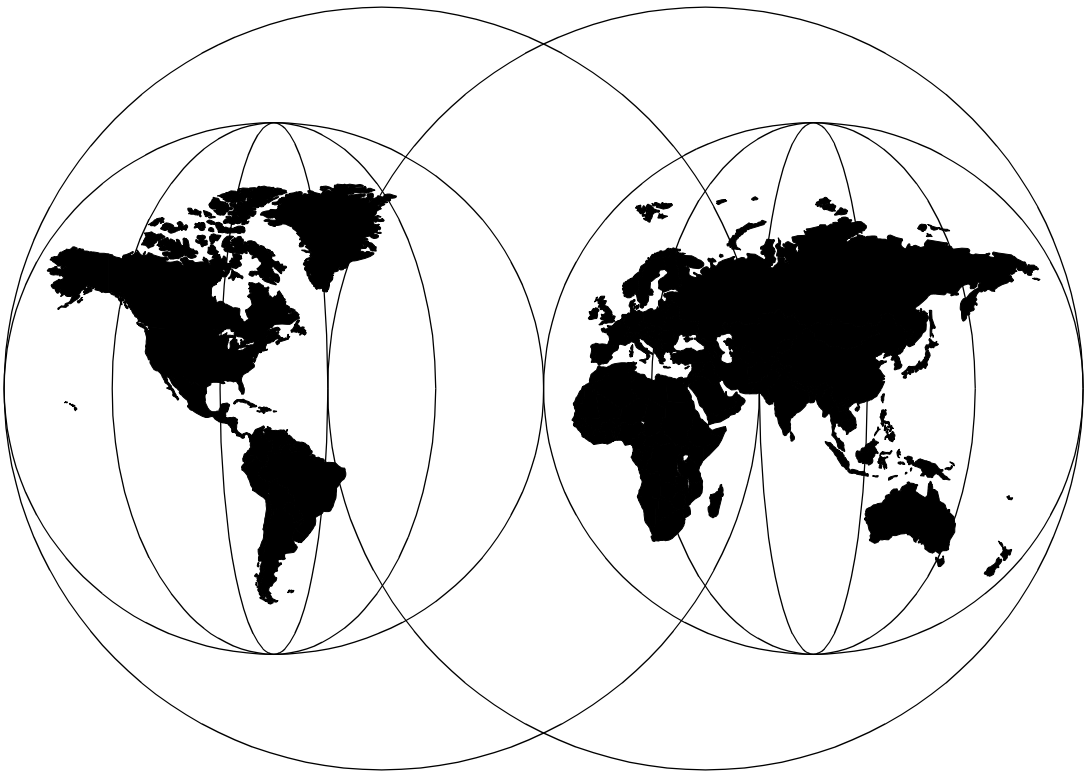
From Client/Server to Network Computing
A Migration to Java

Christophe Toulemonde, Anthony Button, Karen Harrison,
Jae Hyung Lee, Stephen Longhurst, Luigi Walter Sartori



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-2247-00

International Technical Support Organization

SG24-2247-00

From Client/Server to Network Computing

A Migration to Java

May 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix E, “Special Notices” on page 109.

First Edition (May 1998)

This edition applies to CICS Transaction Server for OS/390 Version 1.2, Domino Go Webserver Version 4.6, VisualAge for Java Version 1.0, DB2 for MVS/ESA Version 4, and DB2 Universal Database version 5.0.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

In summer of 1992, at the ITSO San Jose Center, an international team designed and implemented a client/server application to demonstrate how the client/server computing model can be implemented with IBM's mainline transaction and database products in the OS/2 and MVS/ESA environment. At that time, this application was using the best application architecture, development tools and techniques, and products.

Five years later, the new technologies, and especially the Internet, have strongly modified the computer industry and company environments. By combining the Web technologies with the vast resources of traditional information technology, companies around the world are shifting their business activities to the Internet. Business done on the Internet is known as *e-business*. The new application architecture use the Network Computing Framework. Java is the development environment for choice of the Internet, and new connectors exist to access the enterprise resources managed by CICS and DB2.

To gain the advantage of this new model and environment, companies face a choice to either create completely new applications to exploit the Net and allow them to do business on the Internet, or migrate and enhance existing applications to support the Web technologies.

In this book, we explore the migration path. In the summer of 1997, an international team at ITSO San Jose migrated the 1992 client/server application to the new architecture with three main objectives:

- To reuse the maximum amount of the existing application
- To open the application to the Net
- To add a new function to the application.

This book explains the complete migration process from the design phase to production, including the choice of the tools used in the migration. This book will help you to understand the different steps required in a migration process, what questions have to be asked and some of the answers we found in our environment. It also describes the different tools and technologies that we used to accomplish this work. In addition, it presents some alternatives to our choice that may need to be investigated.

This book is intended for technical professionals who are working in the area of designing and implementing network computing applications connected to existing enterprise systems.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization San Jose Center.

Christophe Toulemonde is a Senior ITSO Specialist for client/server and network computing at the Application Development and Data Management ITSO, San Jose Center. He writes extensively and teaches IBM classes worldwide on all areas of client/server and network computing. Before joining the ITSO,

Christophe worked as Technical Manager in an IBM subsidiary, Datablue, in France. You can reach him by e-mail at toulemon@us.ibm.com.

Anthony Button is a Technical Consultant for HUON, an IBM Business Partner. He has experience with all aspects of Client/Server development, from the client to the server and the communication mechanism in between. Anthony is also involved in a consultancy role, which includes working with customers to utilize new technologies and HUON solutions.You can reach him by e-mail at tony_button@uk.ibm.com

Karen Harrison is a senior IT specialist in the UK. She has 4 years of experience in the field of client/server development. She has worked at IBM for 16 years. Her areas of expertise include the life-cycle of application development from requirements and design, through build and test, to implementation and maintenance.You can reach her by e-mail at karen_harrison@uk.ibm.com.

Jae Hyung Lee is a ICP-ITS in IBM Korea. He has 20 years of experience in PSR, Field SE and marketing support. He holds a degree in Electronics Engineering from Korea University. His areas of expertise include transaction systems, and legacy system integration through Internet. He has written extensively on CICS transaction systems. You can reach him by e.mail at cpsjhl@kr.ibm.com.

Stephen Longhurst is a software developer working at IBM Hursley for the CICS/ESA Data Communications group. He is part of the CICS Web Interface development team and also implemented the MVS port of the CICS Gateway for Java. Stephen graduated in 1996 from the University of Southampton with skills in Java, TCP/IP and distributed systems. You can reach him by e-mail at slong@hursley.ibm.com.

Luigi Walter Sartori is an IT specialist in IBM Italy NCS Technical Support. He is especially experienced in database technology and in this area he has supported customers during the implementation of systems using DB2 (on various platforms), DB2 Parallel Edition and DataJoiner. You can reach him by e-mail at lwsartor@vnet.ibm.com

Thanks to the following people for their invaluable contributions to this project:

Eugene Deborin
IBM International Technical Support Organization, San Jose Center

Ueli Wahli
IBM International Technical Support Organization, San Jose Center

Alan Hollingshead
IBM Hursley Laboratories

Susan Malaika
IBM Santa Teresa Laboratories

Geoff Sharman
Transaction Systems Software, IBM United Kingdom

Thanks also to **Shirley Hentzell** for her editorial review.

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 131 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users`http://www.redbooks.ibm.com`
  For IBM intranet users `http://w3.itso.ibm.com`
- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. Foreign Currency and Traveler's Check Application

Back in 1992, the foreign currency and traveler's check application that we then called CS92 for Client/Server, was developed using the basic principles of the client/server model. The scenario was chosen for the following reasons:

- The simplicity of its business concepts, as most people have bought or sold foreign currency before going on foreign travel
- The generic components of its processes, which can easily be applied to any business that involves order and supply processing.
- Its benefits from a client/server computing architecture because of the organization of the business.

In this chapter, we explain CS92 by giving a brief outline of its functionality to allow you to understand the migration process described in the following chapters.

1.1 Business Concepts

We introduce CS92 by describing its operation in a fictitious bank, providing foreign currency and traveler's checks to customer. Based on the bank's organization, CS92 is used in the branches by the bank's clerks and in the central department at the bank head-quarters.

Bank branches supply currency and checks on demand, where demand justifies maintaining stocks at the branch. Otherwise, currency and checks are ordered from the central department and mailed to the customer or to the branch for collection. The branches also cash in traveler's checks and purchase currency.

The central department is responsible for maintaining appropriate stock levels centrally and at all branches. Maintaining these levels is very important for currency, because stock holdings cannot accrue interest, and insufficient stocks mean lost sales.

CS92 deals also with reconciling branch stocks against sales and purchases, exchanges rates, and calculation of branch and central department commissions.

1.2 Business Processes

In functional terms, CS92 covers all aspects of a bank's foreign currency and traveler's checks operation. The business processes involved are:

- Customer order management
- Cashier management
- Branch management
- Central bank management

All of the business processes were modeled, but only two subprocesses were fully implemented:

- Customer purchases from branch cashier

  This is the most frequently performed function in the application. Purchases are normally for the currency and traveler's checks of the destination country.

Payment for this service can be by cash, credit card, local check, or a debit to the customer's account.

Stock levels are reduced, a customer tab is printed, and accounting entries are passed to the accounts application.

- Branch stock replenishment

  At the end of each day, the requests of the cashiers are consolidated. Each request can be an order either to replenish stock or send excess stock. A consolidated branch order is then sent to the central departments.

In this book, we migrated only the customer purchases from branch cashier function.

1.3 CS92 Infrastructure

Based on the bank's organization, a three-levels system structure was implemented:

- The client workstation, used by the Cashier, running OS/2 2.0 with IBM Extended Services with Database Server for OS/2 Version 1.0 and CICS OS/2 Version 1.2
- The local server, located in the branch office, running OS/2 2.0 with IBM Extended Services with Database Server for OS/2 Version 1.0 and CICS OS/2 Version 1.2
- The mainframe server, located in the Head Office, running MVS/ESA Version 4.2 with CICS/ESA Version 3.3 and DB2 Version 2.3.

CS92 ran on a LAN-attached OS/2 workstation that is linked to an MVS/ESA mainframe server. In this project, the existing workstations, LAN, network, and mainframe were used in the hardware configuration. For software, CICS/ESA, DB2, CICS OS/2, DDCS/2, and IBM Extended Services with Database Server for OS/2 were used to demonstrate the client/server enabling capabilities of this software.

Figure 1 shows the CS92 system configuration.

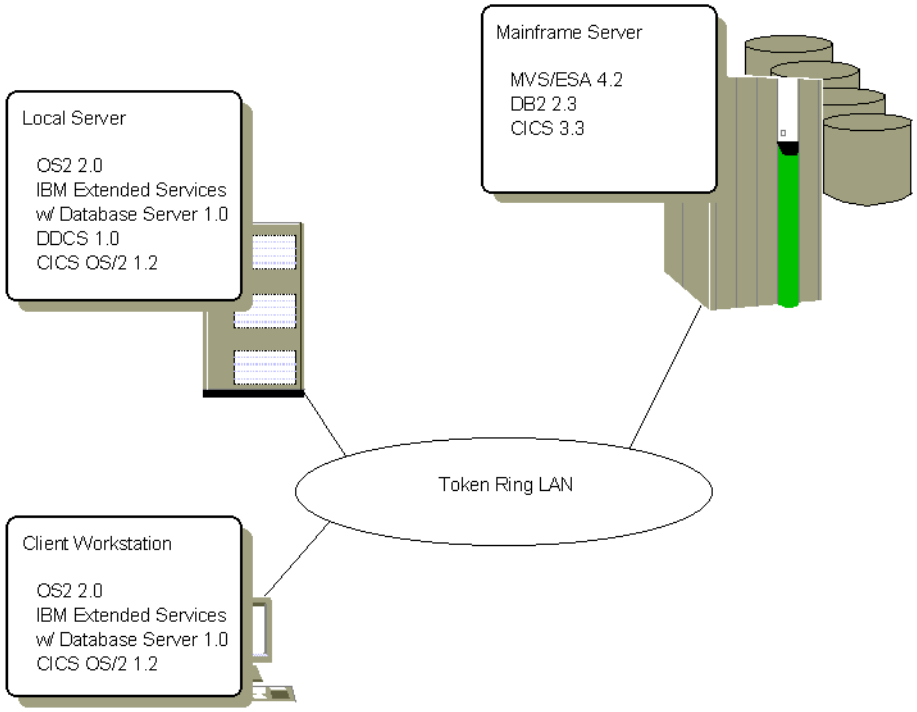


Figure 1. CS92 System Configuration

Although the application was implemented in an enterprise LAN environment using specific hardware and software, the application design using a client/server computing model makes it just as easy to implement the application in other environments, for example, a workgroup LAN environment, and using different hardware and software.

1.3.1 Hardware Configuration

The workstations on the LAN can be either client machines or local servers. They are connected via a communications controller to the mainframe server, which acts as an enterprise server.

The hardware configuration we used maps very nicely to the structure of the business described above. The functions provided by the enterprise server could be mapped to the functions provided by the head office. The functions provided locally by the LAN in each branch could be mapped to the functions provided by the branch. Each workstation on the LAN corresponds to the workstation used by an employee of the branch (for example, a cashier).

1.3.2 Software Configuration

CICS/ESA Version 3.3 and DB2 Version 2.3 were installed on the mainframe server running MVS/ESA Version 4.2.

Each workstation had the following software installed:

- OS/2 Version 2.0 with IBM Extended Services with Database Server for OS/2 Version 1.0
- CICS OS/2 Version 1.2

In the project the workstation had the full database manager installed; however, the client part of the IBM Extended Services with Database Server for OS/2 would have been sufficient.

The local server machine had installed:

- DDCS/2 Version 1.0 multiuser

The development machines had installed:

- IBM C/2 Version 1.1
- IBM C SET/2 Version 2.0
- EASEL Workbench
- Micro Focus COBOL/2.

1.3.3 Communications

LU 6.2 and NetBIOS are the two communication protocols used.

- LU6.2 was used for the CICS communications between CICS/ESA and CICS OS/2.
- LU6.2 was also used for the database communications between DB2 and DDCS/2.
- NetBIOS was used for the database communication between the Client workstation and the Local server.

Figure 2 shows the different protocols used by the database and the transaction communications.

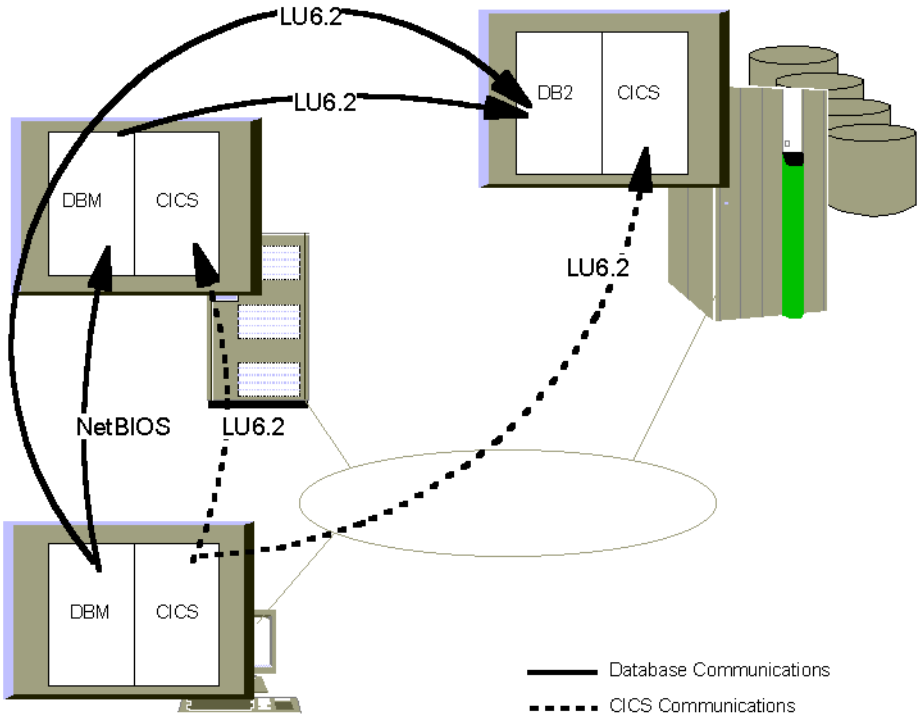


Figure 2. CS92 Communication Protocols

1.4 Detail Design

The key objective of the CS92 project was to show how the client/server computing model, using an online transaction processing (OLTP) application, could be implemented with IBM's mainline transaction and database products. It focuses on the following considerations:

Encapsulation
All access to data was through application code built around the data. The client application did not use dynamic SQL. If the data was resident on the same platform as the client application, data access was built as a separate server application so that client applications would not be responsible for data location or organization.

This encapsulation allowed code to be reused when new server functions were required that resembled functions already developed. The new functions inherited the existing, or base, functions.

Transparency
The project's application design and system design were independent of each other. For instance, services were known by their identifier only, and the service location was controlled independently of the application code. The control of service location introduced some directory services functions into the development.

Although the message routing function was developed at the application level, this layer was considered part of the network operating system layer and should remain transparent to the application developer.

Open Architecture
Access to the server was via C or COBOL programs only. Any program, not just CICS programs, may invoke a function, written in C or COBOL, that accesses services and data.

The services were developed to be easily portable to other platforms. Because each program was developed using modular programming techniques, the programming language standard set of calls was used and environment-specific calls were not allowed. It should be noted that client functions were not easily ported across platforms because the presentation components were environment dependent.

Any change to server functions would have affect only those existing or new client functions that require the function created by the change. There would be no need to amend or relink client applications that required the unchanged or previous version of the server function. This also applies when the decision is made to change server location.

Flexibility
The decision on how to split functions was made as late as possible in the design phase; minimizing the functions at the client reduces the likelihood of a client process disrupting the system. This thin client approach applied throughout this project.

Simple Validation
Data field validation was applied at the client and server functions: at the client to minimize network and server traffic in the event of simple errors, and at the server so that one server would not become dependent on the many clients correctly performing error handling

1.5 Data and Function Placement

Another important characteristic of CS92 is data and function placement.

1.5.1 Data Placement

Data placement was discussed in terms of these considerations:

- Data ownership, which is often the key factor when considering the distribution of data
- Level of sharing, whether by many users or just one
- Data partition between the different locations
- Target environment, based on the cost and the software architecture of the target hardware or the security facilities at the location
- Read/change data, including the transaction rates and the data manipulation profile to allow acceptable response times
- Volatility and currency of data
- Integrity, dealing with the coordination of updates in a logical unit of work.
- Systems management, such as backup of data, propagation of data, and data recovery.

From the above considerations, a decision was made for each entity as to whether to hold a single copy of the data in one location, to hold multiple copies of the data in multiple locations, or to distribute the data, as shown in Table 1.

Table 1. Table Placement

Table	Location
Currency	Client workstation, backup on the Local server
Currency denomination	Client workstation, backup on the Local server
Exchange rate	Mainframe server
Customer order	Local server
Customer order denomination	Local server
Customer order currency	Local server
Cashier	Client workstation and Local server
Cashier order	Local server
Cashier order details	Local server
Branch order	Mainframe server
Branch order details	Local server and Mainframe server
Customer	Local server and Mainframe server

Table	Location
Commission	Client machine, Local server, and Mainframe server
Control information	Local server
Branch	Local server and Mainframe server

1.5.2 Function Placement

The outline design of the application was produced from the requirement specifications. Each business process that was identified was then transformed into application tasks and mapped into application functions.

The decomposition of application tasks into application functions, which can then be implemented as self-contained modules, is key in the design of a client/server application. The application functions had to run on any platform, whether workstation, local server, or mainframe server, and the location of a function had to be transparent to the client part of the application. When the client required a service, it called the function that provides the service, but was unaware of where the service was coming from.

In CS92, functions that do not access data were placed close to the user. With functions that access data, the software products used enabled a flexible approach to function placement. Functions that access data need not be influenced by data placement.

The two examples below demonstrate the flexibility that is available for function placement:

- The cashier table, which contains the cashier ID and name, is held locally and the server function to get the cashier identity resides on the local workstation.
- The exchange rate table, which contains the buy and sell rate for each currency, resides on the mainframe server, but the function to retrieve the exchange rate data was implemented on the client machine. In this particular case, DDCS/2 enabled the local server function to transparently get the data from the DB2 on the mainframe server. No application code is necessary on the mainframe server for this to happen.

1.6 Application Coding

As shown in Figure 3 on page 8, the following programming languages were used to implement the different components of the application:

- EASEL was used to implement the presentation component of the application.
- The application component and the application request handler were implemented using IBM C SET/2 (with 32-bit addressing capabilities) to exploit the capabilities of OS/2 Version 2.0.
- IBM C/2 was used to implement the system server interface component to maintain compatibility with CICS OS/2 Version 1.2, which then supported only 16-bit addressing.
- To achieve the portability of the functions that provide the services to the application on multiple platforms, functions on the workstations and mainframe

server were implemented in COBOL/2, a programming language used in many business applications. Micro Focus COBOL/2 was used to implement the functions on the workstation, and IBM VS COBOL II was used on the mainframe server.

