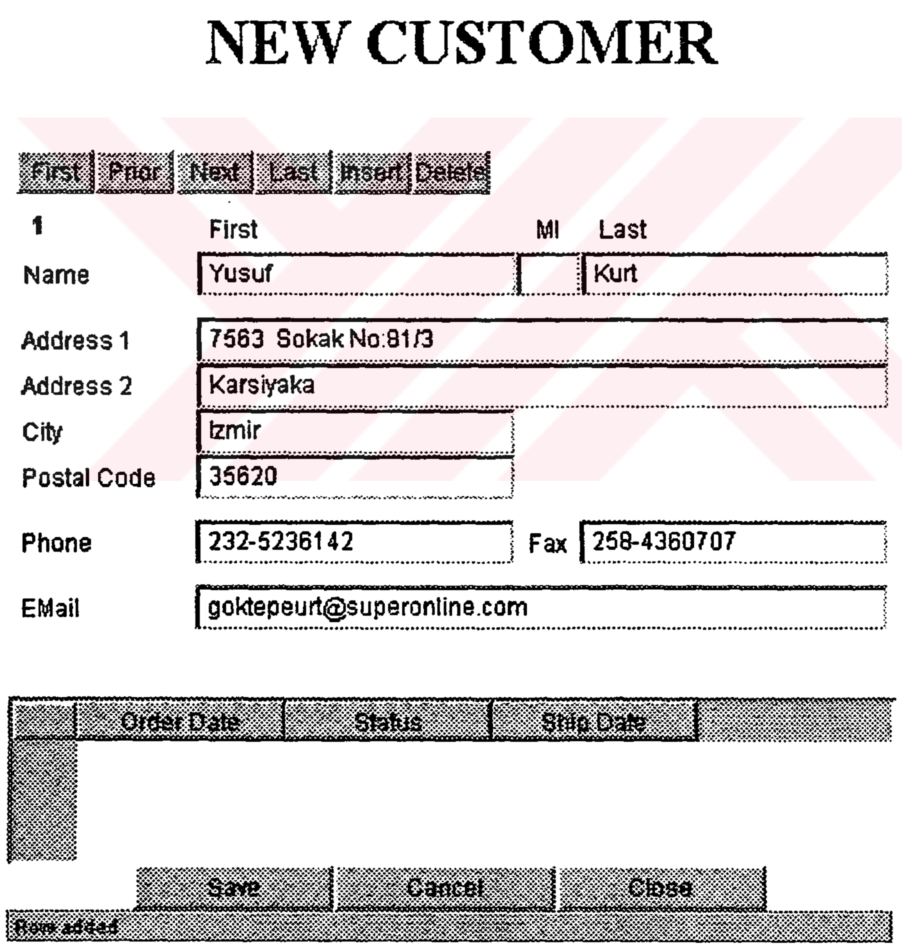
FIND CUSTOMER

Field	Last Name		
Value			
Customer ID	1		
Name	Yusuf		Kurt
Phone-Fax Number	232-5236142	258-4360707	
E-mail	goktepeurt@superonline.com		
Address-1	7563 Sokak No:81/3		
Address-2	Karsiyaka		
Postal Code	35620		
City	Izmir		

Figure 5-1 Screen view of FindCustomerApplet

CustomerApplet

CustomerApplet implements the Customer Form that is used to maintain Customer records. This form allows users to view or update existing customers, or enter new customers. A standard navigation control is used to allow for record browsing. A master-detail relationship between customers and orders is used to display a list of orders placed by each customer. SAVE button post entered data to the database. CANCEL button cancels all modifications. By pressing CLOSE button, it is possible to go to the main menu.