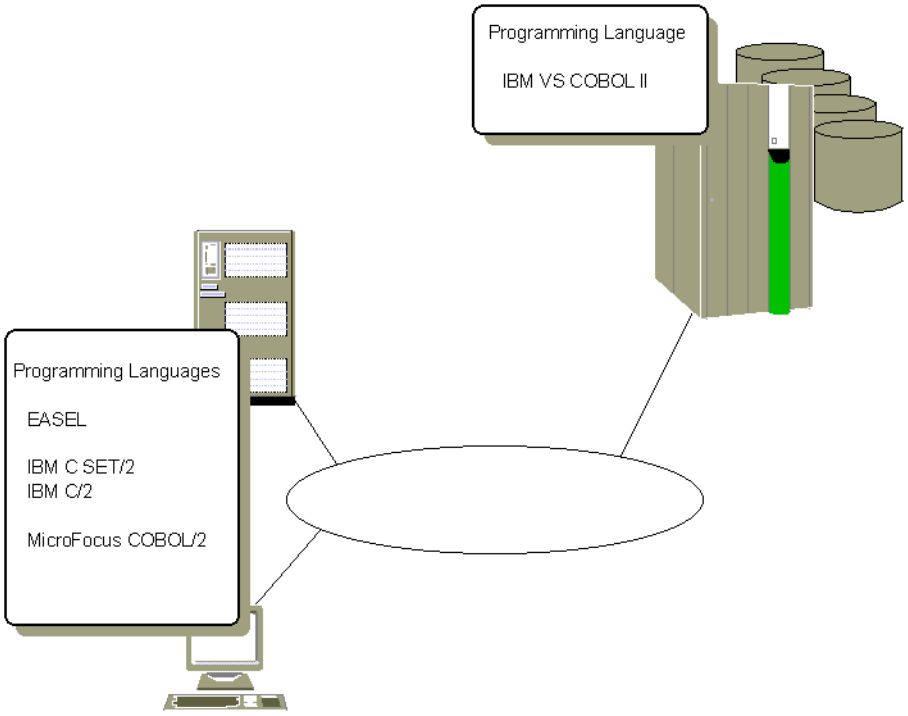


Figure 3. CS92 Programming Language

1.7 Graphical User Interface Design

The approach to designing the GUI consisted of the following steps:

- The processes in the process model that the end user will perform were identified. For example, in order management, the user will perform tasks to carry out such processes as order creation and order update.
- For each process, we identified the entity in the data model that the user would most naturally use to perform the business process. For example, the customer order entity was selected as the object with which the user interacts in order management. We recommended that a prototype of the GUI be developed together with the end user to confirm that the objects selected for the GUI are correct.
- For each entity involved, we identified the actions that can be performed on the entity. For example, actions on the customer order entity include:
  - New
  - Update
  - Delete
  - Print.

- We mapped each action of an object as closely as possible to a process. For example, for the customer order object,
  - New action was mapped to the create-order process.
  - Update action was mapped to the update-order process.
  - Delete action was mapped to the delete-order process.
  - Print action was mapped to the print-order process.
- Next we identified the user will need to work with on the GUI, beginning with the data for the tasks within a process. For example, in the create-order process, we identified tasks that access customer details, stock details, and input order details. All this information needs to be in the window. The information can be refined during prototyping.
- By carrying out the above steps for each object with which the user interacts, we defined the actions and the windows containing data that the user needs to work with. Prototyping could then begin.

Chapter 2. Network Computing Framework

The Open Blueprint is an architecture. Specifically, it is an infrastructure architecture that helps you to build an integrated system from multiple heterogeneous systems, including any product set that fits in.

The Network Computing Framework (NCF) is a skeleton based on the standards of the Open Blueprint. It identifies specific products and services, targeted to the delivery and support of network computing.

In this chapter, we introduce the (NCF). We describe the characteristics of the framework and the environment to implement the framework. We also describe the functions implemented in our sample network computing solution.

2.1 Open Blueprint

Open Blueprint is IBM's software architecture that integrates different forms of distributed computing such as network computing, client/server, and mobile computing on a common base. It provides a structure and set of technologies that enable heterogeneous systems to work together and form an integrated system.

In its easiest form, Open Blueprint is a simple chart that acts as a communication vehicle and checklist. In its complete form, it consists of a handbook and technical papers that a software designer would use to make applications on different systems interoperate.

Open Blueprint defines common functional building blocks and interfaces for distributed computing. These building blocks that consist of different software services are arranged in logical groups. This building block approach leads to modular, scalable systems that are easy to grow and enhance.

Open Blueprint is the common base architecture upon which specific solution oriented models can be built. The NCF has been developed using the Open Blueprint as the base.

2.2 Network Computing Framework?

The NCF is an open, standards-based framework for creating expandable e-business solutions. It's not a product, and it's not based on an IBM only philosophy (though we do offer complete, end-to-end e-business solutions). It's a network computing model, containing six key elements:

1. An infrastructure and a set of services whose capabilities can be accessed by open, standard protocols and a standard component interface, JavaBeans.
2. Clients based on a Web browser Java applet model that support universal access, a thin client paradigm, and exploitation of just-in-time delivery of components to provide a rich user interaction with the server.
3. A programming model and tools that create and exploit JavaBean components. As a result, any tool can produce a component to access any service.
4. Internet-ready protocol support, such as hypertext transfer protocol (HTTP) and Internet inter-ORB protocol (IIOP), which links JavaBean components.

5. A set of connector services that provide access to existing data, applications, and external services.
6. A set of built-in collaboration, commerce and content services as a foundation for an industry of partner-built solutions and customizable applications for e-business.

Figure 4 shows the NCF infrastructure.

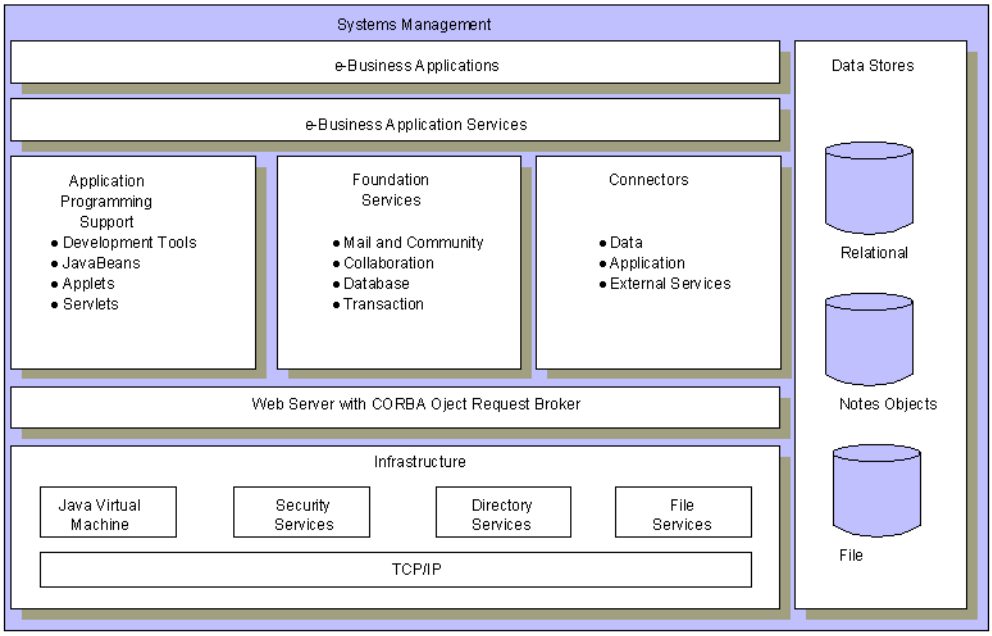


Figure 4. Network Computing Framework Infrastructure

2.3 NC Framework Components

The NCF has eight majors components, each of which is discussed in detail in the following subsections:

- Clients
- e-business application services
- Data and transaction connectors
- Application programming support
- Foundation services
- Systems management
- Web server with object request broker (ORB)
- Infrastructure with Java, directory, and security

2.3.1 Clients

The client element of the NCF provides universal access, a thin client paradigm, and exploitation of just-in-time delivery of components for seamless interaction between a user and information provided by the server. New NCF applications can be written as hypertext markup language (HTML), including dynamic HTML,

with Java applet extensions. Because the client model is based on standard browsing protocols (HTTP) and supports additional standard protocols such as IIOP, lightweight directory access protocol (LDAP), Post Office protocol (POP), and NetNews transfer protocol (NNTP), it offers access to key information and applications.

The NCF client model provides benefits in several key areas:

- A browser technology, used for distributing information and downloading applets from the network, making it easy to deploy, use, and manage new solutions. The NCF also provides several other client options based on user functionality requirements:
  - A simple browser with Java applet capability, so that Java applets can use the Object Request Broker (ORB) and IIOP to communicate with remote objects.
  - Domain-specific desktops that use dynamic HTML and Java applets to integrate local client applications with Web server access to the network.
  - Application-specific clients, such as e-mail.
  - A fully integrated client such as Lotus Notes, which provides rich collaboration and superior mobile support.
- A consistent programming model, which allows developers using Java to learn once and write anywhere. Having a consistent programming model among clients means that end users can reap the unique benefits of each of those devices while taking advantage of common Java and browser-enabled applications.
- Dynamic data exchange, implemented as JavaBean components. JavaBeans can dynamically exchange data using the Lotus InfoBus technology provided in the JavaSoft Java development kit (JDK).
- Support of mobile and fixed clients, from personal data assistants to network computers (NC) and from PCs to high-end workstations. The variety of mobile clients supported by the NCF operate either in disconnected environments or connected environments using wireless and other low bandwidth connections. These clients are supported through the use of replication technology. NCF mobile communication is enhanced with the use of compression, caching, and differencing algorithms.

2.3.2 e-Business Applications Services

NCF e-business application services are building blocks that facilitate the creation of e-business solutions. They are higher-level, application-oriented components that conform to the NCF programming model. They build on and extend the underlying NCF infrastructure and foundation services with functions required for specific types of applications. As a result, e-business solutions can be developed faster with higher quality.

One set of these building blocks is being defined to support the process steps used in electronic commerce solutions. This includes support of intellectual property management and secure business-to-business delivery of content as well as support of the secure electronic transaction (SET) standards developed by VISA and MasterCard with support from IBM and other leading companies. Additional e-business application services will be defined over time.

2.3.3 Data and Transaction Connectors

Any application that operates in support of an enterprise must access information and business processes on a variety of computers. The bulk of today's critical data and application programs reside on and use existing enterprise systems. Hence both developers and operations team need a tool set that facilitates connecting existing applications to the new ones. NCF connectors allow you to extend your critical corporate data, application, and transaction programs to the Web, linking the best of the Internet with the best of the enterprise:

- IBM CICS Gateway for Java brings open access to CICS from any Java-enabled Web client, such as a Web browser or a network computer. This Java applet can directly call CICS programs and data and run on any server platform that is Java-enabled.
- IBM CICS Internet Gateway provides an interface between a Web server and a CICS application using common gateway interface (CGI) scripts. It allows the conversion of 3270 datastreams to HTML format.
- IBM Net.Data allows Web developers to easily build dynamic Internet applications using macros. Net.Data Web Macros have the simplicity of HTML with the power of dynamic SQL. Net.Data provides database connectivity to a variety of data sources, including information stored in relational databases and flat files. Your data sources, such as DB2, Oracle, and Sybase, can be on a wide range of operating systems.
- Lotus Domino.Connect integrates the Lotus Domino server with a broad range of database, transaction, and enterprise application systems. Domino.Connect enables Lotus Notes clients and Web browsers to access enterprise data and applications, including enterprise resource planning systems, such as SAP R/3, traditional transaction processing systems, such as CICS and IMS, and relational database systems including IBM DB2 and offerings from Oracle and Informix
- IBM IMS Internet Solutions contain four solutions that provide connectivity from the Web to transactions in IMS databases. These solutions allow customers to match their communication infrastructures used with IMS to their Web server configurations.
- IBM MQSeries Client for Java enables Web browsers and Java applets to issue queries over the Internet to MQSeries for information stored in mainframe and legacy applications.
- IBM MQSeries Internet Gateway provides a transparent bridge between the Web and MQSeries commercial-messaging applications.
- IBM Host On-Demand provides fast and easy access to mainframe data from the Internet and intranets. Host On-Demand is a high-performance, low-cost solution for Internet users.
- DCE Encina Lightweight (DE-Light) Client, combined with an intermediate gateway, provides a solution for secure network transactions in distributed environments. DE-Light uses Kerberos and SSL security to support the integrity of information being processed in a transaction from Web browser to the server.

2.3.4 Application Programming Support

There are two concepts essential to understanding IBM's approach to this part of the framework.

One is the JavaBeans components. They are the core of the delivery model, both for client-side application support and also for server-side application programming, taking advantage of the development productivity provided by Java's reusability of objects. JavaBeans is the organizing principle of the NCF programming model. It is the platform-neutral, component architecture for Java to develop or assemble network-aware solutions. The JavaBean component model specifies how to build reusable software components, how the resulting JavaBeans describe their properties to visual tools for rapid application development, and how they communicate with each other.

Here are a few examples of JavaBean components:

- JavaBeans that provide easy access to relational and hierarchical data
- JavaBeans that integrate a business's existing transactional applications (CICS, IMS, and Encina) into new Web-based applications built within the NCF
- JavaBeans to access applications from vendors such as SAP, Baan, and PeopleSoft
- JavaBeans that integrate all of the Domino server classes into the NCF programming environment
- Lotus JavaBeans for embedding functions such as text processing, spreadsheets, charting, and calendaring capabilities into applications

The other concept is the assembly of the components. JavaBeans allows developers to create reusable software components that can then be assembled together using visual application builder tools, such as VisualAge for Java. In the NCF, application development is divided into three categories, summarized below:

- Content authoring tools, used to create and manipulate the Web-based application's multimedia content including HTML, graphics, images, animations, audio, and video
- Content assembly and management tools, taking the active contents and composing and scripting them together with other active or static contents to create a complete NCF application. This allows someone with neither sophisticated content development skills nor application development skills to create the final product.
- Integrated application development tools, including tools to support component coding and development such as Lotus Notes Designer for Domino or VisualAge for Java

2.3.5 Foundation Services

The foundation services are a subset of the Open Blueprint Application Services resource managers. They are products that provide essential services to e-business applications, but are not intrinsically e-business applications themselves. The services are:

- Mail and community services which provide e-mail messaging, calendaring, and group scheduling, chat, and newsgroup discussions. These services support standard Internet technologies and protocols and are accessible from all standards-based clients.
- Collaboration services providing groupware and workflow support. They integrate information from any resources, manage how much and what is

shown to the user, allow users to interact with the information, and manage the content.

- Relational database services delivered through IBM's Universal DB2. They provide a facility for managing both the operational data and multimedia content. They leverage existing business logic to build new Web-based applications, support development and execution of database application programs, stored procedures, and user-defined functions using Java. They support the SQL and the Java database connectivity (JDBC) interfaces and integrate transaction services using the XA model.
- Transaction services extend the Web by providing a transactional application execution environment. NCF transaction services support development of both procedural and object oriented transactional application program.

2.3.6 Web Server with Object Request Broker

Just as Java and component-based assembly are central to the application development strategy for the framework, so is the Web server a powerful point of integration in the NCF. Electronic networking businesses today presupposes the use of the Web and therefore a Web server.

The Web server is the entry point to server functionality supporting HTTP and IIOP requests from NCF clients, locating and invoking business logic on the logical midtier server. The Web server provides the programming environment for transactional business applications, linking the Web to the existing enterprise application, data and systems.

2.3.7 Infrastructure with Java, Directory, and Security

The basic infrastructure services are Java, directory, and security.

For both clients and server, the Java Virtual Machine provides the base support for programming. It also provides platform independence for e-business solutions

Directory services that locate users, services, and resources in the network are accessed through standard LDAP protocols and interfaces.

Security services support user identification and authentication, single sign-on, access control to resources and services, firewalls, confidentiality, data integrity, nonrepudiation of transactions, security management, key recovery, and Java security. They are based on robust public key support and certificates.

2.3.8 Systems Management

Finally, there is the fundamental assurance that all this can be managed. NCF systems management encompasses systems, network, and applications management. The NCF provides for the unique management requirements of network computing across all elements of the system including applications, services, infrastructure, and hardware. The basic Systems Management model has four basic categories:

Development: The IBM products included as part of the NCF already have the necessary interfaces to facilitate management. Part of the NC Framework is to provide toolkits that enable customer-written applications and services to also take advantage of the same management services

Deployment: Once applications have been developed, it is essential to ease the process through which they are delivered to the Internet servers.

Operations: This provides support to the WebMaster for statistics, performance tuning, load balancing, along with more traditional backup, configuration management, security services. All are part of this aspect of systems management.

Content: These tools provide support for the visualization of and management of the objects, links, and files that constitute the electronic business on the Web.

Chapter 3. From Client/Server to Network Computing

In this chapter, we explain the evolution of the computing model. We start from the characteristics of the client/server application. Then, we analyze their impact, or their modification in a network computing environment. We, also describe the benefits the new model can provide. Finally, we discuss how the client/server application building blocks can be used as is or modified in a network computing application

3.1 The Client/Server Model

The client/server computing model for distributing applications is a logical model that defines the existence of a client (requester of a service) and a server (provider of the service). A service can entail data, a function, or any resource that can be shared. The server can be local or remote. Usually, the server is remote, and there is thus a need for communications to connect the client and server (or servers).

The main evolution between the client/server model and the networking computing model is in the communication protocol used and the definition of the client:

- Today, TCP/IP provides the ability for corporations to merge differing physical networks while giving users a common suite of functions. It allows interoperability between equipment supplied by multiple vendors on multiple platforms, and it provides access to the Internet. In fact, the Internet, which has become the largest computer network in the world, is based on the TCP/IP protocol suite.

  Unlike the Internet, the intranet has evolved recently, consisting of TCP/IP networks that are entirely under the control of a private authority or company. Those intranets may or may not have connections to other independent intranets (which would then be referred to as *extranets*) or the Internet. They may or may not be fully or partially visible from the outside depending on the implementation.

- The thin client, as presented by NCs, offers a range of simple, economical alternatives to terminals and underutilized personal computers. NCs extend access to network applications, intranets, and the Internet while they lower the total cost of ownership with reduced up-front expenses and significantly less need for support. They are simple to install and easy to use and manage. With the "install once and run everywhere" nature of network computing, including the growing availability of Java applications, NCs make the business desktop easier to manage and support.

  With NCs, there are no longer any local services on the client machine. There is no requirement for any storage on the client machine: the application is dynamically downloaded at execution time, and data is not stored on the client machine.

3.1.1 Distributed computing

The client/server computing model has evolved to a distributed network computing model. This model can be viewed as a single multiuser computing system that functions and performs like a traditional host computer but is built

from a set of discrete machines, interconnected by a very high speed, reliable mechanism. The whole structure is transparent to the application. Services can easily be assigned to server machines on the basis of the changing requirements of users and the changing capabilities of server machines.

In general hardware terms, the application uses workstations or NCs, network connected servers and enterprise mainframe servers. They each run on different platforms but are all interoperable since they are linked by the network, thus maximizing the processing power of each machine. In the network computing model, the concept of distributed computing is extended further, as applications can be distributed to any client machine that has a Web browser, without any preconfiguring of the network.

3.1.2 Transparency

When CS92 was developed, a significant characteristic of the client/server model was that it provided transparency to applications and end users, leaving them untroubled by the complexity of the overall system. Each part of the system encapsulated its own complexity and communicated with the remainder of the system and users by presenting a simple view.

The following features have transparency considerations:

- User view

  The CS92 user saw the application as a single coherent application that provided the means of performing the business functions. The user did not need to know how or where a function or resource is provided.

  User view transparency is the same in network computing: a successful migration should mean that the user is unaware of the change. This is one major requirement for our migration project.

- Location

  The CS92 user was be able to access applications and data without regard to their location in the distributed system. Programming interfaces for accessing data and services enabled requesting programs to be independent of the location of data or services.

  This independence is particularly true of the client in the network computing model: the client can be located anywhere, on the enterprise network, or even connected through the Internet (if there is no security exposure) and still be able to access the application. More, the client can be any kind of hardware, from NCs to PCs, because most of the load is supported by the servers.

- Vendors

  Both client/server and network computing enable machines from different vendors to interoperate. Network computing simplifies the interoperability as there are no specific hardware requirements for the client machine.

- Network independence transport

  When CS92 was developed, many different networks had to be crossed to access data and applications. This diversity of networks meant that applications had to use communication interfaces that insulated them from the protocols of the particular network transport that they used.

In our network computing migration, we standardized on one network protocol, TCP/IP, which facilitates the communication between the servers, the subsystems such as DB2 or CICS, and also the programming of the application.

- Distributed applications and service enablers

  A basic requirement in both client/server and network computing is that the application developer should be able to concentrate on the implementation of a function, rather than on how to make the function accessible to requesters anywhere in the distributed system.

  In our migration, this requirement is satisfied because the application is downloaded dynamically, on the user's computer, at execution time. The enterprise gateways, in our case the CICS Java Gateway and JDBC, enable clients to connect to the enterprise services.

- Performance

  Clearly, every application requires adequate performance. In CS92, it was anticipated that when function and data were accessed directly from the workstation, performance would be improved. However, no performance data was gathered.

  In the case of network computing, all accessing of function and data is by way of the network. It is not yet clear whether the costs of the network times will be outweighed by the benefits of the powerful mainframe processors, improving performance time. It is beyond the scope of this redbook to analyze performance statistics.

- Universal distributable directory

  A directory is required that provides universal naming and naming consistency so that information about resources and facilities in a distributed environment can be managed and located. That is, code on the client does not need to know the identity and location of the server it is making requests of, but some facility does need to know. In the foreign currency application, code was written on the client machine to satisfy the requirement. However the developers acknowledged that this was not a full solution.

  Network computing does not yet have directory services, but does have a common naming standard, the universal resource locator (URL) which partly satisfies the requirement.

- Data Conversion

  In a centralized environment, the entire application is contained in one system and there is only one data representation on that system. For an IBM mainframe, the data representation is EBCDIC, and for a workstation, it is ASCII

  As soon as data is exchanged between systems, however, it is important to know whether the systems that exchange data have the same type of data representation.

  If not, data conversion is required. Several approaches to data conversion exist in the industry. One approach is to convert data to a common known representation. Another is to convert incoming data to the representation known by the local system.

Whatever approach is taken, the application should be shielded from the data conversion. The conversion of the data should be taken care of, for instance, by a transaction manager or resource manager so that neither the conversion nor the data representation will affect the application.

The DFHCNV macro of CICS was used by CS92 to convert data on its arrival at the mainframe.

In the case of our migration to a network computing solution, the data conversion was performed by the VisualAge for Java code, thus making it transparent to the application code.

3.2 Network Computing Benefits

CS92's developers described some benefits expected to be achieved through developing a client/server solution. Here we discuss those benefits in terms of what client/server computing provided and what network computing can provide:

- Cost-effective solution (right-sizing)

  When the original solution was developed, it was sometimes thought that the best way to right-size was to place a significant proportion of the processing on the inexpensive workstation.

  For certain types of application, the current thinking is that a right-sized application has most of its processing on the mainframe, because a cost-effective solution is not only made up of the capital costs of the equipment, but also the operating costs of the system. Network computing follows this philosophy.

- Provide graphical user interfaces to users

  The use of workstations is a cost-effective way of providing a GUI to users. Both client/server and network computing applications can be designed such that the user interface processing is performed on the workstation. Well-designed GUIs for appropriate applications provide significantly higher end-user productivity than those that run on traditional 3270 screens.

  A network computing application can also build a dialog using HTML on a Web page or a Java applet. Developers have the choice of building a GUI dialog or an HTML dialog, depending upon which is more appropriate for the application.

- Organization structure mapping

  In CS92, the bank consists of a hierarchy of semi-independent units that share common information, for example, commission rate. The lowest units in the hierarchy are the cashiers who have their own data, such as personal stock levels. The cashiers require services unique to them such as creating new customer orders. At the same time, they also require data and services held higher in the hierarchy: exchange rate is an example of this. Dealers maintain the exchange rate at the central office because this data is dynamic and the latest rate must always be applied.

  The branch office is in the middle of the hierarchy. It provides services as a response to the cashiers' requests to provide, for example, replenished stock. In turn, however, it acts as a client in relation to the central office, for instance, by making requests to return excess currency.

The central office is at the top of the hierarchy. It provides services to both the branch and the cashier. The three levels of the bank illustrate, in the client/server model, in which a client is a component that requests services, and a server is a component that satisfies requests. The example of the branch office shows that a server can also be a client of another server. The hierarchical structure is different from peer-to-peer structures, and maps more logically to the organization of a business.

The network computing framework maps more closely to the way in which modern business is evolving. The trend is for one enterprise's computing system to communicate directly with that of another enterprise: the connection is made without any regard to the organization of either business. In a similar way, when individual customers browse an enterprise's home page, and possibly order goods or services, they do not care which department or level of the organization is dealing with their order.

- Exploits LAN infrastructure

  When CS92 was developed, many enterprises had a LAN infrastructure with many workstations installed. Client/server applications can successfully use such a structure.

  This usage is not such an important consideration for network computing: providing that a workstation can communicate with its server machine, using TCP/IP, it is not significant whether a LAN infrastructure is used or not.

- Improves availability

  The infrastructure of a client/server application can be used to improve availability. For example, if availability is crucial to the application and sufficient resources are available, then several servers can be defined to perform the same functions. Single points of failure are avoided by running these servers on different machines. When only one server's providing a particular function, the function can be switched transparently to an alternative server if the first one fails.

  Network computing is rather more dependent upon the network: if the network goes down the application cannot run. However, networks are becoming increasingly more robust.

- Enables heterogeneous interoperability

  Client/server applications are based on accepted industry standards and allow for heterogeneous interoperability, so that systems from different vendors can work together to execute applications.

  Java is evolving into the industry standard in which to write network computing applications. This evolution is based on popularity and actual usage partly because of its ability to "be written once, run anywhere."

3.3 Building a Client/Server Application

In this section, we describe some features of a client/server application, and discuss some considerations when building. We also discuss the applicability of these characteristics to a network computing application.

3.3.1 Basic Communication Models

There are three communications models that can be implemented with varying degrees of success by client/server or network computing:

- Conversational model

  In the conversational model, a conversation is a series of synchronous message exchanges between two partners that have a peer-to-peer relationship. Both partners must be active for the duration of the conversation. Either partner can initiate and direct the communications, and both sides can send or receive data. In general, a receiver cannot send data until the sender surrenders that right.

  The conversational model is powerful but may make the programs more complicated. Both sides have to know the logic of the conversation, and they have to agree on the state of the conversation at any given time. From a programming point of view, the two sides of the conversation are not transparent to each other, although they may be to the end user.

  This model is difficult to implement in a network computing application because the server is generally unaware of the state of the browser especially when using HTML. However, there have been some enhancements to the middleware, such as CICS, to improve this conversation support

- Call model

  In the call model, a program sends a request to another system by specifying a program or function to be executed. The process is synchronous to the requester, which waits for the function to complete. The called system executes the requested program or function and returns the results in a predefined format to the calling program. The call format includes parameters for the passing of data.

  The routing of the request and the establishment of communication with the called program are transparent to the caller.

  This is the model used by the existing client/server application. It is the model implemented by TCP/IP, and so is the model used by our migrated application.

- Messaging model

  *I*n the messaging model, a single message is passed from one program to another by placing the message on a queue. No response is required from the client. The message is not necessarily processed instantly. There can be a single queue or multiple queues for different message types and priorities.

  The messaging model, as implemented by MQSeries, is recommended for asynchronous processing but was not used in our migration to a network computing solution.

3.3.2 Application Characteristics

When we migrated the application from the client/server model to the network computing model, we had to consider the characteristics of that type of application, to ensure that the new model maintains, or improves upon, the support for that type.

In CS92, an online transaction processing application is critical to the business of the bank. Hence the following requirements have to be satisfied:

- Rapid response times

- High availability
- High volume transaction rates
- Robust data integrity

3.3.3 Security

Security is a critical issue in any application. Only authorized users should be able to access applications and data. In centralized applications, access control to resources is often managed by a resource access control manager through user IDs and passwords. Preferably, only one resource manager should exist for maintaining and checking access control.

When applications and data are distributed over more than one system, the task of securing applications and data becomes more complex. In the first place, more than one security resource manager is involved, and heterogeneous environments can involve different resource managers. These different resource managers must be able to interface with each other. In the second place, the maintenance of user IDs and passwords has also become more complex in a distributed environment.

Security can be divided into the following three components:

- Authentication

  Authentication is the first security step. When clients communicate with servers, they must identify themselves and the server must verify that the clients are who they claim to be. The authentication function should be contained either in the security subsystem or in the communications subsystem. Several solutions related to authentication exist in the industry.

- Authorization

  After authentication has taken place, it is necessary to check whether the client is authorized to access the resources it is asking for. The authorization check is done through the use of userids and passwords. First, the user ID and password are used on the client machine to check whether an end user is authorized to use the client application. Then, the client application will ask for services in the network, and the user ID and password are sent to check whether the client is allowed to access those services.

  In an intranet environment, authorization is still possible using user IDs and passwords managed by a systems administrator, because the user base is still known. When a network computing solution is implemented across the Internet, using user IDs and passwords to check for authorization becomes much less realistic, and considerable attention must be given to determining which external users are to be allowed to execute enterprise transactions, and to view and update enterprise data.

- Encryption

  To secure the transmission of data through the network, data should be encrypted. Again, the encryption function should be carried out by the communications subsystem or the security subsystem. The application programmer should not be bothered with this function.

3.3.4 System Management

System management is crucial for effective operation of any application. There are particular considerations when the application uses a client/server model. Some of these considerations are simplified when the application uses network computing. The following discusses some of these considerations:

- Network

  The network is essential for client/server or network computing applications. The system management tasks include ensuring the continuous availability of the network, acceptable performance of the network, in both response times and throughput. For client/server and intranet solutions, usage of, and access to, the network must also be managed.

  A network computing application, using Java, may also need to handle more throughput, especially in the initialization phase of the application, when it is being downloaded from the server to the workstation.

- Software distribution

  Software distribution is complex in a client/server environment. Version control between systems is required to ensure software integrity. Rollout must be coordinated to ensure that the application comes on-line consistently across the enterprise and that the rollout has no impact on the user

  To distribute the new or updated software, system administrators define packages that contain data files, programs, system software, and script. They then define the distribution jobs by specifying package, site, execution time, and query. Installation is then by script driver, triggered by user logon

  Software distribution becomes much more straightforward in network computing because there is no longer a need to distribute application code, or system software, such as DB2 or CICS, to client workstations. The code to be dynamically downloaded can be held in a single, production-level library which requires only the well-established disciplines of library management and version control. The only software that has to be loaded on the client workstation is an appropriate Web browser.

- Problems and changes

  Problems do occur in applications: they need to be reported, investigated, fixed, installed, and tracked throughout their lives. In a client/server environment, users can easily report a problem to their support team. The support team, however, may have much greater difficulty in analyzing and reproducing the problem, as they are unlikely to have an identical environment in which to run the application. When a fix, or a change, is made to a client/server application, the installation faces the difficulties of software distribution.

  In an intranet network computing environment, users can still report problems to their support team, and now, this team's investigations should be much simpler as there is only one environment, and one level of code to analyze. Similarly, the installation of fixes and changes needs to be done in only one place.

  The mechanics of problem management are the same in an Internet network computing environment. However, culturally, the raising of problems may change, leading to discussions over the following questions:

- Will an external user make the effort to report a problem, particularly, one of low severity?
- How will the external user report a problem?
- Do external users have to prove their credibility?
- What will persuade a company to fix perceived problems?

3.3.5 Data Management

The tasks required to manage databases include reorganizing the data, backing it up, control, performance monitoring and tuning, capacity planning, housekeeping, and access control.

The operational success of an application depends in part upon the accuracy, consistency and completeness of its data. Here we look at the subjects of backup and recovery, and data integrity in relation to client/server and network computing applications

- Backup and recovery

  In a centralized environment, backup procedures are often implemented and maintained automatically. In a client/server environment, the backup of data on client machines cannot easily be done in this way. Users are unlikely to back up the data on their own machine in a systematic disciplined way, and it cannot even be guaranteed that users will either always turn off their machines, or always leave them on to enable remote access.

  In network computing, the databases are no longer distributed to client machines, so the required data management tasks do not have to be replicated across remote environments. Backup and recovery can again be centralized, because no data is stored on the client machines.

- Data integrity

  Data must be kept in a consistent state, that is, it must have integrity. The developers of the client/server application were unable to ensure such consistency, because one business transaction was implemented by two logical units of work, one updating data on the client machine, the other updating data on the server machine. At the time, the two-phase commit protocol was not implemented by system software across different platforms, which would have ensured that either both updates were successful or neither was. The problem could not be solved by combining the two updates into one logical unit of work because the level of distributed relational database architecture (DRDA) used did not support updates across platforms.

  With network computing, data is typically held on only one platform, so there is no longer a problem with cross platform updates, and normal application design should determine what constitutes a logical unit of work.

3.4 Architecture: Three-Tier Model

CS92 embodied the principles of the three-tier model. The model can be used to describe the hardware configuration, the software configuration, and the application design. Some aspects of the three-tier model can still be applied to a network computing solution, but the model is no longer essential for defining the architecture.

3.4.1 Hardware

CS92 client/server implementation ran on LAN-attached OS/2 workstation. There was a local server machine on the same LAN. The LAN was linked to an MVS/ESA host. Depending upon the configuration of the workstation, it could communicate with the MVS/ESA host either directly or through the local server.

In migrating the application, we used the same basic hardware setup. However, network computers could have been used instead of the local workstations. This is because, with the network computing model, all the resources are stored on the server and downloaded to the workstation only when required. There is no need to store resources on the local workstation.

3.4.2 Software

In CS92, system and application software was loaded on each tier to support the data and function placements. Hence, the appropriate versions of CICS and DB2 were installed on each platform. Application code was also installed on all three tiers. LU 6.2 and NetBIOS were the protocols used to communicate between the tiers.

In migrating to network computing, CICS and DB2 are no longer needed on the client machine, instead a Web browser must be installed, and the communication protocol is now TCP/IP. The application code is only installed on the Internet Web server.

3.4.3 Use of the Tier Model in Migration

We largely adhered to the three-tier model when we migrate CS92. The client became skinnier when data was no longer stored on it, and therefore, there was no requirement for a database management system (DBMS). As part of a network computing solution, application code is not stored permanently on the client, reducing it even further. In our implementation, there was no need to communicate directly between the local server and the mainframe. However, neither the thinness of the client nor the absence of a link between the other two tiers abandons the three-tier model.

In more general terms, the three-tier model is not essential for considering a network computing solution. There are a number of advantages in dropping the mainframe or the local server, with their associated hardware and software configurations. From a systems management view, it is simpler to have only two hardware components: client and mainframe, or client and server. Also from a systems management view, both installation and maintenance of software and application are more straightforward. When data is stored in only one place, then its design is easier because placement and duplication are no longer issues that have to be decided. In terms of application logic, code placement questions are more easily answered, and logical units of work that update data cannot be spread across more than one platform.

There are advantages in keeping a three-tier model. For an existing application that already uses this model, it can be preferable to maintain the existing setup. Retaining a mainframe and a local server can maximize the flexibility and transparency of a system, so that an application can switch from running on one platform to running on another. This is particularly relevant if the application is a generic solution that can be run on multiple systems. If, however, the application

is custom tailored, the systems management costs might outweigh other perceived benefits of switching the application from one server to another.

Chapter 4. Network Computing New Environment

In the following sections, we describe briefly the new IT environment that helped us to migrate the application. We concentrate on the products and features that we used in the migration.

4.1 Java

Java is not just a programming language, it is a platform that can be used to create and deliver information over any network. Java is portable and architecture-neutral for any system implementing the Java Virtual Machine. Programs written in Java can be distributed and executed on any client system that has a Java-enabled Web browser. Developers are freed from having to write multiple versions for multiple platforms and even from having to recompile for each platform. Because the applets run on the client system, scalability and performance are no longer tightly tied to Web server systems. High performance requirements are also addressed through multithreading, the ability to link native code, and the just-in-time compiler.

Java is portable and flexible. In the past, we have had languages that claimed portability, but none delivered anything remotely resembling portability. Developed from the ground up as object oriented and portable, Java has already addressed the biggest challenge. Companies now are free to let departments select the hardware platform of their choice without worrying or having to budget for integration and porting costs.

Java applications are implementation-neutral. We often get caught up in the fat client, thin client rhetoric. Because Java is truly object-oriented, you are free to distribute the application business logic in the manner that best fits your needs. Modifications are made easily and transparently as business needs change.

Although Java is most often described as a programming language, its more unique capability is its platform independent virtual machine facility. The excitement of Java is that it combines an object programming paradigm with the client/server distributed architecture and the broad connectivity of the Internet, while solving many of the problems industry has encountered with client/server software and hardware management.

Java is an interpreted object-oriented language, similar to C++, which can be used to build programs that are platform independent in both source and object form. Its unique operational characteristics, which span Web browser and network computers as well as servers, enable new client/server functions in Internet applications while enforcing a discipline that make software management possible across almost any hardware platform.

To achieve platform independence, the Java language allows no platform-dependent operations and it excludes some C++ functions such as a preprocessor, operator overloading, multiple inheritances and pointers. All Java programming is encapsulated within classes, and the development toolkit includes an extensive set of classes. These special classes, which are critical to assuring platform independence, include GUI functions, input/output functions and network communications.

The Java compiler, however, does not generate machine code. Instead, it generates intermediate code called Java *bytecode*. This bytecode is interpreted by the Java interpreter, which executes the instructions on the particular hardware platform. The Java interpreter and run-time system are collectively called the Java Virtual Machine or JVM.

The interpreter also inspects the bytecode at execution time to ensure its validity and safety to the machine environment. The isolation the Java interpreter provides, coupled with the Java run-time systems provided by vendors, create a a platform-independent virtual machine environment.

The Java language can be used to construct Java applets and Java applications.

4.1.1 Java Applet

A Java applet is a small application program which is downloaded to and executed on a Web browser or network computer. A Java applet typically performs the type of operations that client code would perform in a client/server architecture. It edits input, controls the screen and communicates transactions to a server that in turn performs the data or database operations.

Invocation of applets occur through the use of a new applet HTML parameter as shown in Figure 5.

```
<applet code="ibm.cics.jgate.test.TestECI.class" width=520 height=290
codebase="/jgate/classes">
</applet>
```

Figure 5. Applet Tag

This tag is used within HTML pages to indicate when applets are to have control and to specify the display area to be used by the applet. When a Java-enabled server is downloading a page and encounters this tag, it also downloads the applet bytecode in the same way it downloads an image referenced by an HTML image tag. The Java-enabled browser, such as Netscape Communicator or Microsoft Internet Explorer, then interprets and executes the applet bytecode. The applet may edit screen input, generate screen output, and communicate back to the computer from which it was downloaded. An example of applet processing would be an applet in constant communication with a server to receive stock trade information which it would update in a window on the screen. Multiple applets can execute concurrently.

The downloading of applets should not have a significant performance impact on the response time to end users since the applets are typically not very large. In fact, applets, by performing processing on the browser or network computer, can improve the overall browser performance by eliminating iterations with the Web server. Note that also, just as images are cached in Web browsers, applets are cached and this minimizes the frequency of applet downloading. A current performance consideration is the iterative compiling of the Java bytecode at the time of execution. This, however, is rapidly being addressed by the industry and is losing its importance.

4.1.2 Java Application

A Java application is a program written in Java that executes locally on a computer. It allows programming operations in addition to those used in applets which can make the code platform dependent. It can access local files, it can create and accept general network connections, and it can call native C/C++ functions in machine-specific libraries.

4.1.3 JavaBeans

JavaBeans is a component model for Java. The intention of JavaBeans is to define a model that enables suppliers to create and ship Java components that can be composed together into applications by end users. JavaBeans is a reusable piece of code written in Java. After a JavaBeans function is built, it can be shared, sold, or reused. The whole idea of JavaBeans promises to deliver vast reusable libraries of functions that can quickly and easily be assembled into new applications to meet urgent business requirements. The JavaBeans specification means that application development tools, by adapting to the specification, can now freely interchange beans without code modification, thereby greatly increasing your ability to choose the tool that best suits your needs.

4.1.4 VisualAge for Java

JavaBeans, true object-orientation, and the ability of application development tools to easily interoperate deliver on the value and promise of rapid application development. Java projects deliver real business value in weeks and months, not years. Companies are able to leverage existing systems and data, enter new markets, and even develop new businesses in record times.

VisualAge for Java is the application development tool of choice when building Java-compatible applications, applets, and JavaBeans components.

VisualAge for Java is the newest member of IBM's VisualAge family. VisualAge for Java is a powerful suite of application development tools that allow you to build complete 100% Pure Java applications, applets, and JavaBeans by using the VisualAge "Construction from Parts" paradigm.

Combining the power of a true rapid application development (RAD) environment for Java with the ease of visual programming, VisualAge for Java connects the thin Java client to existing server applications, allowing you to extend your existing server applications to the Internet, intranet, or extranet, and to connect an existing Java solution to a server for access to data, transactions, or applications.

VisualAge for Java simplifies your Java development process in four major ways:

1. Client/server programming in Java is made easier through automatic generation of JavaBean components and middleware code that connects the Java client to existing transaction, data, and application servers.
2. An intelligent development environment enables your enterprise to build scalable Java solutions that run on Windows 95, Windows NT, OS/2, AIX, OS/400, and OS/390.
3. An automatic version-control facility allows you to easily go back to a working version of your code. This will be enhanced with a fully integrated

repository-based team environment that encourages simple and easy collaboration and management of development projects.

4. An advanced project-based-development environment enables programmers to create Java applications/applets or JavaBean components using construction from parts.

VisualAge for Java has the following key components:

- An integrated development environment enables VisualAge for Java to deliver rapid application development to your Java program during the development phase.
- A visual composition editor allows you to assemble applets, applications and beans from pre-selected parts on the visual builder palette.
- An enterprise access builder for transactions generates JavaBeans components that establish fast connections between the Java client and CICS Transaction Servers.
- An enterprise access builder for data allows database manipulation through Java language, using the JDBC standard.
- An enterprise access builder for applications allows you to build applications that connect Java clients to existing or new applications on the server, using Java client to Java server remote method invocation (RMI), Java client to C++ server through Java bindings for C++ classes and RMI.
- Real-time changes are reflected in currently run applications, letting you add a class, add a method, or change a method while still in the test phase.
- Integrated change management and version control provide the ability to store code, and to import and export files from the repository to classic file systems, using a single-user development repository.

4.2 Web Server

The Domino Go Webserver is an industry standards based Web server that allows you to build Web applications with connectivity to the databases you use daily in your business. Domino Go Webserver is a complete Web server product with advanced security and development features. It gives you, your customers, and your suppliers a secure, quick and easy way to complete business transactions electronically. It allows you to host and manage applications that:

- Distribute a wealth of information to your customers, partners, and prospects, using text, graphics, audio, and video.
- Publish presales or postsales product information.
- Create information that your audience can interact with using electronic forms or by e-mail.
- Dynamically track how your customers, suppliers, and personnel use the information you publish.

Domino Go Webserver is easy to install, use, and manage. The server provides consistent application programming interfaces (APIs), administration, and configuration across the spectrum of server platforms, desktop systems, departmental servers, all the way up to IBM System/390 mainframe systems.

Appendix A, “Domino Go Web Server for OS/390 Operations” on page 95, describes the use of the Domino Go Webserver for OS/390 to distribute the applet code to Web browsers

4.2.1 Features

Domino Go Webserver supports SSL Version 3 including client authentication. The SSL V3 implementation also includes server authentication, data encryption, and additional message digest hashing algorithms (see “Secure Sockets Layer” on page 36 for more information). If your environment has a SOCKS-based firewall for access to the Internet, Domino Go Webserver can be used in a proxy server role to access destinations outside the firewall. Client connections that use SSL are tunneled through the proxy server, eliminating the need to decrypt and reencrypt the data at the proxy.

Domino Go Webserver includes an SNMP subagent, which maintains server information and performance data in an SNMP management information base (MIB). From any SNMP-capable network manager (such as IBM NetView for AIX, TME10 Distributed Monitoring, or HP Open View) you can display, monitor, and adjust thresholds for your server performance.

Support for HTTP now includes full HTTP 1.1 compliance, which provides support for persistent connections and virtual hosts. Persistent connections allow the server to accept multiple requests and to send responses over the same TCP/IP connection. This eliminates a great deal of connection processing, resulting in faster responses to the clients and reduced network bandwidth requirements. Virtual hosts allow the use of one IP address to serve multiple files instead of requiring different IP addresses for different files.

Domino Go Webserver supports the expanded CGI which includes the Java programming language in addition to the other languages supported, such as C, REXX, and Perl. It provides a Java development environment based upon JDK1.1 standards for JavaBeans, JDBC for database access, and Java servlets. Java servlets, a Java server-side application, offer very significant performance enhancements over CGI applications and are truly portable.

Users can write applications that extend or customize how the Web server hands off client requests using the Go Webserver API. You can use existing HTTP methods or tailor them to your needs. This API support is language neutral, and user extensions run as part of the Web server process, with significantly better performance than CGI applications.

Domino Go Webserver also offers:

- A Web site content rating support
- A server activity monitor that allows you, through a browser, to monitor performance and status information about the server and the network
- A Java GUI that you can use to access your server activity information
- A search engine that delivers advanced search and navigation capabilities
- A Web usage mining tool that helps you organize your Web site more efficiently, determine the relative value of pages, and target marketing-based, on-page groupings.