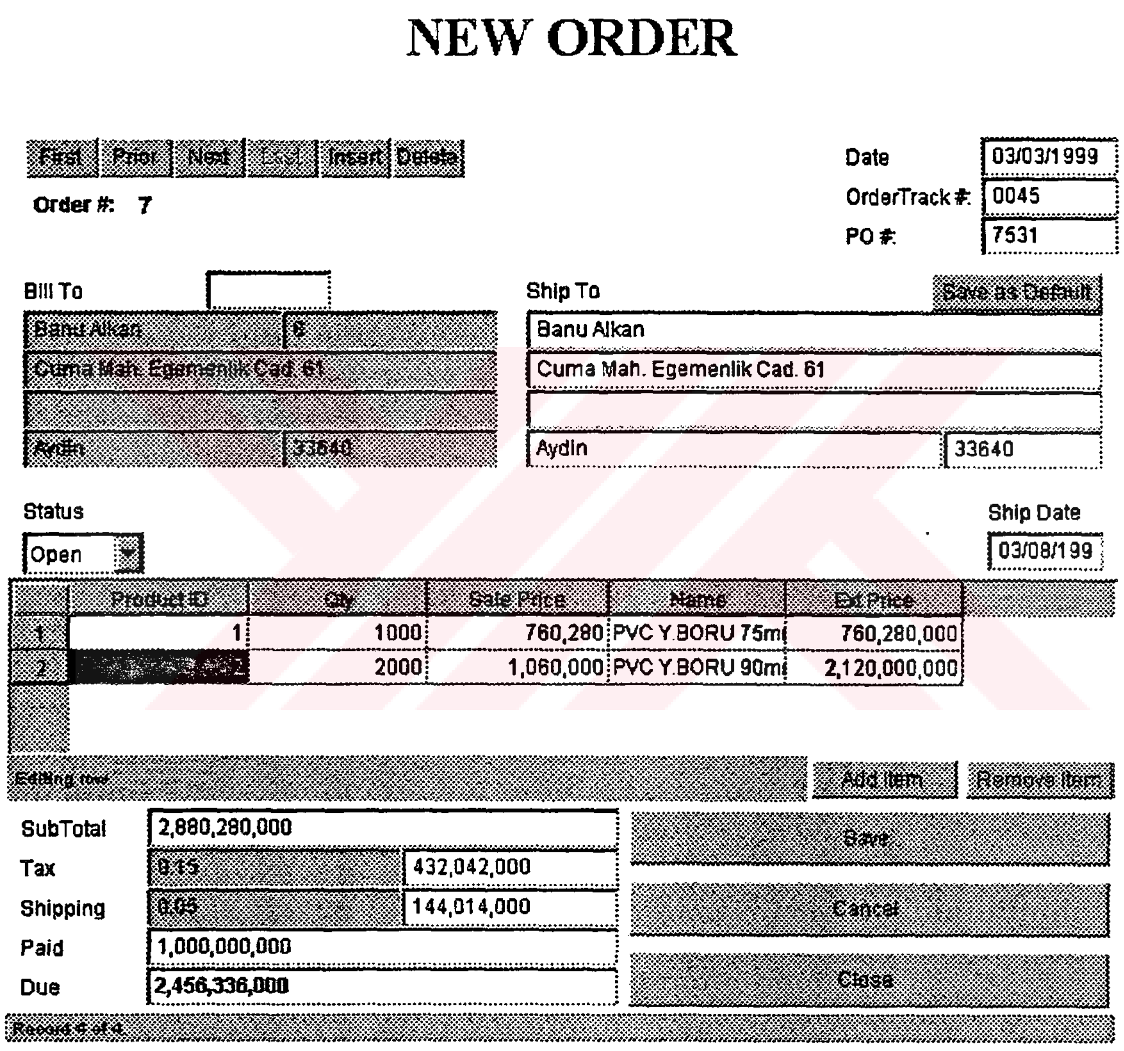


1	First	MI	Last
Name	Yusuf		Kurt
Address 1	7563 Sokak No:81/3		
Address 2	Karsiyaka		
City	Izmir		
Postal Code	35620		
Phone	232-5236142	Fax	258-4360707
EMail	goktepeurt@superonline.com		

Figure 5-1 Screen view of CustomerApplet

OrderApplet

OrderApplet is used to view, insert, and update Order records. SAVE button post entered data to the database. CANCEL button cancels all modifications. By pressing CLOSE button, it is possible to go to the main menu.