4.2.2 Security Considerations

Domino Go Webserver supports the HTTP feature of basic authentication along with SSL encryption and SSL client authentication. By default, the pages are protected using TSO user IDs and passwords with basic authentication. The *Webmaster's Guide* describes how to configure the server for these types of security.

4.2.2.1 Basic Authentication

Basic authentication provides a means of protecting HTML pages on the server from unauthorized access. You define server rules for access control and, when a controlled page is requested by a browser, the server asks the browser for a user ID and password. This makes the browser prompt its user for this information. Without any form of encryption this technique is open to abuse because user IDs and passwords are transmitted unencoded.

When authentication is required, you can define users and passwords in files, set up access control lists, or use the system security subsystem running on your OS/390 system, such as RACF. You can also write custom code that is invoked when authentication is done, allowing site-specific operations to be performed. You might write a function to invoke a CICS program for the verification, or lookup users in a DB2 database for example.

4.2.2.2 Secure Sockets Layer

SSL is a security protocol that was developed by Netscape Communications Corporation, along with RSA Data Security, Inc. This protocol ensures that data transferred between a client and a server remains private. It allows the client to authenticate the identity of the server. In addition, SSL V3 allows the server to authenticate the identity of the client. Web browsers use https requests, to specify secure connections. https is an URL access method for connecting to HTTP servers using SSL. It uses digital certificates and requires specific configuration of the server. HTML pages, user IDs and passwords, and Java code can all be downloaded with the knowledge that connection is secure. An important point to note is that when a Java applet starts to execute, if it uses its own protocols to communicate with other servers; these are not secured by the https encryption. With the CICS Gateway for Java and JDBC connections, all the network communications are unsecured.

4.2.2.3 Signing Java Applet Code

Java 1.1 allows for applets running in browsers to have controlled access to their environment by the use of object signing. With the correct authority, granted by the user, applets can do the same things as Java applications. This means that developers can use the dynamic downloading advantages of applets and still write full specification programs. The applet code must be signed with a digital certificate before it is deployed on the server. The browser can verify this certificate and the user can be assured of the authenticity of the code and accountability of the developer. Appendix C, "Creating Signed Java Applets" on page 99 describes how applet code can be signed, and describes security principles in more detail.

4.3 CICS

CICS is IBM's general-purpose OLTP software. It is a powerful application server that runs on a range of operating systems, from the smallest desktop to the largest mainframe.

It is flexible enough to meet your transaction-processing needs, whether you have thousands of terminals or a client/server environment with workstations and LANs exploiting modern technology such as graphical interfaces or multimedia.

It takes care of the security and integrity of your data while looking after resource scheduling, thus making the most effective use of your resources. CICS seamlessly integrates all the basic software services required by OLTP applications, and provides a business application server to meet your information-processing needs, today and in the future.

CICS processes transactions, that can consist of many computing and data-access tasks to be executed in one or more machines. The tasks may include handling the user interface, data retrieval and modification, and communication. In CICS terms, these operations are grouped together as a unit of work or a transaction.

A transaction management system (sometimes called a *transaction monitor*) such as CICS:

- Handles the start, running, and completion of units of work for many concurrent users
- Enables the application (when started by an end-user) to run efficiently, to access a number of protected resources in a database or file system, and then to terminate, normally returning an output screen to the user
- Isolates many concurrent users from each other so that two users cannot update the same resource at the same time.

4.3.1 CICS Gateway for Java

The IBM CICS Gateway for Java brings easy state-of-the-art, open, access to CICS from any Java-enabled Web client, such as Netscape Navigator or a network computer.

A Java applet in the Web client can directly call CICS programs and data simply by invoking the small Java class supplied with the Gateway. When the applet is invoked, all the necessary code is downloaded to the client platform automatically, so no work is needed to prepare Web clients for CICS access.

The CICS Gateway for Java provides a way to access the transaction capabilities of CICS servers from the Internet. It combines the portable, architecture-neutral, object-oriented strengths of the Java programming environment with the power of CICS to bring access from the Internet to CICS applications.

Figure 6 shows the CICS Gateway for Java installed on a workstation. It accesses the CICS transactions using the external call interface (ECI) or the external presentation interface (EPI) of the CICS client.

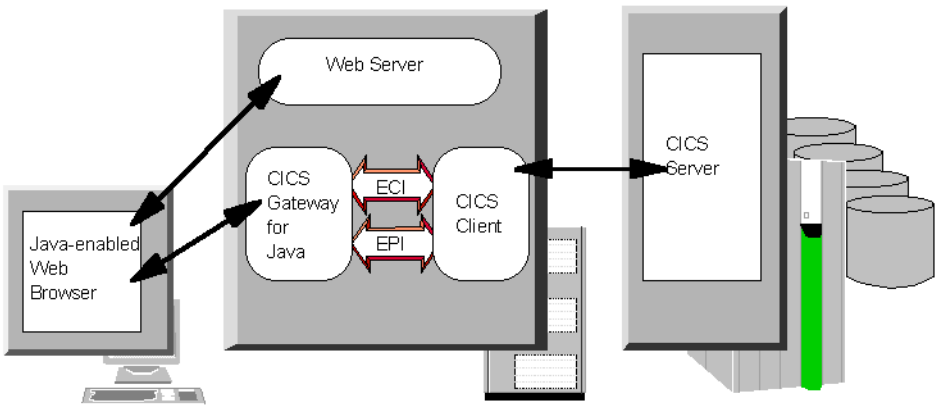


Figure 6. CICS Gateway for Java on a Workstation, with CICS Client

As showed in Figure 7, the CICS Gateway for Java execute on OS/390 providing Java applets access to CICS Transaction Server 1.2 in a two tier configuration.

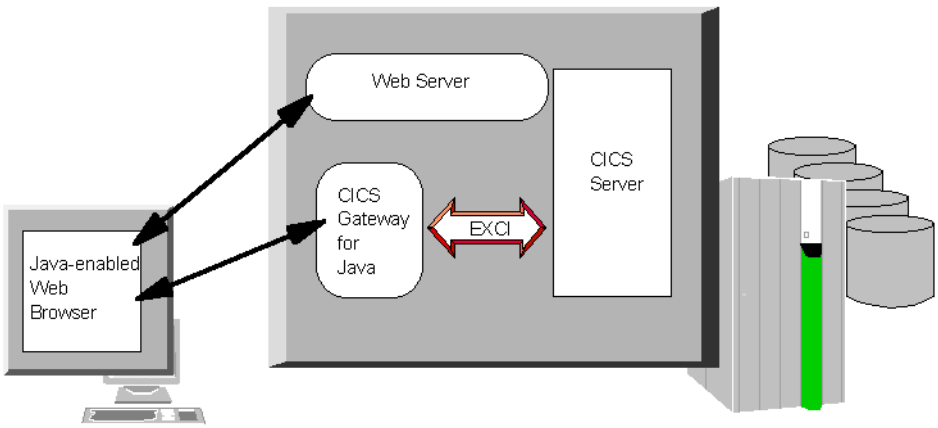


Figure 7. CICS Gateway for Java on OS/390

4.3.1.1 Operation

The CICS Gateway for Java is a TCP/IP server application that runs separately from CICS. Java applets make TCP/IP connections to the CICS Gateway for Java that uses ECI or EPI requests to connect to the CICS server. These programming interfaces, provided by the CICS Client and CICS workstation products, allow programs external to CICS to invoke CICS services.

When the CICS Gateway for Java is running on OS/390, ECI and EPI are not available. On OS/390, the CICS Client is not available, and CICS Transaction Server for OS/390 does not provide these interfaces to OS/390 programs. When running on OS/390, the CICS Gateway for Java actually uses the external CICS interface (EXCI) provided by CICS TS. The applets that are clients to the CICS Gateway for Java on OS/390 still use ECI request data, that are transformed to EXCI calls by the CICS Gateway for Java Thus the applet code can remain

unchanged, regardless of where the CICS Gateway for Java resides and which CICS server is used.

Figure 6 on page 38 shows how the CICS Gateway for Java can run on a workstation and use the CICS Client to access CICS servers in a three-tier configuration. The CICS Client can access multiple servers simultaneously, using different protocols

Figure 8 on page 39 shows the CICS Gateway for Java running on the same workstation as the CICS server, in a two-tier configuration, without the need for a CICS Client. The CICS Gateway for Java can access only the single local server, and no other CICS servers.

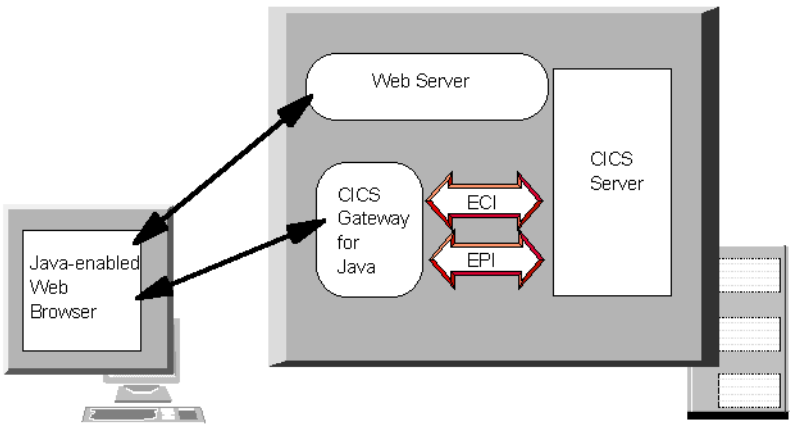


Figure 8. CICS Gateway for Java on a Workstation, with CICS Server

Figure 7 on page 38 shows that the use of the EXCI allows the CICS Gateway for Java to access multiple CICS TS regions, but not other CICS workstation servers.

4.3.1.2 CICS Gateway for Java on OS/390

For our environment, we used the gateway in its OS/390 version. Appendix B, “CICS Gateway for Java — Installation and Setup” on page 97 explains how we installed the gateway.

Figure 9 shows an overview of the CICS Gateway for Java executing on OS/390. The steps show the sequence of events for an applet that is downloaded to the Web browser and then issues a synchronous ECI request.

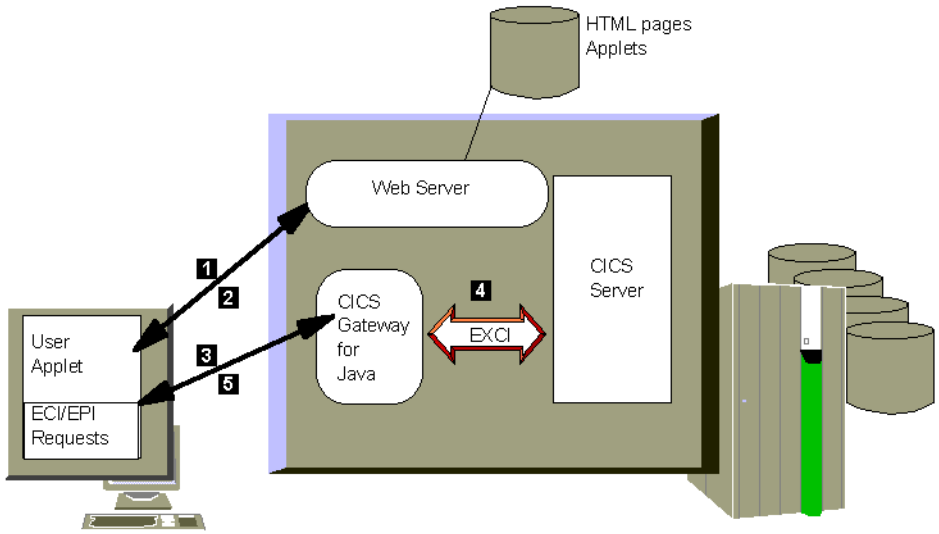


Figure 9. CICS Gateway for Java on OS/390: Sequence of Events

The sequence of events is the following:

1. The browser requests an HTML page from the Web server, and the page contains an applet tag.
2. The browser requests the Java class files and starts executing the applet.
3. The applet creates a JavaGateway object, which opens the TCP/IP connection to the CICS Gateway for Java. The applet also creates an ECIRequest object, which contains the information for an ECI request, including a CICS program name and a COMMAREA. The JavaGateway flow method is invoked by using the ECIRequest object as the parameter. This sends the request information to the CICS Gateway for Java.
4. The CICS Gateway for Java receives the request information and issues an EXCI call to the CICS TS server.
5. The CICS Gateway for Java returns the results to the applet. The ECIRequest object in the applet now contains the COMMAREA as updated by the CICS program.

4.3.1.3 Programming Interface

The CICS Gateway for Java provides several classes and interfaces that you can use to access CICS programs from Java applets or applications. These include:

- ibm.cics.jgate.client.JavaGateway
- ibm.cics.jgate.client.ECIRequest
- ibm.cics.jgate.client.EPIRequest
- ibm.cics.jgate.client.Callbackable
- ibm.cics.jgate.client.GatewayRequest

When an instance of the JavaGateway class is created, the constructor method requires the TCP/IP address and port number of the CICS Gateway for Java to which a connection will be made. The object opens a TCP/IP socket connection to the specified CICS Gateway for Java, and this connection remains open until

the close method is called or the object is destroyed. The GatewayRequest class is a superclass of the ECIRequest and EPIRequest and contains variables common to both. The Java program creates instances of ECIRequest and EPIRequest, and these objects are used as parameters to the flow method of the JavaGateway object. With the latest version of the CICS Gateway for Java, a new feature, called *Local Java Gateway,* enables the Java Gateway to communicate directly to the local CICS client without the need for a network communication between the Java Gateway and the CICS client.

The Callbackable class is a Java interface that is used when you require an asynchronous call with callback. You need to write a class that implements the Callbackable interface and pass an instance of this class to the setCallback method of the request object. When flow is called, it returns immediately, and when the data is actually returned by the CICS Gateway for Java, the Callbackable object is run in a new thread.

4.4 DB2

DB2, in its versions for the MVS and Workstation platforms, provides data storage facilities for our application. DB2 offers a comprehensive set of functions, which has been extended over the years, for implementing distributed applications. New facilities, not available in 1992, are now in place, which broadens the number of possible technical solutions. For example, it is now possible to establish a DRDA connection from DB2 for OS/390 to DB2 for OS/2, because DB2 for OS/2 now implements the DRDA Application Server function. It is possible for an MVS COBOL program to access DB2 for OS/2 tables, without any need to use another COBOL program on OS/2 invoked through CICS intersystem facilities.

IBM DataJoiner is another recent product which many businesses are implementing successfully. It provides total location transparency to the applications, which can access the total corporate data assets as if they constituted a single database. Note that, with proper adds-on to the product, this is true also for nonrelational data stores, for which the same SQL interface used for relational databases (IBM and other) is also available.

In our project, we did not exploit any of these new products or features. Time constraints, coupled with the willingness to show other new solutions such as the CICS Gateway for Java and VisualAge for Java brought us to the decision to keep the new application as close as possible to the original, as far as data access goes. Nevertheless, in approaching a real-life application migration, you would probably consider exploiting some of these new technologies.

DB2 is fully integrated with Web technology so that data can be easily accessed from the Internet or from your intranet with complete security. The following facilities included with DB2 allow you to Web-enable your database applications right out of the box:

- Net.Data - A Webserver Database Gateway
- DB2 Java Support

We used those new Web integration technologies to support our application migration.

4.4.1 Net.Data

Net.Data is a Web-enabling tool for interactive database-to-Web applications. Net.Data is IBM's strategic product for enabling Internet/intranet access to relational data on a variety of platforms. Figure 10 shows the Net.Data architecture.

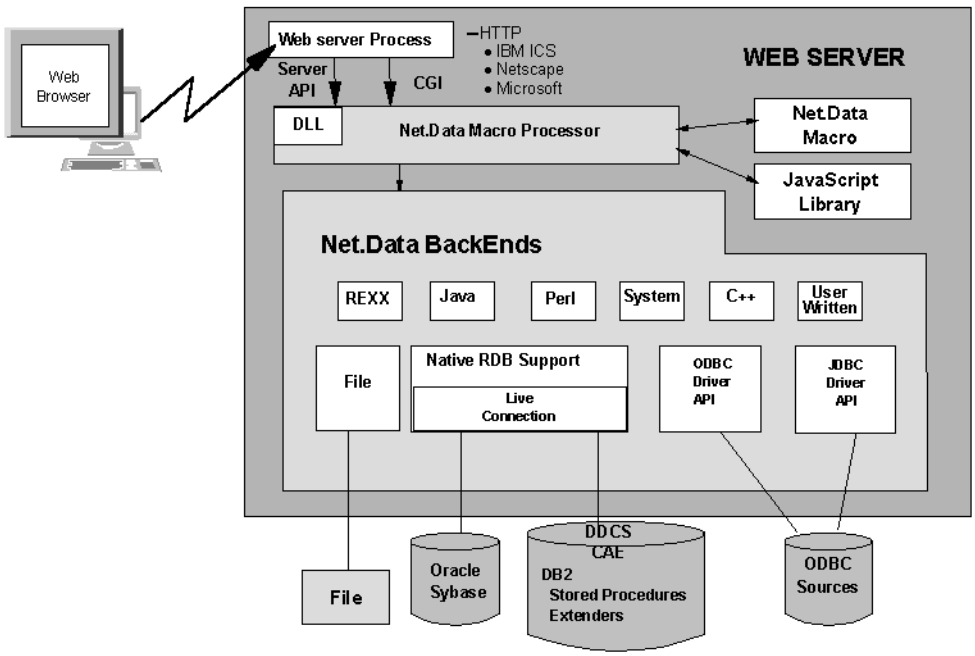


Figure 10. Net.Data Architecture

Net.Data provides open data access to DB2 and other data sources including Oracle, Sybase, and any ODBC data server. With Net.Data, your application can use DB2 to build dynamic Web pages to support electronic commerce applications. Net.Data runs on OS/2, AIX, HP-UX, Windows NT, Solaris, and SCO-UNIX.

Net.Data provides high-performance, robust, application development function and allows exploitation of existing business logic through open programming interfaces. Net.Data tightly integrates with Web-server APIs such as those from IBM, Lotus, Netscape, and Microsoft, providing higher performance than CGI applications.

Net.Data provides connection management to your key relational databases for optimum performance. Net.Data can establish a continuous connection to specified databases. Net.Data maintains the connection throughout the Web application and across invocations of Web applications. Since the database connection is continuous, the application does not experience the overhead of repeated connects to the database. The result is peak performance of your interactive Web application. The Net.Data application is a macro with a rich macro language, variable substitution, conditional logic, and optional function calls. Net.Data supports client-side processing with Java applets and JavaScript. Serve- side processing includes Java applications and REXX, Perl, and C/C++ applications.

4.4.2 DB2 Java Support

Java applications are very attractive to customers who would like to develop a single application running on any operating system and who would like to reduce the cost of application distribution and maintenance.

JDBC is a database access interface for Java applications that is being delivered with DB2. The DB2 JDBC database interface supports this API.

DB2 provides native support for Java at client workstations and DB2 servers. Java is supported on the client workstations in two ways:

- A Java application uses the DB2 Client Application Enabler (CAE), which must be installed on the client workstation, to communicate with the DB2 server. Customers with existing DB2 client/server configurations can now use Java as a database application development tool.
- Java applets allow applications to be developed that access DB2 servers without requiring DB2 CAE code to be installed on client workstations. Java applets can be automatically downloaded to the client workstation at application invocation time. Java support in DB2 servers consists of the ability to create native Java-based, user-defined functions and stored procedures. Figure 11 shows a sample of a JDBC applet.

```
import java.sql.*;
public class appletSample extends java.applet.Applet
{
   { Class.forName("COM.ibm.db2.jdbc.net.DB2Driver"); } // load applet
driver
   String results = "";
   public void start()
   {
      Connection con = DriverManager.getConnection(
                       "jdbc:db2://malmo:8888","userid","password");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT branch_name FROM branch");
      while(rs.next()) // fetch a row until EOF
      {
        String name = rs.getString(1); // fetch column value
        results = results + ", " + branch_name; // concatenate names
      }
      rs.close(); stmt.close(); con.close(); // close up
   }
   public void paint(Graphics g)
   {
      g.drawString("Branches", 0, 10);
      g.drawString(results, 0, 20);
   }
}
```

Figure 11. JDBC Applet Sample

4.4.3 Configuration

Figure 12 shows the configuration difference between Net.Data and JDBC.

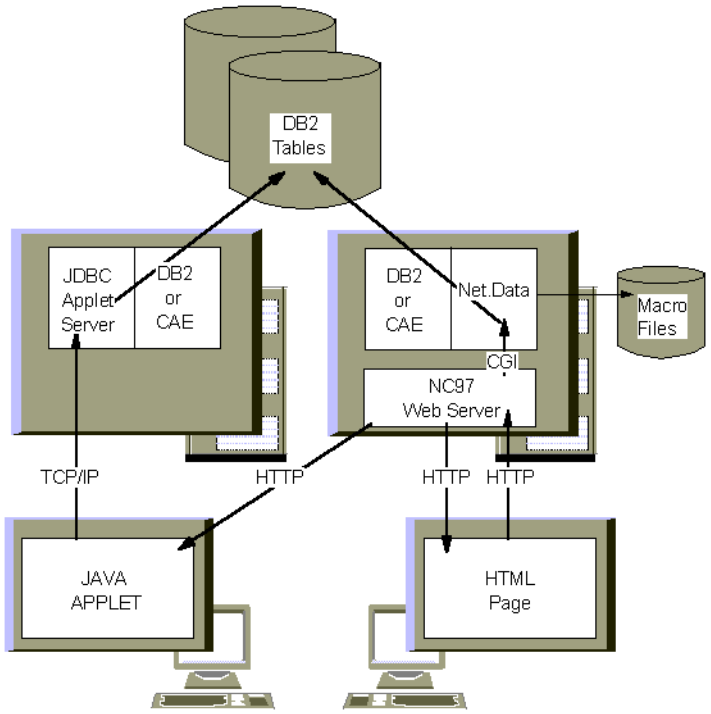


Figure 12. Net.Data and JDBC Configuration

The two Web browser applications, the HTML page using a Net.Data connection, or the Java applet using a JDBC connection, are downloaded from the Web server using HTTP protocol:

- Net.Data

  Using HTTP, the HTML page sends back to the originating Web server some information needed to access the database. Using CGI, the Web server then passes the argument to the Net.Data program. Using the DB2 CAE, Net.Data connects to the database and accesses the tables.

- JDBC

  Using TCP/IP, the Java applet uses the JDBC applet driver to open a connection with the JDBC applet server. The JDBC applet server then, using the DB2 CAE, connects to the database and accesses the tables. It is also important to notice that the JDBC applet server can be located on another machine as the Web server, if the Java applet has been developed using JDK 1.1.

Chapter 5. Network Computing Security Environment

When we migrate CS92 to NC97 using this new network computing model, one of our biggest challenges was to remove the security exposures that the Internet brings. Handling security changes fundamentally between the client/server model and the network computing model.

This chapter describes the security issues you need to be aware of when developing a Java based solution. Java security features are currently in a state of change, and vary depending upon browsers and Java virtual machine implementations. The information here should enable you to write a Java program that will not be broken by security restrictions when you deploy it.

5.1 Java Security Features

In general, Java applets that are loaded from a network are deemed untrusted and are restricted in the actions they can perform. An untrusted applet cannot:

- Make network connections to hosts other than the originating host.
- Read and write files on local disks.
- Start external programs and processes.
- Load libraries and define native methods.

These restrictions allow a user to download an applet with reasonable assurance that it cannot harm their system. It is up to the implementation of the Java virtual machine (JVM) in the browser to ensure that these restrictions are policed. Security bugs in JVM implementations have been found and Javasoft maintain a chronology page on their Web site to record the status of these bugs. Since its introduction, the Java technology has evolved in the public eye and when security bugs are found, fixes have quickly followed.

Web browsers that support Java 1.0 applets are very strict about enforcing the security rules and the term *sandbox* has been used to describe the restrictive environment that the applet runs in. This is not a problem for small applets that enhance the look and interface of Web pages, and do not provide much complex functionality.

Applications written in Java do not have these same restrictions and can access the system in the same ways as traditional compiled programs. This is because applications must be installed locally on a computer and run from the local disk. It is the responsibility of the person installing the application to know what it is and that it can be trusted. This approach loses the advantages of dynamically downloadable applets, where no code needs to be installed on user machines.

Java 1.1 addresses this issue by allowing applet class files to be digitally signed by the developer. This allows the end user to determine the degree of trust that the applet can be afforded. When the trust is established, the applet can be allowed to perform actions that an untrusted applet could not do. For example, in NC97, the client applet needs to make network connections to servers other than the originating host.

5.2 Leaving the Sandbox

The latest versions of the popular web browsers, Microsoft Internet Explorer version 4, Netscape Communicator Version 4 and HotJava Version 1.1 provide mechanisms for the user to allow Java applets access beyond the sandbox. These are based on digital signature and certificate technology and on the user's interaction with the browser. When a Java applet has been digitally signed, the user can allocate security settings based on the signature. The applet can then do things that would otherwise not be allowed if the applet was without a signature. 5.6, "Digital Certificates" on page 51 details how using digital certificate technology allows the user to make these security choices safely.

Once a mechanism for trusting a Java applet is used, the developer is free to write code for any purpose. You have the full power of the Java language and can write sophisticated client programs without any restrictions. These then use the dynamic downloading capabilities of browsers, removing the need to administer client platforms and install software manually.

Within an intranet environment, the applet trust should be implicit, since the code has been developed by the company and the intranet is secure. The use of external trusted third-party Certification Authorities as described in 5.6, "Digital Certificates" on page 51 would not be necessary.

The only drawback at the current time is the mechanisms used by different browsers to establish whether or not an applet is trusted. Each of the popular browsers uses a different mechanism, and Netscape Communicator requires the use of special API calls in the Java applet code to request the privileges to perform insecure actions. The certificate technology required to sign the code is also different for each browser. Applet code that has been signed for one browser will not work with the others. The developers are required to provide a separate copy of the signed code for each browser.

Sun Microsystems have evolved the sandbox model in their architecture for Java 1.2 to include fine-grained access control. The details are in the Java Security Architecture (JDK 1.2) document available from Sun Microsystems. The new APIs have a purpose similar to that of the Netscape Communicator Capabilities API described in 5.3, "Netscape Capabilities API" on page 46. When Netscape Communicator supports Java 1.2 developers should no longer need to use the Capabilities API. Other browsers that claim support for Java 1.2 will also need to provide the security APIs that will be part of the core Java API. This will solve the problem of requiring specific code for particular browsers. Java 1.2 should also provide a standard for signing Java archive (JAR) archive files that will be common to Java browsers. This will allow for a single JAR files to be signed and then used in any browser.

5.3 Netscape Capabilities API

A signed Java applet that runs in Netscape Communicator is not automatically allowed to access system resources. The developer must use calls to the Capabilities API to enable privileges before the actions can be taken.

The PrivilegeManager class controls the access to a set of system targets. For example, in order to successfully make a network connection the following code must be used:

```
PrivilegeManager.enablePrivilege("UniversalConnect")
```

The complete set of targets covers all the things you may want an applet to do, and groups of targets allow for more general access. For example, the TerminalEmulator target is required by a terminal emulator such as 3270; it allows you to read and write files and establish network connections. This target includes the targets UniversalLinkAccess, UniversalPropertyRead, UniversalListen, UniversalAccept, and UniversalConnect. The Netscape developer Web site contains details on using the API, including a complete list of all the targets.

Netscape has implemented an extension to the core Java API to provide support for capability-defined security. To perform any kind of function that involves interacting with other resources, such as establishing network connections, a call to the method enablePrivilege in the Netscape specific class PrivilegeManager is required. We have implemented this code in the TransactionManager and DataManager. If this code is executed in a browser other than Netscape Communicator, a run-time error will be thrown. To safeguard against this, extra code is required to determine whether the applet is executing in a Netscape Communicator browser. Various solutions have been published on the Internet. The solution we adopted involves implementing a SecurityContext class that has two static methods: isCommunicator and isCapableOf. isCapableOf is a public method that returns a boolean to indicate whether a privilege can be granted. isCommunicator is a private method that attempts to load a Netscape security class. A boolean value is returned that determines whether the load was successful

5.3.1 Implementation

When an applet make an enablePrivilege call for the first time, the user must decide how to process the request. Netscape Communicator displays a panel like that of Figure 13.

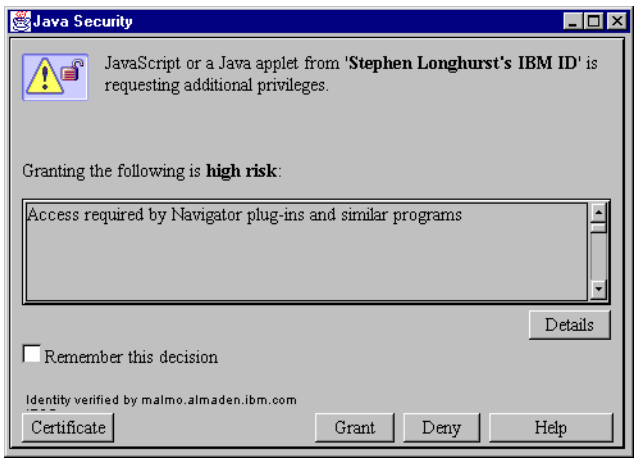


Figure 13. Netscape Security Warning

The name of the certificate is displayed, "Stephen Longhurst's IBM ID" in Figure 13, and a short description of the target being requested. The full certificate can be displayed by clicking on the Certificate button. If the certificate is not issued by a recognized authority, then the applet is automatically denied the requested privilege with no user intervention.

The user can grant or deny the privilege, and set the checkbox so that the panel is not displayed every time the applet enables the privilege. You can check and alter the privileges afforded to a certificate by using the using the "Security Info" option any time.

The advantage of this method of setting the privileges is that the user does not need to know in advance what an applet needs to do to run properly. During the execution, the messages are displayed as required. We would recommend issuing enablePrivilege calls for each target your applet requires access to when execution first starts. This gets the user's dialogs out of the way at the beginning, rather than interrupting the user during normal operation.

5.3.2 Principles

The principles in the Capabilities API refer to who is allowed to do certain actions, and the targets refer to what they want to do. The principle of an applet is normally associated with its signature; hence, only signed applets can make use of the Capabilities API. It is the name of the signature that is listed in the Java/JavaScript section of the "Security Info" panel.

For development purposes, having to sign the applet continuously for testing is not very productive. You can get Netscape Communicator to recognize the applet's code base as a principle when evaluating the privileges by adding the following line to the preferences file:

```
user_pref("signed.applets.codebase_principal_support", true);
```

This allows you to write code that uses the Capabilities API, but does not need to be signed.

5.4 Microsoft Internet Explorer Security Zone System

Microsoft Internet Explorer 4 introduces the concept of dividing the Web into different zones. You can assign different levels of security to each zone, trusting each at a different level. For example, you can set the intranet zone to low security (high trust) but the Internet zone to a high security level (low trust). Figure 14 shows the security settings panel in Microsoft Internet Explorer version 4. You can see that the Internet zone has a high security setting by default.

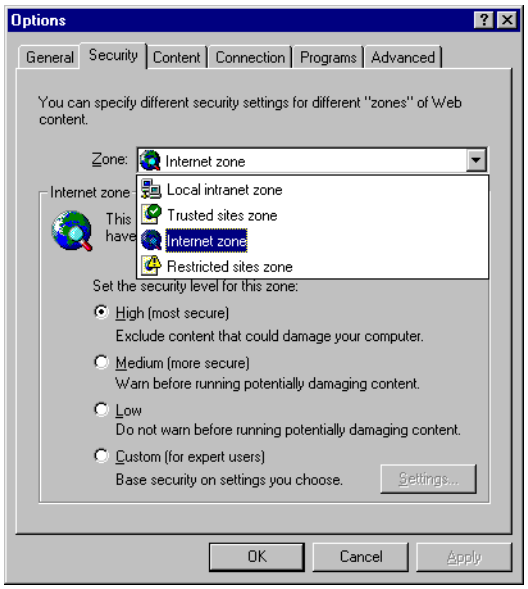


Figure 14. Internet Explore Security Setting Panel

Java applets are allowed levels of access and capabilities based on the zone from which they originate. By signing an applet, the developer can also specify the level at which the applet needs to run. If the applet is loaded from a higher security zone, the user needs to decide whether the applet will be allowed to do what it requests.

The capabilities that a Java applet can be granted are defined in high, medium and low settings. The high setting is equivalent to the sandbox-like environment. The medium setting adds the capability to access a scratch pad. The scratch pad is a secure area, proprietary to Microsoft Internet Explorer, that allows applets to store information on the client machine. It is accessed with Java APIs that are provided with the Microsoft Java Developers Kit. The low setting allows applets complete access to the system.

Microsoft claims this approach allows fine-grained control over what a Java applet is allowed to do. If you discount the medium setting because it is the same as the high setting with one extra capability, you have a either a sandbox or complete access to everything. As a user, if you allow a Java applet to run with low security, then it has complete access to your system. This is unlike the Netscape model that allows you to control exactly which resources an applet can access.

If Microsoft Internet Explorer allowed more control over what the high, medium and low settings mean, then the model would be simpler to use than Netscape's Capabilities API, from a development point of view. The Microsoft Java Security White Paper implies that this is the case, but we could not determine how to do it. It is probably done using the Internet Explorer Administrator's Kit.

5.5 The HotJava Security Model

The HotJava browser has quite a flexible approach to security from a user's point of view. A user can finely control the things that an applet can and cannot do,

based on the site it comes from or the certificate it is signed with. The security dialogs allow for defaults as well as customized settings, and you can group sites and certificates together for easy configuration. This greater flexibility makes it easier for somebody to unwittingly allow an untrusted applet to perform undesired operations. Applets that are not signed can be allowed privileges that the other browsers make available only to signed applets.

Figure 15 shows the basic security dialog in HotJava. You can set the options for both signed and unsigned applets.

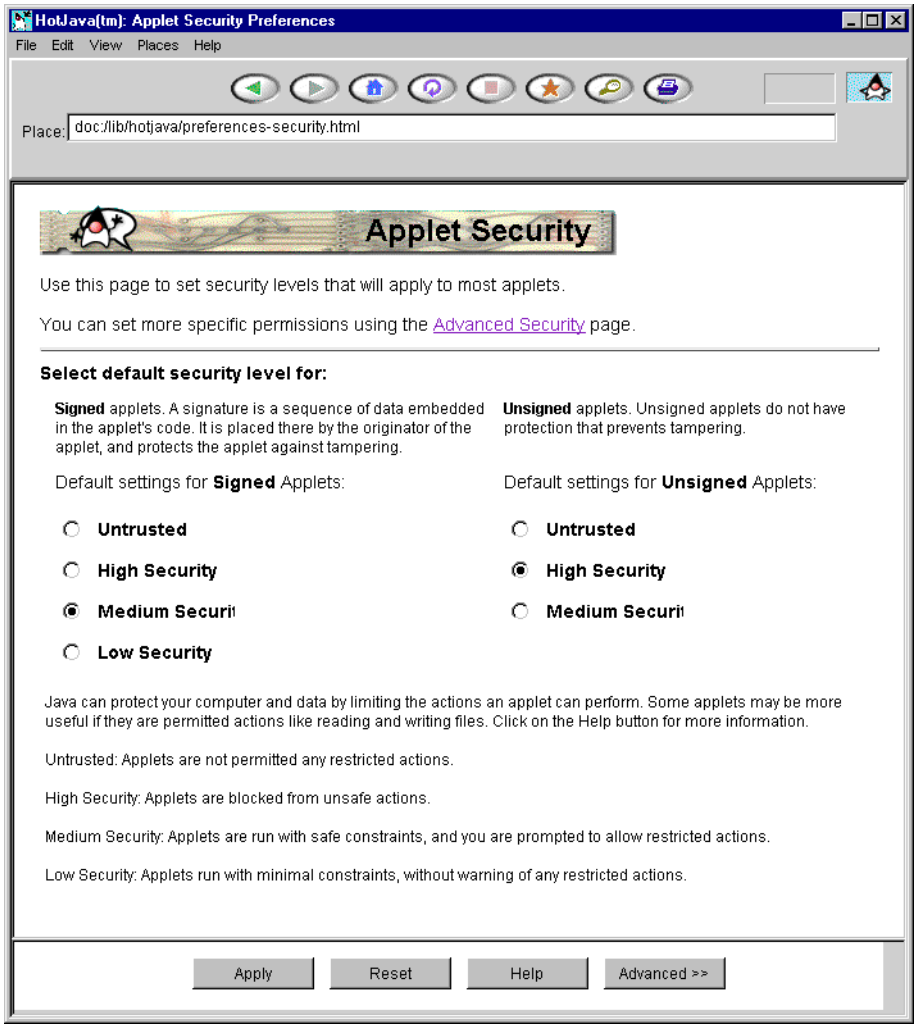


Figure 15. HotJava Basic Security Panel

The advanced options, shown in Figure 16 allow you to configure specific permissions granted to applets. This is very comprehensive and allows control over things like specific sites the applet can or cannot connect to, exact directories on your disk that the applet can or cannot write to, and the system properties the applet is allowed to access.

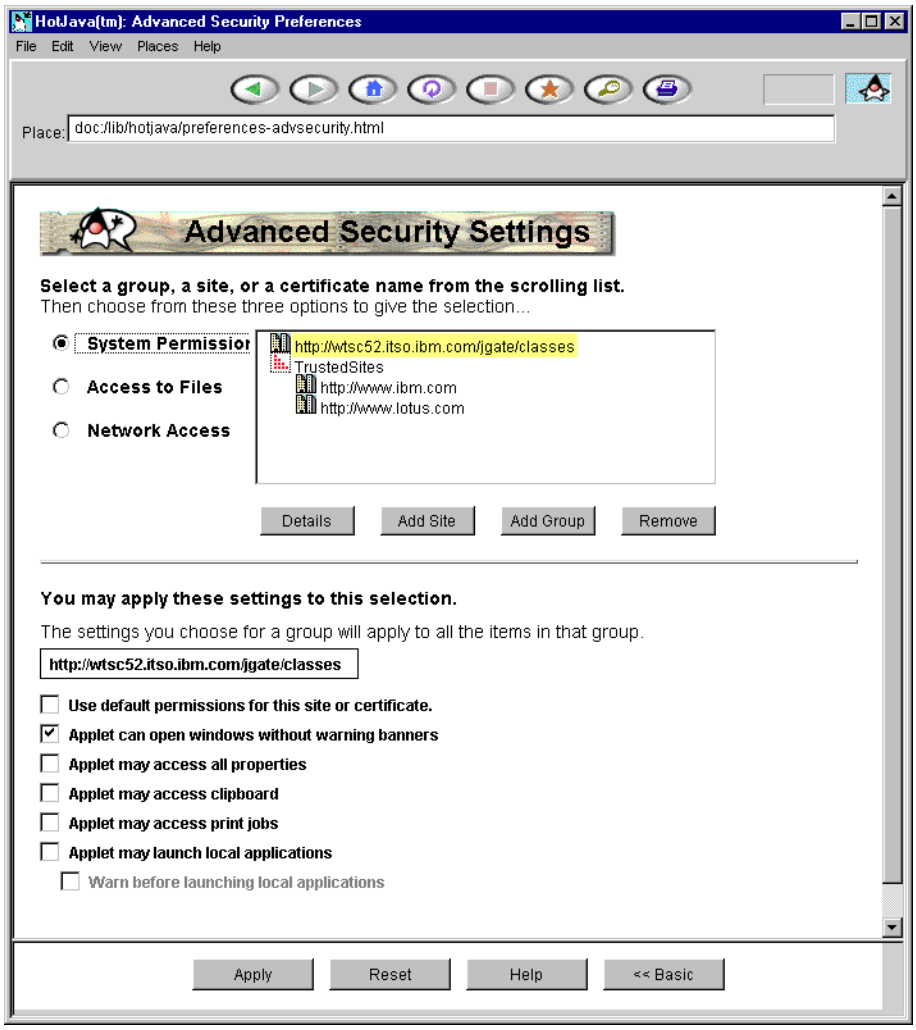


Figure 16. HotJava Advanced Security Preference Panel

HotJava is relatively new and is less sophisticated than the other browsers. One thing missing that is available with Microsoft Internet Explorer and Netscape Communicator is a centralized administration point. By using administration products from Microsoft or Netscape, end-users browsers can be configured by a single administrator. The configurations can then be locked so that users cannot alter them. This allows administrators to decide on a security policy that all browsers must follow, and prevent the end users from circumventing it. In a company intranet environment, enforcing a security policy may be very important.

5.6 Digital Certificates

A digital certificate is a data structure that contains three pieces of information: a name, a public key, and a digital signature computed over the other two. The certificate is signed by a trusted third party called a *certification authority* (CA). It is the job of the CA to verify that the name information is correct. If you trust the CA, then digital certificates provide a safe method for distributing public keys via an electronic medium.

A digital signature is a mechanism that associates data with the owner of a particular private key. The digital signature for a message is created by hashing the message to produce a message digest. This message digest is then encrypted with the private key of the individual sending the message, the key then becomes the digital signature. When the message is received, the recipient decrypts the digital signature with the public key of the sender. The message digest is then calculated from the received message and compared with the decrypted one. If the two match, then the message has not been tampered with. By using the public key of the sender to verify the message, we can be sure that the message was encoded by the private key known only to the sender.

Your copy of Netscape Communicator or Microsoft Internet Explorer comes loaded with the digital certificates of trusted CAs such as Verisign, and the IBM World Registry. Figure 17 shows the Netscape Communicator security info dialog, displaying a list of the CA certificates. If you create your own CA for intranet use then your browsers must be loaded with your CA certificate. This function is provided by the Certificate Server, and is the only manual configuration step that the browser user must take before running applets signed with your developer certificates.

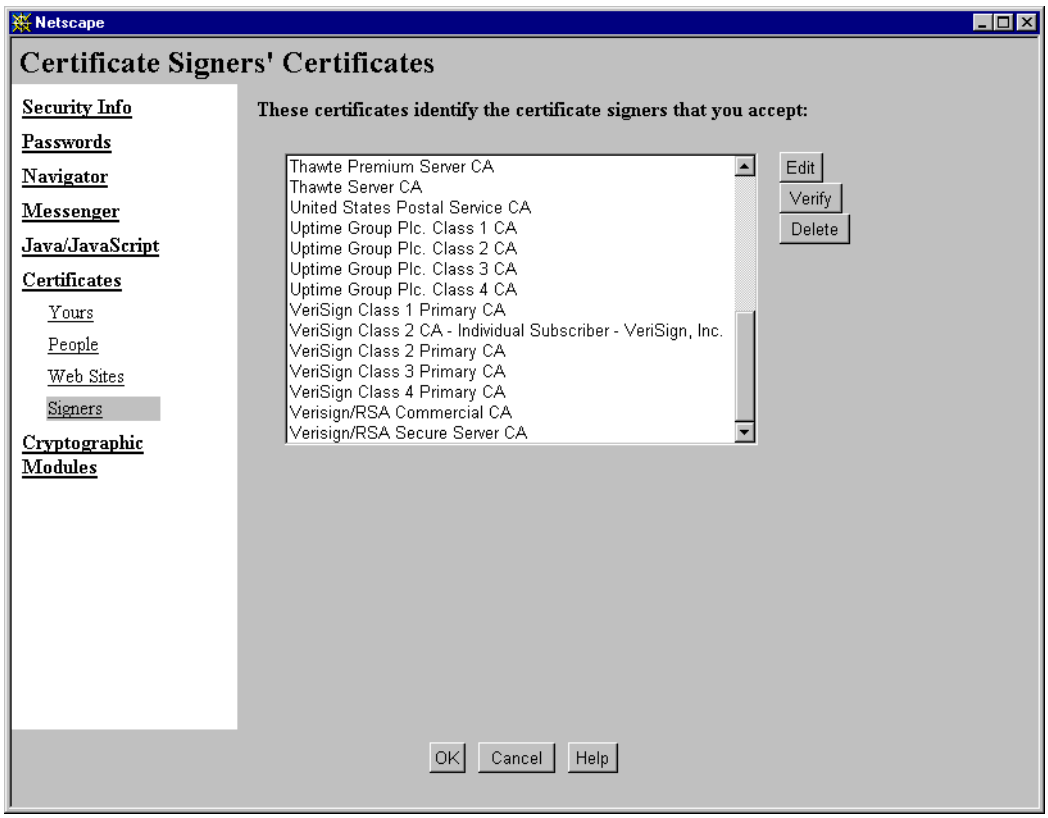


Figure 17. Netscape Security Info Window

Figure 18 on page 53 shows a Netscape Communicator browser importing a new CA certificate.