Order #: 7

Date 03/03/1999
OrderTrack #: 0045
PO #: 7531

Bill To: Banu Alkan | 8; Cuma Mah. Egemenlik Cad. 61; Aydin | 33640

Ship To: Banu Alkan; Cuma Mah. Egemenlik Cad. 61; Aydin | 33640

Status: Open

Ship Date: 03/08/199

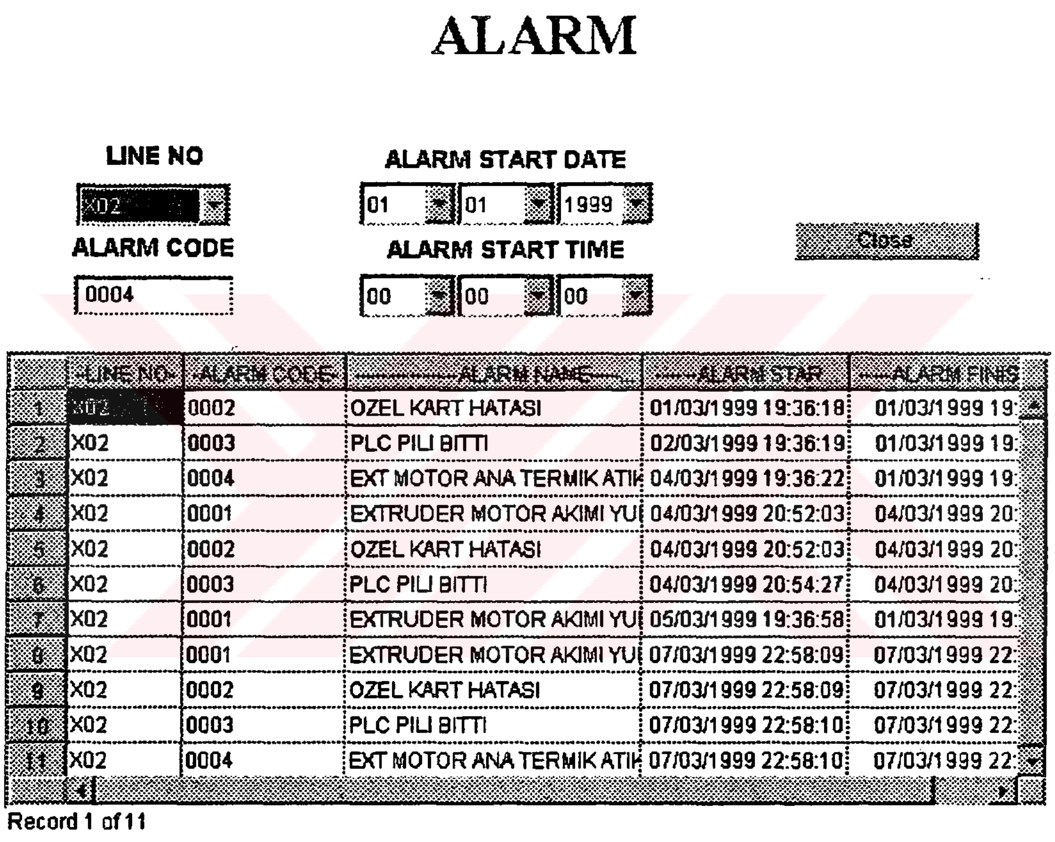
	Product ID	Qty	Sale Price	Name	Ext Price
1	1	1000	760,280	PVC Y.BORU 75ml	760,280,000
2	2	2000	1,060,000	PVC Y.BORU 90ml	2,120,000,000

SubTotal	2,880,280,000	
Tax	0.15	432,042,000
Shipping	0.05	144,014,000
Paid	1,000,000,000	
Due	2,456,336,000	

Figure 5-2 Screen view of OrderApplet

AlarmApplet

Alarm applet lets users view alarms provided by ALARM table of REMOTE database. Specifying line number or alarm code with date/time combination can filter displayed alarms. When choosing LINENO, all alarms with selected attributes will be shown. By pressing CLOSE button, it is possible to go to the main menu.