Figure 18. Importing a New Certificate with Netscape

5.6.1 Why Sign Java Applets?

In order for downloaded Java code to have access to a system beyond the sandbox, you must explicitly grant it privileges. In order to make the decision as to whether to grant the access or not, certain conditions must be met. Therefore, the downloaded Java applet must meet the following criteria:

- Authenticity

  You must be able to authenticate the person the applet has been developed by, and that you trust that person. With off-the-shelf software, the packaging usually identifies the software publishers very explicitly. Other physical identifiers like holograms and certificates of authenticity allow you to trust the program you install on your machine. When you download an applet into your Web browser, you do not get the same guarantees about where the software has come from, or who developed it.

  When Java code is digitally signed, you have reliable information about who developed the code. You can then decide what to allow the program to do. When the Web server employs secure sockets, you can be assured of where the code is being downloaded from.

- Integrity

You need to be sure of the integrity of the code that you download. If it has been altered in any way during transmission, you cannot trust it. Encryption and digital signatures ensure integrity.

- Accountability

  When you receive code that is signed by a particular developer or organization, you can be sure that they cannot deny it is their code. Only they should have knowledge of their secret key, which is used to create the signature. If something does go wrong, you know who is accountable for the mistakes.

Appendix C, “Creating Signed Java Applets” on page 99 explains the process required to build a signed archive file for the main Web browsers.

5.6.2 Obtaining a Digital Certificate

The *javakey* program and Microsoft code-signing technology allow you to create your own certificates for signing Java code. The Java security Web pages at Sun describe how to create certificates using *javakey* and “Microsoft Authenticode Technology” on page 101 shows how to use the *makecert* program to create a Microsoft code-signing certificate.

Obtaining a Netscape code-signing certificate is more difficult. You can purchase one from a commercial CA such as Verisign Inc. or Thawte Consulting. The average cost is about $20 per year. Verisign call the certificates enabled for code signing Class 2 or Class 3 certificates. Class 2 are for personal use while Class 3 are corporate certificates and cost significantly more. At the time of writing, Verisign issue these certificates to U.S. and Canadian residents only. Netscape maintains a Web page of companies that issue certificates to clients, but they are limited today to Belgium, Luxembourg, Brazil, Spain, and South America. Thawte Consulting offer services in more countries, but not for code-signing certificates in every country. The lack of services in countries other than the US is due to the difficulty in verifying individual identities. The U.S. Social Security Number provides a consistent method to uniquely identify a U.S. resident, while not all other countries have a similar mechanism. Verisign and the other companies also offer Microsoft code-signing certificates, if you need to publish software for the Internet.

If you do not want to pay money for a Verisign certificate, or you live in a country where certificates are not available, you can download the Netscape Certificate Server evaluation copy. This allows you to set yourself up as a certificate authority, within an Intranet environment, and issue yourself and others with developers’ certificates. These certificates, like the Microsoft developer test certificates, will not be trusted on the public Internet. Using the Certificate Server is a good way to learn about the technology and become familiar with the processes of obtaining and using digital certificates. It is quite simple to install and set up, and comes with comprehensive instructions. There is also a code patch available from Netscape that allows the Certificate Server to issue certificates to the Microsoft Internet Explorer browser. A detailed description of the Netscape Certificate Server is beyond the scope of this chapter, but it is worth looking at for evaluation.

5.7 Security Features in Java 1.2

The Java 1.1 platform does not provide concrete implementations of some key security features. Certain certificate formats and algorithms are to be introduced with Java 1.2. The certificate management infrastructure will include support for X509v3 certificates. A new permission based security mechanism is also proposed, similar to the Netscape Capabilities API. This will allow common code that requests access outside the sandbox to be written. It will be compatible with all browsers and Java environments. The problem today is the need for browser specific code, the Netscape Capabilities API, to be written. The support for X509v3 certificates will remove the need to sign applet code for specific browsers. More information on security and the proposed specification for Java 1.2 is available at Suns Web site.

5.8 Secure Java Applets

To write and install secure Java applets, you need to understand three different domains:

- Writing the applet

  You need to be aware of how Web browsers enforce security on Java applets if you want to write code that needs access to the system beyond that allowed by the sandbox model. Different browsers use different mechanisms to show whether code can be trusted:

  - For use with Netscape Communicator, your code must include calls to the Capabilities API. You create a JAR file that must be digitally signed with a code-signing certificate compatible with the Netscape tools.
  - For use with HotJava, you must inform the browser users of the requirements of your applet. Users then have to configure the browser to allow the applet the access it requires. Your code may or may not be digitally signed, but if it is you must create a JAR file and use the *javakey* program to sign it. This JAR file must be different from the one that is used with Netscape Communicator.
  - For use with Microsoft Internet Explorer, you must create a cabinet (CAB) file that is digitally signed and specify the level of access that your code requires. You need a code-signing certificate that is compatible with the Microsoft Authenticode tools.

  When using a digital certificate, the browser must recognize the Certificate Authority that signed your certificate. Either you need to obtain the certificate from a recognized authority, such as Verisign, or import your authority's certificate into the browser and mark it as trusted.

- Securing the applet communication

  The IBM CICS Gateway for Java Version 1 classes do not use encryption when transferring data from the applet to the server. This will not be secure in an Internet environment. You may need to think of other communication options if using CICS, or wait for security features to be included with the CICS Gateway for Java. You can also use third-party libraries to implement encryption of your COMMAREAs and database information independent of the CICS Gateway for Java or JDBC. JDBC connections also do not employ

encryption, and face the same problems as the CICS Gateway for Java communication.

- Web Server Security

  If you require authenticated access to your Web server, there are two options. You can employ basic authentication or use SSL client authentication. When the server is required to authenticate a request you can use custom written code to access existing security services, or design new policies using the server administration tools. For example, your server might invoke a CICS program that uses the CICS API command EXEC CICS VERIFY PASSWORD to authenticate a particular user.

Chapter 6. Designing a Network Computing Application

The application design tasks for CS92 included tasks specific to developing a client/server application. In this chapter, we look at the design tasks to see which of them relate to a network computing application. A task might be relevant simply because it is part of good application design, or because it relates generally to distributed computing. Alternatively, there may no longer be any need to perform a task because it applies only to client/server applications. In addition, we look at new tasks that network computing makes necessary.

6.1 From CS92 to NC97: Application Selection

Any application that has distributed processing requirements is a good candidate for development using the network computing model. An existing application with a robust mainframe element but which needs to reach a wider audience of users is an ideal candidate for migration to network computing.

It would have been possible to develop an entirely new application that used the network computing model, however, we wanted to show that an existing client/server application could be migrated easily to a network computing environment for three main reasons:

- CS92 had been developed to demonstrate the principles of the client/server model. It was using the principal technologies of this model, and therefore it was a good case study to show how they migrate to the new model.
- The application was developed five years ago: its client portion was coded in EASEL and its server portion in COBOL. The server code still existed, but as there was no longer support or a development environment for EASEL, the GUI part of the application had to be redeveloped. It appears to us that this situation may often occur in a real situation, when choices made in the past have to be reconsidered because of an important change in technologies.
- The last reason derives from the first two. In applying the principles of the client/server model, the CS92 developers isolate each application part, the presentation, the business logic, and the data access. Therefore, it was easy for us to concentrate on the redevelopment of the GUI only, limiting the scope of our project in term of resource and time. It also gave us the opportunity to use Java and exploit the potential of network computing.

The migrated application, that we called NC97 for Network Computing in 1997, was on its tracks

6.2 NC97 Infrastructure

The task of defining the configuration of the system is still necessary. In a migration project, this can simply reconfirm that the existing configuration satisfies the requirements or, more radically, the configuration could be redesigned to remove one tier of the architecture. The decision would be specific to each project, depending upon the resources available and any other uses that the configuration is being used for.

Figure 19 shows the new infrastructure that was chosen to migrate the application.

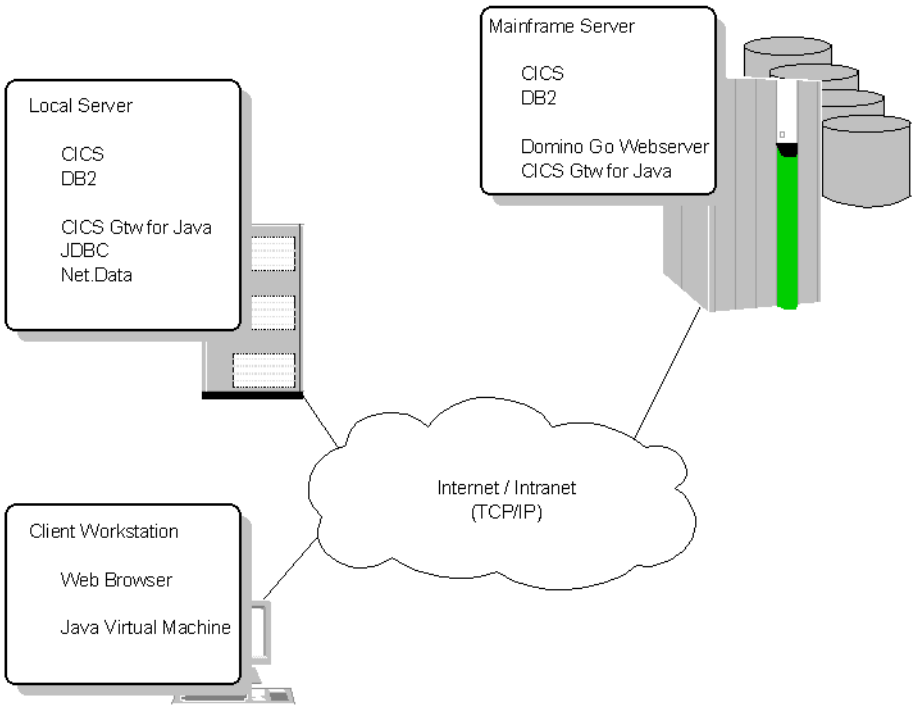


Figure 19. NC97 Infrastructure

On one hand, we tried to keep as much as possible of the existing programs on the local server and on the mainframe server. On the other hand, we tried to implement the new technologies offered by the NCF to the extent possible.

Therefore, we developed a new front end to the application using Java and Java applets. Our development tool was the newest member of IBM's VisualAge family: VisualAge for Java.

To connect to the existing DB2 data and CICS transactions, we used two NCF connectors:

- JDBC to access the DB2 data
- CICS Gateway for Java

We added a new function to the application using Net.Data.

A Web browser was the new application interface to the user. HTML pages and our Java applets were stored in a Web server, the Domino Go Webserver for OS/390.

6.2.1 Hardware

When looking at the hardware configuration, we faced two alternatives:

- A two-tier configuration, where the client using a Web browser connects to a single host running the Web server and the subsystems (DB2 and CICS).

  This configuration has advantages for administration, maintenance, and system management because of the unique server in the configuration. A unique team can manage that machine. There is only one version of the

software, system, or applications. The users have only one entry point to the systems wherever they are located, on the intranet or the Internet.

This configuration, however, has a drawback for the migration. All the distributed programs and data have to be relocated to one server. This can lead to many changes in the application programs.

- A three-tier configuration where the client connects two servers, the branch office local server and the bank mainframe server, one also running the Web server.

  This configuration is similar to the client/server configuration, keeping the same distributed structure in the two servers. Only the processes and data located on the client machine need to be migrated to one or the other server. Therefore the migration process is easier.

  This configuration may need more administration, maintenance and system management than a two-tier configuration. But it is not more than what have existed with CS92 and the same support team with appropriate education can continue to support NC97.

To respect one of the project objectives, maximum reuse of the CS92 application, we decided to use the same basic hardware configuration for NC97 as had originally been used in CS92. There were some variations, such as no longer needing workstations with storage capacity.

6.2.2 Software

Concerning the software, the decision was to take advantage, if needed, of the new functions of the system software that were used in 1992. The other decision was to take advantage of the new development technologies especially in the rapid application development area and to use the tools that support it.

6.2.3 Communications

The Internet uses TCP/IP as the communication protocol. We also wanted to avoid multiple communication stacks in our environment. Therefore, the communication protocols changed from SNA LU 6.2 and NetBIOS to TCP/IP as shown in Figure 20.

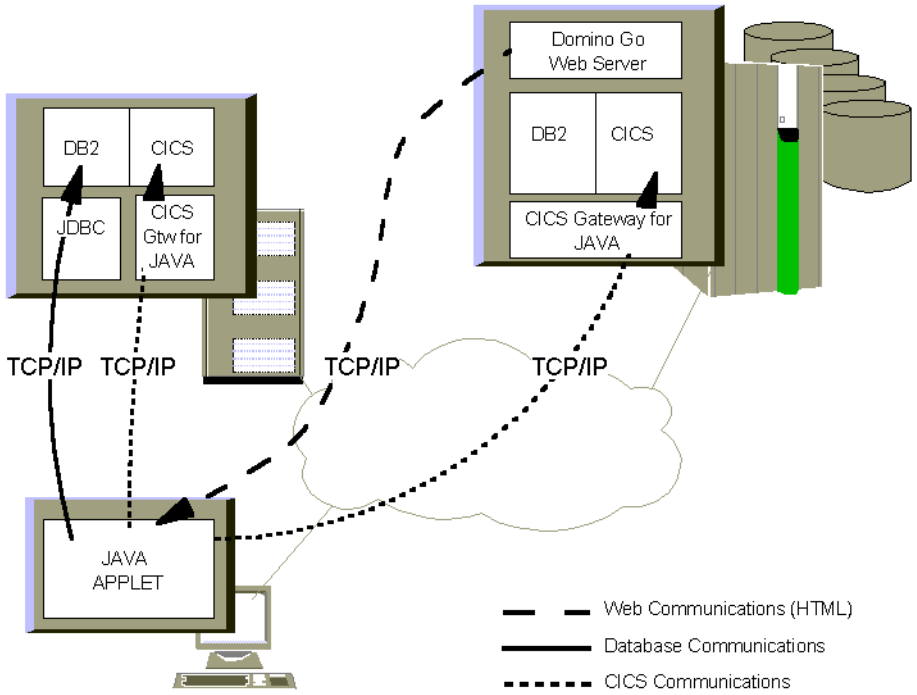


Figure 20. NC97 Communication Configuration

In this environment, all communications are using the TCP/IP protocol over the intranet and the Internet. Compared to CS92 communication configuration (see Figure 2 on page 4), this new configuration is much simpler to manage (one communication stack). It also requires less resources (memory and CPU) than the previous configuration.

6.3 Application Design Tasks

The following tasks are fundamental to any application development, regardless of the framework within which the application is developed.

6.3.1 Application Design

In a migration project, these tasks have to be performed to validate the previous design, but do not need to be started completely anew:

- Revalidate the business needs.

  In a migration project, the business needs would probably have changed between the two versions of the application. We then need to revalidate the business needs that this application addresses.

- Revalidate the data model

  If the business needs have changed, then the data model that describe the data to a business and the relationships among the data may also have to be changed.

- Check the business processes

  The next step in the development process of the application involved identifying and analyzing the business processes, and to group them into

major business processes areas. This was a joint effort between the application designers and the business analysts and the end users.

The business map identifying the data entities that each process requires and establishing the business rules for each process has to be confirmed with the new environment.

- Check the specifications for requirement

  Using the business map as a framework, the requirement specifications of the application are produced for the business areas that were considered.

  These requirement specifications have to be checked against the new environment.

In our migration project, we considered that the environment did not change so that we could use the same hypothesis.

6.3.2 Outline Design

An outline design of an application is produced from the requirement specifications: each identified business process is transformed into application tasks. In the case of a migration project, this transformation was performed when the application was first developed, so the task now is to verify that the outline design is still valid and able to be implemented.

The application tasks can then be analyzed to determine which tasks require application function to be written. However, in the case of a migration project it is typical to want to keep as much of the existing application function as possible. Reasons for this can be:

- Operational stability

  The existing mainframe function can be optimally tuned for performance, fully integrated into the existing production schedule, and have efficient housekeeping.

- Resource constraints

  It is possible that all-COBOL skills, for example, are employed in today's environment on year 2000 work on legacy systems.

- Degree of change

  The more change there is in an application, the longer it takes to change, and the more risk is associated with it.

The degree of change influenced our decision to keep the COBOL code. Approximately 60% of CS92 was written in EASEL and C SET/2, which meant if we could keep the COBOL code, then 40% of the application would not be subject to change. Keeping the COBOL code drove the definition of what outstanding application function had to be implemented in Java, and to an extent also drove the flow of the user interface. Because of the modular design of the original application, it was possible to work back from the individual building blocks of the COBOL code, without being constrained by large pieces of functionality contained within one module. Thus, it was easier to design the flow of the application, and to define exactly what functions needed to be developed in Java.

6.4 Presentation and Application Separation

The design principle of separating the presentation layer from the application business logic is still relevant for network computing. This separation was achieved quite easily in the migration project because CS92 had previously made the distinction between presentation and application. The presentation layer was written in EASEL and the application layer in COBOL. We retained the application layer and rewrote the presentation layer in VisualAge for Java.

The decision to have application logic performing all data manipulation became a little blurred for us, because in CS92, EASEL code was used not only to handle the presentation logic, but also to handle some business logic. For example an EASEL routine calculated the total amount payable so that it could be displayed to the user. This calculated piece of data was then passed back to the application logic for storage on the database, without any further calculation. Chapter 6.7.2, “Updated Function” on page 68, discusses in further detail what we decided to do with this function, which did not clearly belong to one layer or the other.

6.5 Data and Function Placement

There is still a fundamental requirement to have location transparency for applications, and hence as much flexibility as possible. It is important to be able to change the location of data or function without changing application code.

6.5.1 Data Flexibility

Full transparency of data is not always easily attainable. In the case of CS92, for example, a major part of the DB2 table design was to determine where the data should be placed. As a result, tables that were specific to a geographic location, such as a bank branch, did not carry a branch identifier. If this sort of table is migrated from the server to the mainframe, an extra column must be added to the table, and application code must change accordingly.

6.5.2 Function Flexibility

Transparency of application function is achievable by physically moving the code from one platform to another, or by using system software such as CICS or DB2 to perform remote access. The location of code should not be volatile, but requirements such as performance tuning and load balancing may necessitate the moving of code; hence, means of routing client requests to the appropriate functions must be provided.

6.5.3 CS92 Application Request Manager

CS92 used the application request manager (ARM) to support the flexibility requirement. The ARM stands between the application logic on the client workstation and the distributed system services such as access to DB2 data or call to a CICS transaction. It has two functions:

- It verifies that the application logic is requesting a known service and using the correct parameter for that service.
- It routes the request through the system service interface that handles the particular service requested.

This architecture was implemented in CS92 for two reasons.

- It served as the interface between the EASEL and COBOL code.
- The ARM protected the client from needing to know the location of server code.

We do not need this function in our migration because the Java code can make the equivalent calls via the CICS gateways.

Client protection is more relevant to this application routing discussion. For example, the client requests the service GetCashierID, so the ARM determines which program it requires and what service provides it. The developers recognized that directory services external to the ARM should be used to hold the precise location information, but time constraints prevented them from implementing this.

6.5.4 Routing Techniques

In migrating the application we decided to place function on the same platform as the data. We also decided not to migrate the ARM. These migration decisions had implications for the Java code as the router had to decide for itself which CICS Java Gateway the ECI request should be routed to, and the name of the COBOL program that would perform the data access. Using object oriented techniques a router object can be implemented to encapsulate this information. The router implementation can be changed depending upon how the information is made available.

Here are four possible solutions:

- The applet can be provided with information about the location of programs and servers, either at development time or run time. During development, the information is hard coded into the router object. At the end of development, the router can be altered to read the information from run-time parameters. The run-time parameters are coded in the HTML applet tag.
- In the existing application, each DB2 COBOL program is suffixed with a letter that identifies the location of its DB2 data. The Java code must use this metadata to make its routing decision.
- Maintain a centrally placed DB2 table holding the location of the CICS Gateway for Java for each function implemented by COBOL code, and update the encapsulated routing function described above to access this table.
- The same design considerations are still relevant, and there is now a protocol defined, called lightweight directory access protocol (LDAP). It is a standard protocol used by clients to access a directory information server. For our configuration we would need to define the services being provided by the CICS and DB2 servers to the directory server. The client would then request information about the services using the LDAP protocol.

There is still a lack of full system support for this area. We decided, in the interests of time, to go with the hard-coding option (see Chapter 7.5, “Router” on page 79.)

6.6 Data Placement

The task of establishing where to place data is much simpler in a network computing application because there is only one place to store the data, unlike the three-tier model of the client/server application. In the case of a migration project, such as ours, however there was data stored previously on the client. This data and its associated DBMS tables have to be moved from the client to a server machine, whether a mainframe or a local server.

One reason for having data on the client is that it is specific to the user of that workstation. In CS92, this was the case with the currency and currency denomination tables, which were specific to a cashier. When these tables are migrated to the server machine, an additional column must be added so that specific instances of a currency can be identified with a specific cashier. The task, therefore, involves listing all the tables stored on the client machine and performing data analysis to determine what new key information must be added to a table to uniquely identify a row.

Read frequency is a second reason for holding data on the client. In the case of CS92, the commission rate was an example of data duplication, in that the table holding the single commission-rate row was held on the client, the local server and the mainframe server. This was done because, although the commission rate was maintained and updated very infrequently on the mainframe, it was read frequently on all three platforms. When reference data is held on the client which is a copy of data held and maintained elsewhere there is no need to migrate it from the client.

If, however, the purpose of holding frequently read data on the client is to improve performance, then the design phase of a network computing development must provide an alternative solution. One option is to download the reference data at the point where the applet first starts up; another option is to retrieve the data only when needed. Clearly, both approaches have performance implications, so we must consider factors such as the amount of data and its complexity.

6.6.1 Reference Data

In terms of complexity of installation and configuration, it is undesirable in one transaction processing application to have two ways of accessing the DB2 data. However, there are additional considerations when migrating an existing application. In a client/server environment, it is customary to hold reference data on the client machine for ease and speed of access. The data is usually maintained on the server machine and downloaded to the client. The frequency of this downloading is dependent on the volatility of the data.

In a network computing environment, however, it is not valid to store reference data on the client because the client machine may lack local storage capabilities. Nevertheless, there is a need to provide drop-down lists and other user selection data. It is quite likely that an existing transaction processing application will not have any application code for downloading reference data, as it is more likely to have used a system utility. As a result, in migrating to network computing, it is undesirable to have to write new server code simply to download the reference data.

When there is a large amount of reference data to download, then megadata problems can occur. The main megadata problems arise when the user enters vague selection criteria against large tables. There is a slight distinction between satisfying a user query and, for example, populating a drop-down list box for user selection, but fundamentally, the problem is always that of an unknown amount of data being returned.

Solving the problem requires quite complex server code. One of the possible solutions is repositioning to avoid retrieving the same data twice, but this requires the same transaction executing more than once. There is further complexity when the server code communicates to the client that it has encountered megadata and the client code must handle the megadata correctly. An additional problem arises when CICS is used to transfer the data, because it has a size limitation of 32 KB, which the megadata might exceed.

In CS92, each cashier holds lists of currencies. The number of different currencies is unknown and can be large: retrieving such data can give rise to megadata problems. The CS92 development team already recognized that megadata was a potential problem and suggested some possible solutions, but did not implement them.

During the migration project, we did not want to add any coding complexity to the COBOL code; we, therefore, tried to solve some of the megadata problems with JDBC.

We decided to use JDBC to download our reference data. The client code requests the data when needed, knows the format it will be sent in, and can display it to the user. Reference data is thus available, without changes to the existing transaction processing programs.

We replaced an existing server program, which retrieves a list of currencies associated with a cashier, with Java code that makes JDBC calls directly to the DB2 data. In this way, complexity is removed from the server code and the CICS limit is bypassed. The Java code can then handle the unknown amount of data in a number of ways:

- List windows with proportional scroll bars
- Context-sensitive selectable list boxes
- Predefined user limits
- Java code displaying only a subset of the returned result set.

6.6.2 NC97 Data Placement

Although the technology exists to support the placement of all data either on the mainframe or on the local server, we decided to leave the existing mainframe and server tables in their current positions to avoid unnecessary changes of the back-end programs.

However, the first decision we made was not to have data held on the client machine. In a network computing environment, the client may be very “skinny” and its location can be completely remote, so it is not realistic to store data there.

As a result, we had to decide where to place this data and our only change was to migrate three tables from the client to the local server.

In CS92, three tables are held on the client only: Cashier, Currency, and Currency Denomination. The Cashier table when held on the client is a single row table, so when it is migrated to the local server, it simply results in a multirow table and changes are not required. In contrast, the Currency and Currency Denomination tables are specific to a single cashier, so when they are migrated to the local server the currency data for all the branch cashiers is held on the same table. Therefore the cashier must be added as part of the unique key so that one cashier's data can be distinguished from that of another.

We could have migrated all existing tables to the mainframe. However, in the same way that the currency information is specific to a single cashier, much of the order information is specific to the branch. Therefore, too many of the tables, and thus the COBOL code, would have to be changed, adding the cashier and the branch as part of the unique key.

Similarly, we could have migrated all the existing tables to the local server. In this case, however, we would not have satisfied the existing data placement requirements. Specifically, in our application, the exchange rate table had to be on the mainframe, because it was here that the centrally placed dealers, maintained the data held on it. Additionally, we would be making changes not justifiable as part of a migration project.

6.7 Function Placement

In the original application, the infrastructure allowed the placement of function on any of the three platforms. For functions that did not access data the principle was to place them close to the user. Equally, however, functions that did access data were also placed on the client: system software was used to access the data remotely. This was to illustrate the flexibility of the client/server model.

A number of considerations influence function placement when migrating to a network computing environment.

- Choice of development language

  For example, Java applets are better suited for downloading and presenting windows, whereas COBOL is better for data manipulation and retrieval.

- Minimal redevelopment

  If the function already exists, can it be reused in its current placement?

- Reuse

  This includes not only reusing existing functions, as above, but also, if new code is being developed, in which position is it more likely to be reused?

- System software

  What system software is available to support function placement on a specific platform? For example, is JDBC available on the client to enable DB2 data to be available directly to the client?

- Speed of communication

  If a three-tier model is used, are there any performance implications in keeping, for example, code on the server machine which accesses data on the mainframe?

In NC97, the COBOL could not be kept on the client, but we wanted to reuse the COBOL function rather than rewrite that part of the application in Java. Therefore, establishing function placement became choosing on which server, local or mainframe, we should put the functions. A logical approach, which minimized network traffic, and minimized reliance on system software was to place the function on the same platform as the data.

6.7.1 Existing Function

The COBOL code in the existing application was placed on the client. In our implementation of the thin client, we had to move the code to either the mainframe or the local server, or both. Figure 21 lists the possible function placements.

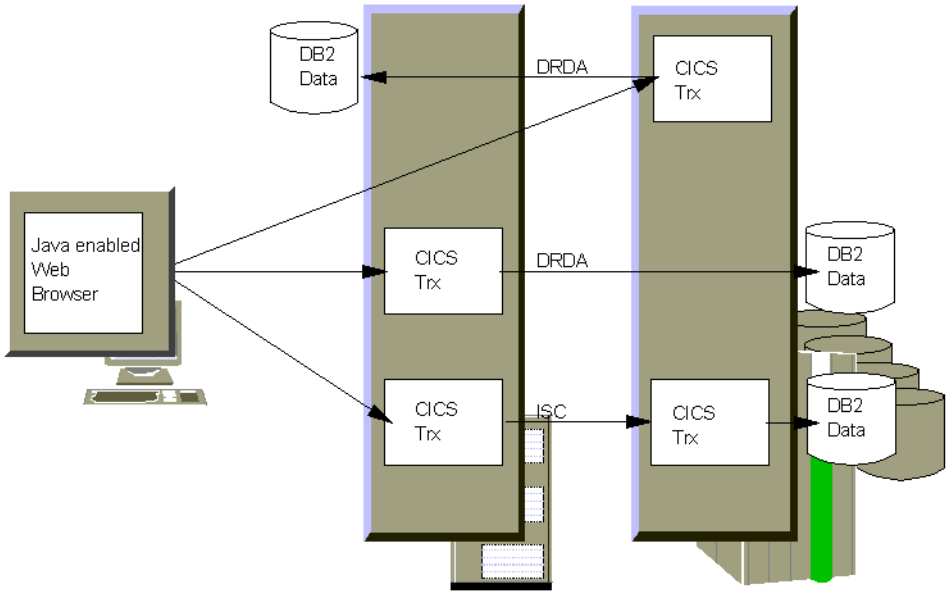


Figure 21. Possible Function Placements

It is possible to place the COBOL code on the mainframe server and to use DRDA to access DB2 data on the local server. Similarly, it is possible to place the COBOL code on the local server and to access the mainframe DB2 data with DDCS and DRDA. We could also use CICS to establish these connections between the two servers.

However, as our client uses Java, we wanted to demonstrate the two versions of the CICS Gateway for Java, without additionally linking the mainframe and the local servers. As a result, we decided to place the COBOL code on the same platform as its related DB2 data as shown in Figure 22.

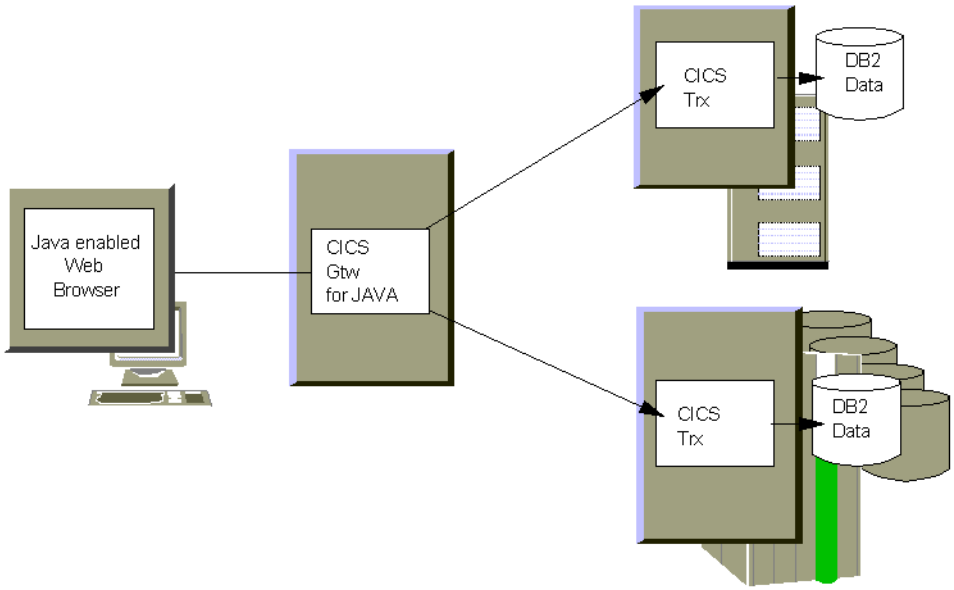


Figure 22. Function Placement Using the CICS Gateway for Java

In CS92, one program retrieves both currency denomination table data, which is held on the local server, and the exchange rate table data, which is held on the mainframe server. Because we decided to place COBOL code on the same platform as its data, we had to split the function of the code. Fortunately, another program could retrieve the exchange-rate table data, so the only coding change we had to make was to remove the exchange-rate retrieval from first program.

6.7.2 Updated Function

In CS92, the EASEL code calculated the total amount payable. We discussed where this updated function ought to be placed. If multiple-client GUIs were to be developed, for example a Lotus Notes version as well as a Java version, then it would be more efficient to code the calculation in a server just once, rather than in each client implementation. However, this approach conflicts with our requirement to reuse as much of the mainframe code as possible without any amendment.

We also discussed whether we should make our client as skinny as possible by not adding function that could be performed at the servers. In this particular case, as the business function requires an immediate display of the total cost, the calculation should be done on the client machine.

We decided to develop the function as a Java applet, as this approach would satisfy a number of our requirements:

- The client remains skinny.
- New function is independent of the existing COBOL code.
- The new function is developed only once.
- The applet supports an open architecture, in cases where an alternative GUI is developed.

6.8 Designing the Application Access

A design task is needed to determine how users will access their network computing application. If the application is to run on an intranet, the client machine can be configured on a one-off basis, by installing a Web browser on the machine, and then defining its default home page as the address of the Web server machine. In an Internet scenario, a client machine would still need a Web browser installed, so that a user could find the home page of the owning organization of the application, as this would be the starting point for the application.

Figure 23 shows the different installation choices we have in installing the application code.

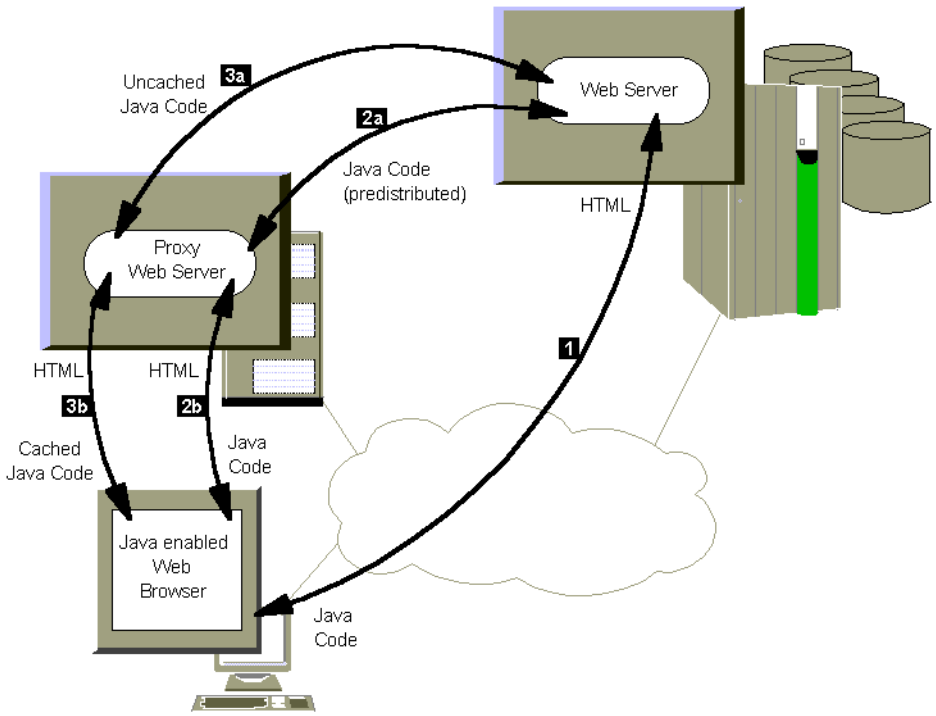


Figure 23. Installation Choice of Java Code

Having brought the user to the required home page, where the Java applet tag is held in the HTML, the next decision is where to download the Java code from. Basically, there are three options available to install the Java application:

- The Java code can be held on the mainframe and be accessed by the client through a web server also on the mainframe.
- An alternative is to define a server machine in the configuration as the Web server. However, if the Java code is developed and maintained on the mainframe, this requires a distribution mechanism from the mainframe to the server.
- A combination of the two approaches is to hold the Java code on the mainframe and define the server machine as a proxy Web server. This means that a client machine makes HTTP requests of its local Web server. If the required Java applets are already cached on the local server, then it satisfies

the client requests. Otherwise, if the code is not cached, then the local Web server makes HTTP requests to the mainframe web server.

Using proxy Web servers in this way, in an intranet environment, can improve the performance of an application, as communications between the local server and mainframe are minimized. The frequency of Java code refresh from the mainframe to the local server can be specified, so that stable Java code will not be frequently downloaded from the mainframe to the local server.

In NC97, since we were operating in an intranet environment, we defined the user's default home page, as the home page of the bank. We decided to define the mainframe as the Web server. We did not define the local server as the Web server because we wanted to minimize the need to distribute code. We did not use proxy Web servers during development of the application because during testing, frequent refreshing of the code on the client was more important than performance.

6.9 Designing for the Web

A network computing application can be Internet- or intranet-enabled. The degree of enablement is a major design decision, in which a number of factors must be taken into consideration.

6.9.1 User Base

Figure 24 illustrates the three types of users a network computing application may encounter.

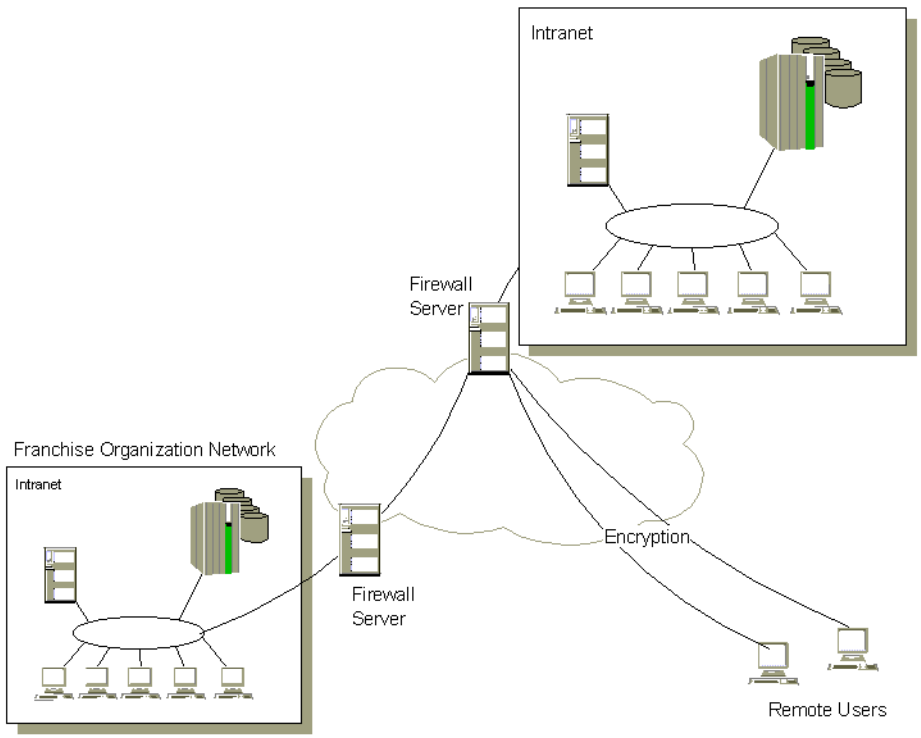


Figure 24. Internet/Intranet Options

The three types are:

- The corporate users

  In NC97, the user base remains unchanged: it is still cashiers in bank branches. Therefore, an intranet solution is more appropriate. Fewer infrastructure changes are required because the bank's enterprise-wide network can be used. From a systems management and security viewpoint the users of the application are known: from the physical location of the users and their machines, through to their user IDs and passwords. However, implementing the application using intranet network computing positions the application for extranet and Internet access.

- The associate users

  A possible evolution of the user base would be to expand it to a franchised company, such as a travel agency, whereby a travel agent could use the same parts of the application as a branch cashier, and would act as a satellite cashier to a particular branch. The travel agency company would have its own enterprise network, but would access NC97 through the Internet; this structure is called an *extranet*. Each enterprise would have a server machine acting as a firewall between its own network and the Internet. The firewall ensures that the data travelling across the Internet is secure. No application amendments would be necessary, as the travel agent is using the same functions as a cashier. The systems management of this extranet remote access would be similar to that of the intranet implementation because the user base would still be identifiable.

- The individual users

  The next evolution would be to allow access by individual customers through the Internet. However, such a change is more fundamental. The concept of a systems administrator allocating user IDs and maintaining passwords is simply not relevant when the enterprise does not know who its customers are. Changes would be required to the business application, as the GUI was originally written for a cashier, not a customer. For example, the cashier ID is a vital data element, which is not relevant when a customer orders currency directly.

6.9.2 System Operation

There are system management implications in extending the scope of a network computing application. If the application is confined to a specific location, it would be possible to make it available to other dispersed parts of the organization—perhaps worldwide.

The result of such an extension is the loss of predictability of the behavior of the environment. The original application may be tuned to a specific and measured workload (transaction rate daily, batch processing nightly) with hardware resources properly sized to accommodate the peaks in usage. Those assumptions are not valid anymore, particularly if the application is opened externally to the Internet.

If the access is worldwide, or at least widely dispersed, the problem of the different time zones should make you reconsider the distinctions you had between daily and nightly operations. You may come to the conclusion that your newly extended application now has to be operational for 24 hours 7 days a week. So, for example, if the nightly operations were reporting ones, you should make sure that they can still run as-is, when, potentially, users can update the

underlying data. Is it acceptable for the requirements of the reporting application? What about the conflicts in locking database resources? If the conclusion is to rewrite the reporting application, is it just a reshape using the same technology, or do you have to switch to a different, more suitable, technical solution? The same applies for the backup activities: if previously they were run during the night, to ensure that no user or application was accessing the data, this assurance can no longer be relied upon. The result is that you would probably need to move to an on-line (or hot) backup of your databases.

Scheduled availability of systems is also affected by time zones, different countries, and cultures. For example, national holidays follow different rules and you should consider this, for example, when planning your computer environment maintenance.

When extending the scope of an application, another consideration is function placement. For example, if the application is accessible to external users on the Internet, you could decide to place, whenever possible, most of the processing on your server machines, in order to please the users with less powerful computers. The other way is to push most of the processing onto the client machines to avoid overloading your corporate servers.

6.10 Designing for Integrity

A design task is necessary to determine what the requirements for data integrity are and how they can be satisfied. It is possible to design an application that does multiple updates, potentially across different platforms, within a single logical unit of work (LUW). However, there are implications to this kind of design that need to be considered. In this section, we look at those implications and discuss considerations specific to network computing.

6.10.1 Designing Logical Units of Work

CS92 had a design that was inadequate to ensure data integrity. The business transaction to create a customer order actually performed two updates to data held on different platforms: the client machine and the local server. Because of the lack of two-phase commit support in the system software used, these updates were performed in two different LUWs.

Because we migrated to a network computing solution, we could not store data on the client. When we migrated the data to the local server, both DB2 tables were on the same platform, so a small piece of redesign and code change solved the data integrity issue by including both updates in the same LUW.

If all the transactions designed for CS92 had been implemented, we would probably have faced similar problems when migrating business functions, which had been split as separate LUWs across the mainframe and the local servers. In this case, again with some redesign and code changes, we could have exploited the two-phase commit support of DRDA, so that COBOL code executing on the local server could access and update data on the mainframe server, thereby allowing DB2 to manage the commit processing.

With network computing there is no system software on the client to manage the synchronization of remote updates. The client machine is “stateless”: it cannot control the updates taking place remotely. Since the updates are always remote

then this control must move to the back end. Once control is moved to where updates are actually taking place, then the synchronization of updates and the size and business contents of logical units of work become a matter of following good design principles.

6.10.2 User Awareness

Although we can design an application to ensure that there is always data integrity, it does not necessarily mean that users are always certain that their data is in the consistent state that they expected.

In NC97, there is a lot of network communication, system software, and application code, between the point when the data is entered on the screen to the point when a confirmation message of success is displayed. At any point, there is a possibility of failure in one or more of the various links in the chain. How do end users know at which point the failure occurred, and therefore whether their update has completed successfully?

Clearly, a message indicating success must be issued by the updating transaction and communicated back to the end user. Success should be, by far, the most frequent case, and such a message will leave the user in no doubt as to the outcome. Other situations are harder to cater for. The Java code on the client can identify that a communications failure has occurred between it and the CICS Java gateway server, and display a message accordingly. “Wait” type logic could be included to determine failures when communication does not return from the Java gateway, but in this case, the Java code would have no detail as to the nature of the failure. In a production-strength application, browsing or refreshing functions would have to be provided to allow users to check on the success of their update.

Informing users of the success or failure of their updates is a usability issue, and the design of this part of the application must be performed in conjunction with an end user representative, so that the cost and complexity of any particular solution is appropriate to the level of requirement.