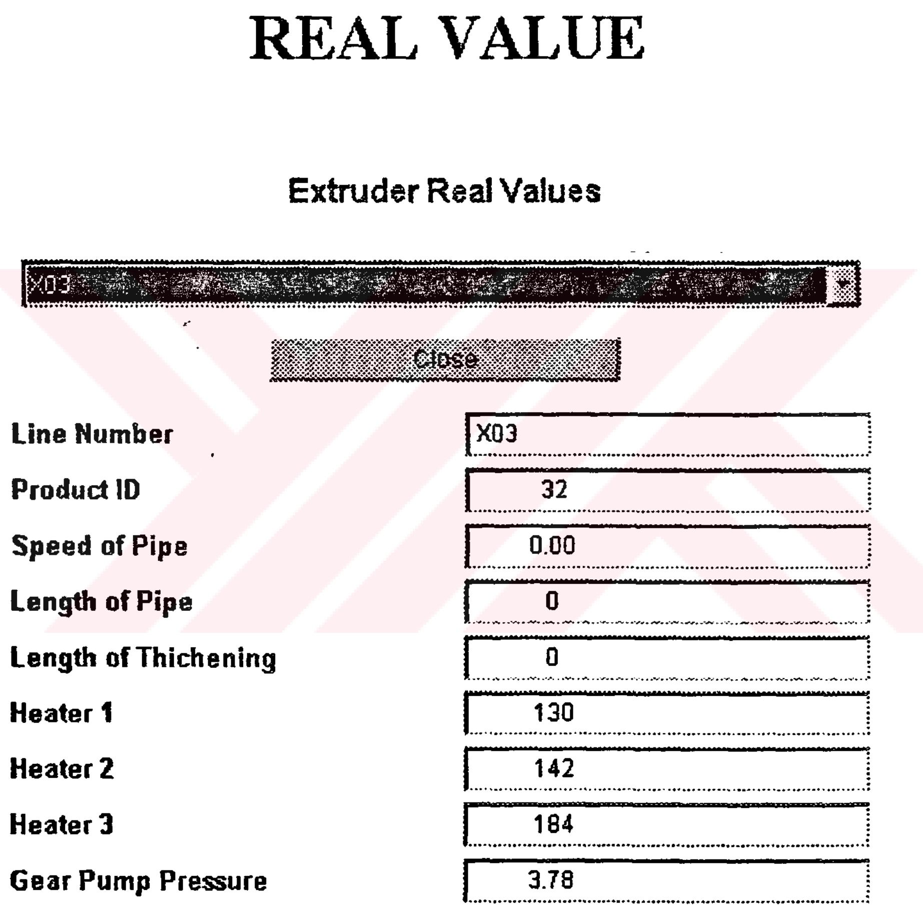
	LINE NO	ALARM CODE	ALARM NAME	ALARM STAR	ALARM FINIS
1	X02	0002	OZEL KART HATASI	01/03/1999 19:36:18	01/03/1999 19:
2	X02	0003	PLC PILI BITTI	02/03/1999 19:36:19	01/03/1999 19:
3	X02	0004	EXT MOTOR ANA TERMIK ATIK	04/03/1999 19:36:22	01/03/1999 19:
4	X02	0001	EXTRUDER MOTOR AKIMI YU	04/03/1999 20:52:03	04/03/1999 20:
5	X02	0002	OZEL KART HATASI	04/03/1999 20:52:03	04/03/1999 20:
6	X02	0003	PLC PILI BITTI	04/03/1999 20:54:27	04/03/1999 20:
7	X02	0001	EXTRUDER MOTOR AKIMI YU	05/03/1999 19:36:58	01/03/1999 19:
8	X02	0001	EXTRUDER MOTOR AKIMI YU	07/03/1999 22:58:09	07/03/1999 22:
9	X02	0002	OZEL KART HATASI	07/03/1999 22:58:09	07/03/1999 22:
10	X02	0003	PLC PILI BITTI	07/03/1999 22:58:10	07/03/1999 22:
11	X02	0004	EXT MOTOR ANA TERMIK ATIK	07/03/1999 22:58:10	07/03/1999 22:

Record 1 of 11

Figure 5-1 Screen view of AlarmApplet

RealValueApplet

The purpose of RealValueApplet is monitoring working condition of production floor to authorised people. By selecting line number, several real-time values are displayed. Real-time values are updated with five-second interval by SCADA software. RealValueApplet also updates those values by re-executing query within a thread.



REAL VALUE

Extruder Real Values

Line Number	X03
Product ID	32
Speed of Pipe	0.00
Length of Pipe	0
Length of Thichening	0
Heater 1	130
Heater 2	142
Heater 3	184
Gear Pump Pressure	3.78

Figure 5-1 Screen view of RealValueApplet

5.3.3. Structure of the Project

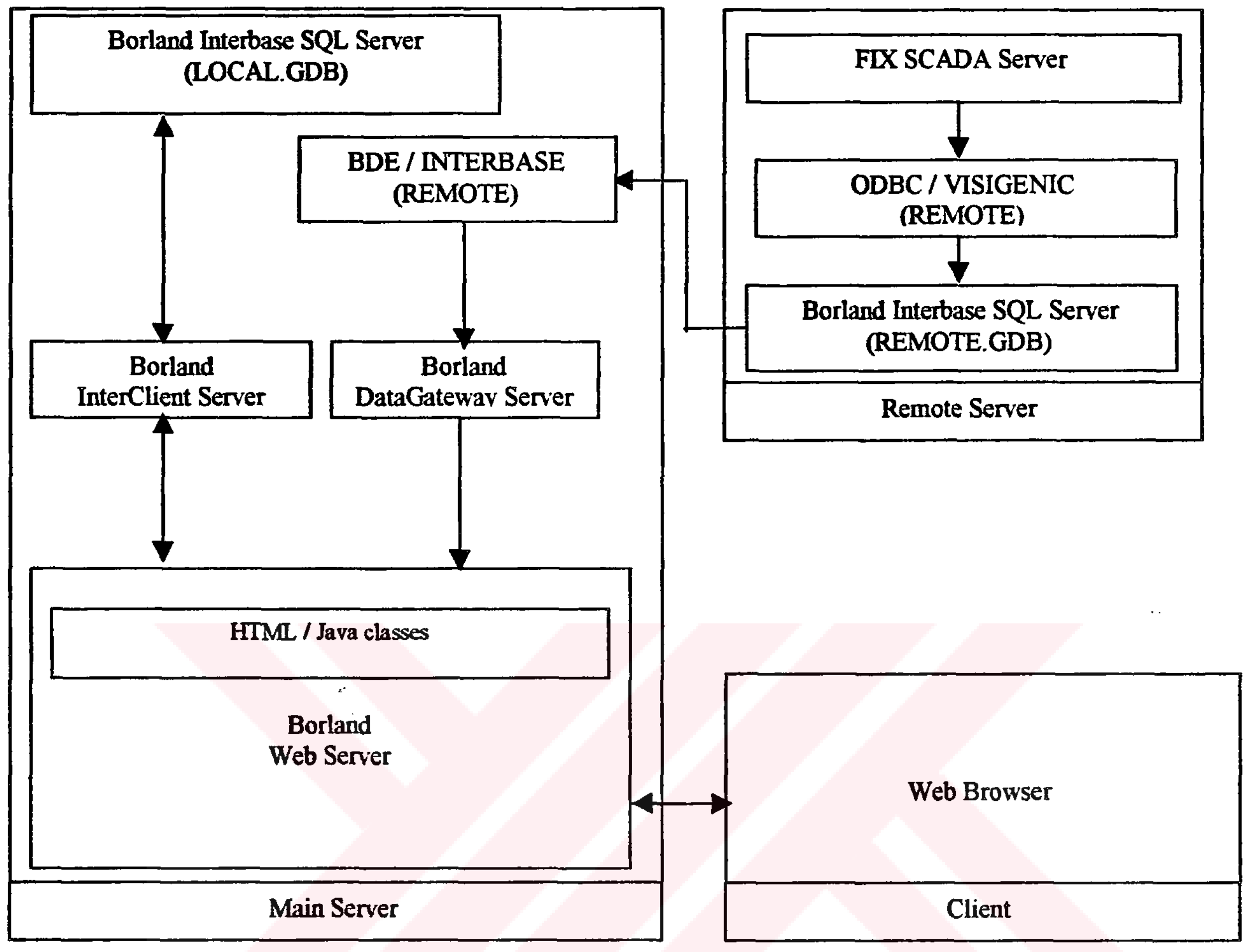

Borland Interbase SQL Server (LOCAL.GDB)
BDE / INTERBASE (REMOTE)
Borland InterClient Server
Borland DataGatewav Server
HTML / Java classes
Borland Web Server
Main Server
FIX SCADA Server
ODBC / VISIGENIC (REMOTE)
Borland Interbase SQL Server (REMOTE.GDB)
Remote Server
Web Browser
Client

Figure 5-1 Structure of the Project

Borland InterBase SQL Server that serves LOCAL database, Borland DataGateway Server, Borland InterClient Server, Borland Web Server run on the Main Server machine. HTML pages and java applets/classes were on the same machine too.

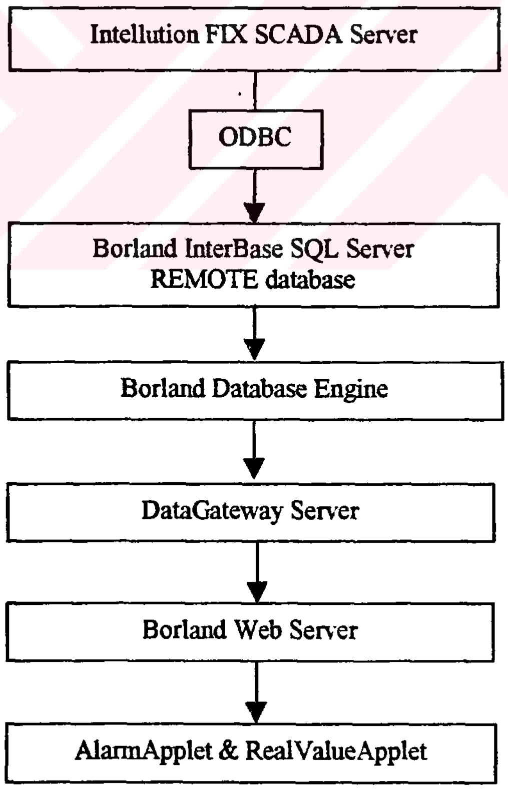
I called this database as LOCAL, because it reside on the same machine with java applets. However it is possible to install Borland InterBase SQL Server on another machine located different location and serve LOCAL database with that server to get more distributed environment.

FIX SCADA server collects information (alarms and real-time data) from production floor and write it to REMOTE database by using InterBase ODBC driver by Visigenic. Clients (or applets running on client machines) get data from REMOTE database by Borland DataGateway and Borland Database Engine.

5.3.4. Data Flow :

Remote database

1. FIX SCADA server collects real-time data from control systems at the production floor.
2. FIX SCADA server then sends that data to REMOTE database via ODBC protocol.
3. Borland InterBase SQL Server running on the Remote Server Machine serves REMOTE database



Intellution FIX SCADA Server
ODBC
Borland InterBase SQL Server
REMOTE database
Borland Database Engine
DataGateway Server
Borland Web Server
AlarmApplet & RealValueApplet

Figure 5-1 REMOTE database data flow

4. AlarmApplet and RealValueApplet instances loaded by clients receive data from Borland InterBase SQL Server running on the Remote Server Machine via Borland DataGateway Server and Borland Database Engine.

Local database

1. Borland InterBase SQL Server running on the Main Server Machine serves LOCAL database.
2. Borland InterClient Server serves LOCAL database.
3. Applets except AlarmApplet and RealValueApplet loaded by clients receive data via Borland InterClient Server.

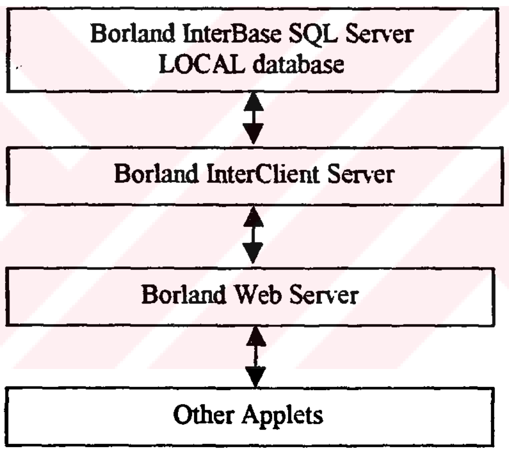


Figure 5-2 LOCAL database data flow

5.4. Common Problems

While developing the application, I encountered some problems.

1. A column that is the primary key of a table should not be assigned to ChoiceControl component. If done so, selected column cannot be displayed with ChoiceControl component.

2. Internet Explorer should not be used for displaying applets, If those applets were developed with JDK 1.2 and contains layout managers, e.g GridBagLayout. Netscape communicator 4.5 is better.
3. Data in GridControl components weren't refreshed automatically. repaint() method must be used to refresh.
4. Jdbc-odbc bridge should not be used with java applets. Jdbc-odbc bridge driver is suitable for java applications and only development phase of java applets. Othervise SecurityException occurs or URL in database connection statement couldn't be founded by clients.
5. Borland InterClient product can be used, If database and web server are on the same machine. If applets connect to more than one database on different servers, Borland DataGateway is the right choice.

CONCLUSIONS

This project examines database connection with Java applets via Internet. As companies continue to automate their business, the need to organize, maintain, and access electronic data increases.

With the growing popularity of Java, the network programmers on both the public Internet and private intranets desire to write one common interface for all of their organization's databases, portable to many platforms.

Java and the JDBC API is the future solution. Java will continue to be driving force in application development, and JDBC will be the connecting API of choice for robust database systems.

REFERENCES

Gosling, J., & McGilton, H. (1995). The Java Language Environment: A White Paper. Mountain View, CA: Sun Microsystems.

Hamann, J. (1996). Analysis of Java Database Connectivity and its Application in a Multi-Platform, Multi-DBMS Environment. Texas A&M University.

Hardy, H.E., (1997). History of the Net. http://info.isoc.org/guest/zakon/Internet/History/History_of_the_Net.html

Jamsa, K. (1996). Java Now. Las Vegas : Jamsa Press.

JavaSoft. (1998). The Java Tutorial. Mountain View, CA: Author.

Kehoe, B.P. (1992). Zen and the Art of the Internet. New York: Addison-Wesley Publishing Company, Inc.

Lemay, L., Perkins, C. H. (1996). Teach Yourself Java In 21 Days. Indianapolis : Sams Net.

Microsoft. (1996). ODBC-Open Database Connectivity. New York: Author

Microsoft. (1998). A Brief History of the Internet. New York: Author.

Niemeyer, P., Peck, J. (1996). Exploring Java. Sebastopol: O'Reilly & Associates, Inc.

Norberb, A. & O'Neill, J. (1992) A History of the Information Processing Techniques Minneapolis: Office of the Defense Advanced Research Projects Agency, The Charles Babbage Institue.

Shaughnessy, S. (1997). DataExpress Technology Overview. New Haven, CA: Borland International.

Sterling, B. (1993, Feb 13). The Internet The Magazine of Fantasy and Sicence Fiction, pp. 5-10

Sun Microsystems. (1997). JDBC Guide. Mountain View, CA: Author.

Tappendorf, S. (1995) ARPANET and Beyond

Visigenic. (1996). Developing Database Independent Applications with ODBC. New Haven, CA: Author.