6.11 Designing for Security

Security is a crucial issue when expanding the user base to include any access from the Internet. In a controlled environment, such as an intranet, the main concern of security is to ensure that the data and the processes that manipulate the data are protected from any unauthorized access and updating. When an application is Internet-enabled the definition of security becomes very much broader:

- Application owner

  Owners of applications still need to ensure that their processes and data are protected, but now have the added concern of not knowing who the users of their application are.

- Application user

  Users of an application have to be confident that the application they are downloading to their machine is reliable and trustworthy, and will not damage their machine in any way.

- Communication

Not only must application owners and users be assured that their ends of the communication are safe and secure, they both need to be confident that the network is secure. In this context, secure means that no third party can masquerade as either the application or the user, and that no third party has access to the communication: it is completely private between the two.

- Applet to Web server

  Enabling SSL security on your web server (see Chapter 4.2.2.2, “Secure Sockets Layer” on page 36) enables Java code to be downloaded securely, providing code integrity. Digital certificates provide mechanisms for authenticity and accountability, allowing users to trust applets. Neither of these mechanisms, however, automatically provides security for any communication that the Java code then performs independently.

- Applet to others servers (CICS and DB2)

  Our NC applet uses the CICS Gateway for Java and JDBC to communicate with the server machines. These products do not provide encryption of the data sent from the applet to the servers. Information like CICS and DB2 passwords flow across the network in the clear. It is up to the applet code that you write to provide security for these connections. Currently there is no easy way to do this for the CICS Gateway for Java or JDBC. Future product enhancements will include providing encryption, in the form of SSL, for the CICS Gateway for Java. For the Intranet, this is not an inhibitor to using the technology. Until encryption is supported, use on the Internet will be restricted to applications that do not require confidentiality of information.

6.12 Designing for Year 2000 Compliance

Any application that has to operate with future dates or which itself must operate into the 21st century must be checked to ensure that it handles dates correctly. The application must be analyzed to see what processing is dependent upon dates and in what way. Then the format of the data, and the way in which the date is manipulated, must be checked. Any shortcomings must be corrected, and the entire application should be tested for its ability to handle change of century dates, and the functionality should be regression tested.

In the case of CS92, the analysis showed that no processing was date dependent. Dates are stored on the DB2 tables, but they are for reference purposes only: no logic depends upon the values held in the date fields. The use of DB2 confirms the confidence that the application is Year 2000 compliant: DB2 fully supports dates in the 21st century and edits all dates used in SQL to retrieve, manipulate or update its data. Therefore, no code changes are required to support the year 2000 and beyond.

Chapter 7. Developing the New Client Application

In this chapter, we explain how we develop the new NC97 applet using the NCF model. We also relate our decisions, giving all the parameters that led us to it.

7.1 Graphical User Interface

To users of a system, the migration from a client/server solution to a network computing application should not detract from their ability to perform the task at hand. The design decisions taken when developing the GUI for a client/server application are still valid if the underlying business processes are unchanged. A migration to network computing is a good time to reevaluate the GUI to determine if any new features that have become available to GUIs since the development of the original GUI will enhance usability. It is important when considering the usability of a GUI to understand the profile of a typical user.

Network computing, in particular the Internet, has enabled companies to open up new channels into their business processes. A GUI for an application that is available on the Internet cannot make the same assumptions about its user base as an Intranet-based GUI.

- Intranet GUI design

  For an intranet based application the GUI is a tool to complete the business process in the most efficient manner. One of the major drivers in moving to a network computing solution is that system support costs can be reduced. Companies have also been looking for reductions in their personnel costs. One of the ways that this is achieved is by using computer systems that require less experienced staff. A good GUI design, in this circumstance, will enable inexperienced operators to become proficient in the business process with minimal training.

- Internet GUI design

  In addition to the usability aspect of Internet applications, companies must be aware that any person using their GUI will be forming opinions about the company. This aspect can be exploited to advantage by displaying marketing information. Many applications on the Internet have a very eye-catching colorful GUI. Such presentations are as much about providing marketing messages as performing any business function.

7.1.1 Java features

A key feature of the network computing model is the use of Java for its platform independence. When producing a platform-independent GUI, a separation must be made between the platform-specific functions and the GUI components.

A factor that the GUI depends upon is the features that are available within Java. JDK 1.0.2 was criticized for problems in its Abstract Windowing Toolkit (AWT), which is the Java implementation of the window components.

In JDK 1.1, Sun Microsystems aimed at solving some major AWT deficiencies, with a strong focus on quality and performance. The AWT enhancements include the beginnings of a richer infrastructure for larger-scale GUI development, including APIs for printing, easier/faster scrolling, pop-up menus, clipboard

support, a delegation-based event model, imaging and graphics enhancements, and more flexible font support for internationalization. Additionally, the Windows (Win32) version of AWT was completely rewritten for improved speed, quality, and consistency with the other platforms.

The Java foundation classes (JFC) are another step toward a complete GUI in Java. The following features that are linked with the new JFC were not available at the time of writing this book. JFC will provide

- Java-to-native-windowing-system drag-and-drop capability. Using such a feature, users will be able to drag from Java to nonnative windows such as Windows and Motif.
- High-level components such as: TreeView, TableView, ListView, PaneSplitter, ToolBar, TabbedFolder, ColorChooser, FontChooser, and StyledText.
- Low-level components such as icon, ToolTip, StatusBar, and MessageBox, as well as sliders, gauges, and spinners.
- Dynamic and virtual views of some components, such as TreeView, so that when you have a lot of data in a tree, you don't have to display one icon per item.
- Support for better displays and type faces. JavaSoft has worked with Adobe to create a Web-based, PostScript-like system for displaying Web pages and applets.

7.1.2 Common Look and Feel

As well as designing the layout and behavior of the windows in the GUI, we had to decide how we could achieve a consistent user interface and what the mechanism for delivering the GUI should be.

Common look and feel is a design principle for GUIs, particularly within an application, and ideally across related applications; but in the case of our application, look and feel common with what? NC97 starts from a Web page, typically the bank’s home page written in HTML.

We had to chose how to implement the network computing version of the GUI between the following solutions:

- HTML

  HTML is not a programming language: it is best used for constructing and displaying Web pages. It is possible to implement the GUI in HTML, but the result would be a strange hybrid. The windows look like Web pages but with many missing features of a GUI, such as tabbing to entry fields. Therefore, there are some serious usability issues.

- Native Java graphics

  There are strong arguments for reproducing the dialog exactly as is. No user training is required, as the GUI will look and behave in a manner completely familiar to the user. However, the client/server GUI was written using Presentation Manager, which is specific to the OS/2 platform. A lot of Java application coding would have to be written to reproduce this appearance exactly. This investment in the mechanics of dialog construction could hardly be justified, especially when the result would be platform specific.

- AWT

VisualAge for Java uses the AWT, which contains basic, portable screen design tools. These tools become platform-specific only at run time. For example, the developer uses VisualAge to define a field as a drop down list box; when this code is executed in a Windows NT environment, the list box's appearance and associated controls are slightly different from its appearance in OS/2. Hence, using AWT gives an application an appearance that is consistent with the platform it is running on.

- JFC

  JFC is a future method of achieving common look and feel planned by Sun Microsystems which intends to have a universal Java appearance that will look the same regardless of the platform it is running on. When this is delivered, it will be another option from which to choose.

We decided to build the GUI using VisualAge for Java, reproducing as much of the existing dialog as possible. When users of the client/server version of the application switch to the network computing version, it will appear very familiar to them. In the same way, if their workstation's operating system changes to Windows NT, the application will behave in the same way, and still look familiar. We chose VisualAge for Java because it is a visual programming tool and runs on a variety of platforms.

7.2 VisualAge for Java

We developed NC97 using VisualAge for Java. This application development tool generates Java code that is the NCF language. VisualAge for Java allowed us to:

- Develop the user interface
- Include the connection to CICS, using the CICS Gateway for Java
- Include the connection to DB2, using JDBC and Net.Data

7.2.1 Application or Applet Decision

As we describe in Chapter 4.1, "Java" on page 31, the Java language can be used to construct Java applets and Java applications and VisualAge for Java supports both. A decision must be taken when migrating to a NCF application based on the following criteria:

7.2.1.1 Application Advantages

Performance is an advantage of developing an application as a Java application rather than as a series of applets. The application can be installed on the client machines on which it is to run, thereby saving on download time.

Security is also simplified. The owners of an application know which machines it is installed on, and are not concerned with making secure connections, as they control both the client machines and the computers to which they are connecting.

7.2.1.2 Applet Advantages

Applets, unlike applications, are not installed on the client machine. Using applets makes the system management tasks easier, because there is no need to distribute, install, or maintain the Java code remotely. If a Java applet is to be used in a production-strength business application, then the applet can become fully Internet-enabled in a robust way, because there is a straightforward method

of ensuring that a client is running the correct version of the code, that is, dynamically downloading at run time.

7.2.1.3 NC97 Decision

During the migration process, we decided to use Java applets. Our design is to make the clients skinny by not installing and administering code on the client machines. As we had already decided to operate the application in an intranet environment, we could exploit that secure environment by enabling the Java applets to connect to both the mainframe from which they were dynamically downloaded, and the server machine where some of the data and associated programs reside.

7.2.2 Applet Design Goals

The design goals that we worked towards are:

- Minimize the changes to the CICS transactions.
- Maintain separation of presentation layer from the data layer.
- Maintain the current GUI screen flow.
- Make it NC compatible.
- Route CICS requests to multiple CICS Java gateways
- Make it executable in as many Web browsers as possible

7.2.3 Applet Prerequisites

NC97 needs to connect to, at least, the local and the mainframe servers (see Figure 19 on page 58). Therefore it requires the enhanced security model that allows network connections to hosts other than the one from which it was served. This is a feature of JDK1.1, which is compatible with VisualAge for Java.

7.3 Object Modeling

As in the client/server model, a key stage during the process of designing the applet is to define a data model that will hold the data to be displayed on the GUI. This data modeling task has evolved during the past few years with the increasing maturity of object-oriented analysis and design techniques. As Java is an object oriented-language, we developed an object model for our application.

Objects identified are:

- A system object to hold current data on orders that are in progress
- A transaction manager to call CICS transactions
- A router object to enable linking to multiple CICS Java Gateways
- A window manager to handle windows that are open.

The size and complication of an object model for an application depends upon the complication of the data model. CS92 was a fat client application with large amounts of local data storage and CICS transactions were not altered. Therefore, NC97 requires a similar data model.

If, however, the application is being migrated from a thinner client/server solution where less data is stored on the client, then the CICS transactions will be available so that less data is required on the client.

7.4 Window Manager

We designed the applet as a set of panels that are connected together to form frames.

As shown in Figure 25, the window manager handle the connection and the application flow.

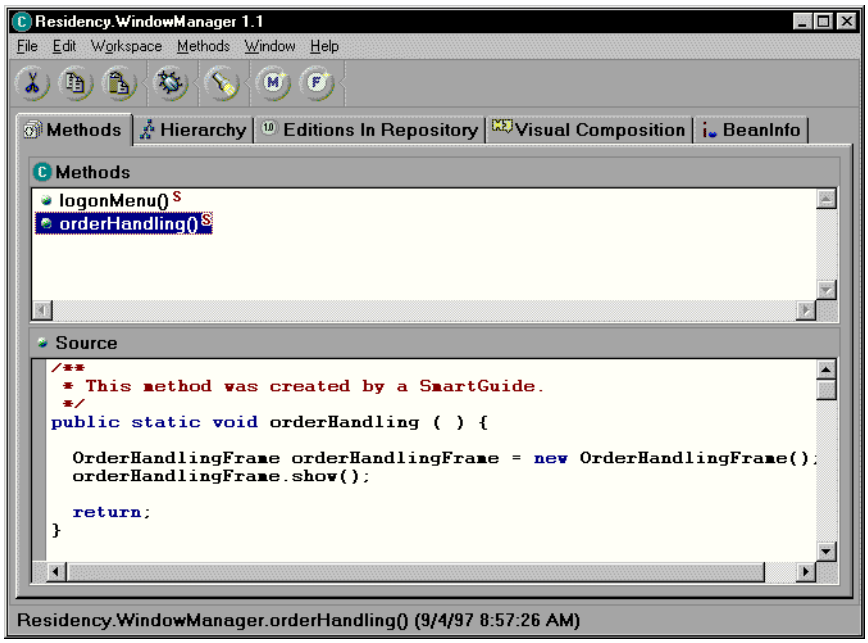


Figure 25. Window Manager Object

Method logonMenu manages the cashier's connection to the application. Using the cashier's identifier and password, the application can select the cashier's data and can also transmit that identification to the back-end systems on the local and mainframe servers.

Method orderHandling manages the business transaction. The cashier is presented a set of screens to handle the business transaction, selling or buying foreign currencies. The method, then, creates the order.

7.5 Router

It is important to be able to change the run-time parameters for the CICS Java Gateway transactions and JDBC requests at execution time. We have implemented a Router object that stores these parameters when the applet is initialized. The parameter values are initialized from HTML named parameters. As our architecture uses two CICS Java Gateways, a decision about which gateway to send the request to is made at run time. This is also the responsibility of the Router object. As shown in Figure 26, we are using a simple implementation for this, a lookup table is defined in the development environment.

The lookup is based on program name which is reasonable in our case. A more sophisticated routing algorithm can be used if required.

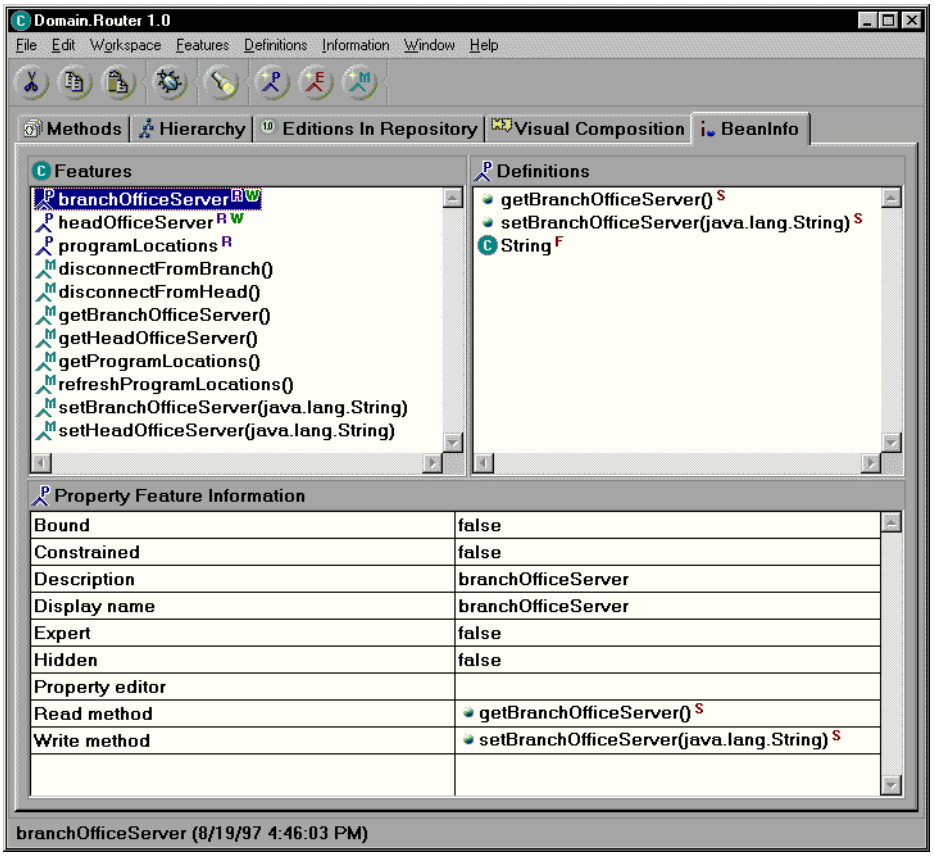


Figure 26. Router Object

7.6 Transaction Manager

To create Java applets that interact with an existing CICS application, you need the following components:

- The CICS Gateway for Java, part of CICS, that provides communication and data conversion between the Java applets and CICS servers
- The CICS Access Builder, part of VisualAge for Java, that simplifies the Java programming by encapsulating the CICS transactions and data as Java beans.

The CICS Access Builder creates a Java bean, which encapsulates the data transferred between the CICS transaction and the Java applet, together with classes to handle the data marshaling and conversion. The CICS Access builder comes with a class library including a bean corresponding to the CICS unit of work, for synchronizing with the CICS transaction.

We used a TransactionManager object to provide separation between the presentation layer and the data layer, therefore removing the need for CICS transactions to be part of the GUI code (See Figure 27 on page 81).

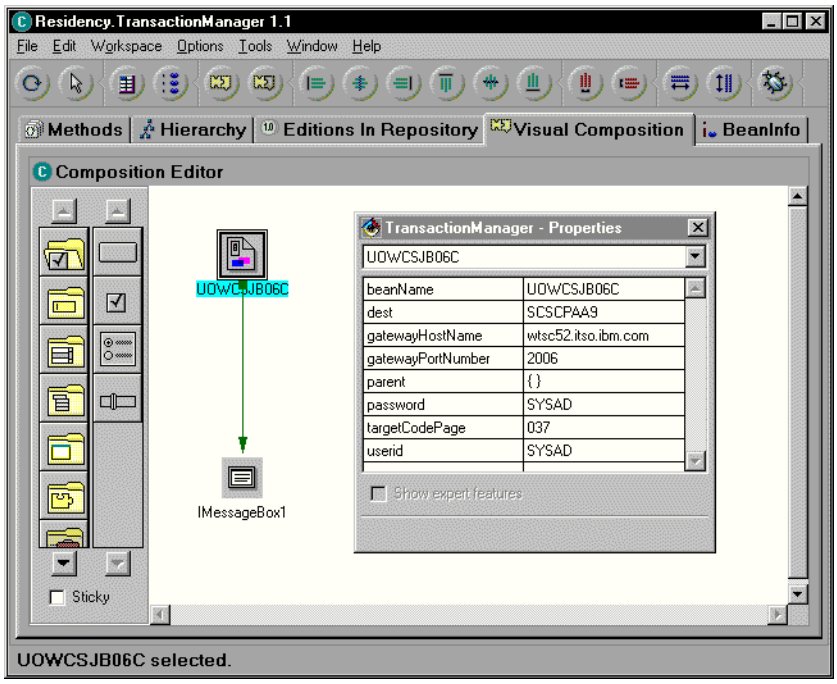


Figure 27. TransactionManager

The CICS Access Builder uses class IVJCicsUOWInterface for the CICS unit of work and the abstract class IVJCicsEciCommArea for the COMMAREA.

As a sample, we describe in the following how we created the Java bean of a CICS transaction. The COMMAREA of our sample transaction is shown in Figure 28.

```
*****************************************************************
* ITSC - San Jose   Client/Server Computing                      *
* CSJ - Foreign Currency Application                CSJCOM05.CBL *
*                                                               *
* DESCRIPTION                                                   *
*       - COMMON AREA FOR EXCHANGE RATE                         *
*****************************************************************
01 DFHCOMMAREA.
   04 PROGRAM-NAME          PIC  X(08).
   04 ERROR-MESSAGE.
      06 MESSAGE-COMMAREA   PIC  X(80).
      06 SQLCODE-COMMAREAX  PIC  X(07).
      06 SQLCODE-COMMAREA   REDEFINES
         SQLCODE-COMMAREAX PIC   9(07).
      06 SQLSTATE-COMMAREA  PIC  X(05).
   04 EXCHANGE-RATE.
      06 C-TYPE-OF-CURRENCY-E       PIC  X(03).
      06 C-EXCHANGE-RATE-BUY        PIC  X(07).
      06 C-EXCHANGE-RATE-BUY-R      REDEFINES
         C-EXCHANGE-RATE-BUY        PIC  ZZZ9.99.
```

Figure 28. COMMAREA

The steps are the following:

1. Create the COMMAREA bean.

   The first step is to parse the COMMAREA, using its COBOL definition, to create the corresponding Java classes. As shown in Figure 29, the CICS Access Builder imports the COBOL file, CICSCOM5, and generates a COMMAREA bean, COMMAREA5, and the associated classes that map COBOL data to Java data, one-to-one.

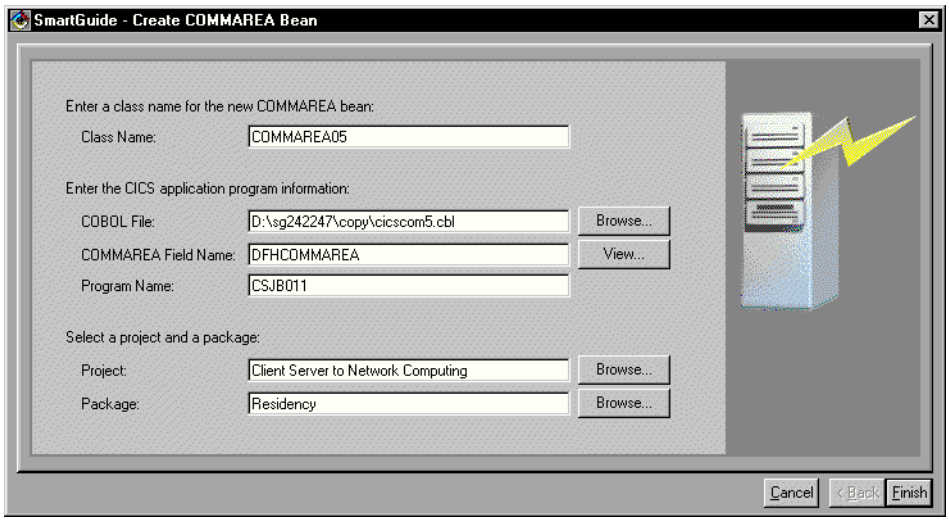


Figure 29. Create COMMAREA Bean

   If the parsing is successful a class that extends IVJCicsEciCommArea is created. Other classes are also created to define the COMMAREA. Our example produces three classes corresponding to the three sections of the COBOL copybooks:

```
COMMAREA05_DFHCOMMAREA
COMMAREA05_DFHCOMMAREA_ERROR__MESSAGE
COMMAREA05_DFHCOMMAREA_EXCHANGE__RATE
```

2. Define the CICS unit of work

   The CICS Unit of work is designed to be programmed visually. To create a Unit of Work bean, an instance of IVJCicsUOWInterface is placed on the canvas of the composition editor. The properties of the bean can be altered. Figure 30 shows the values used in our sample.

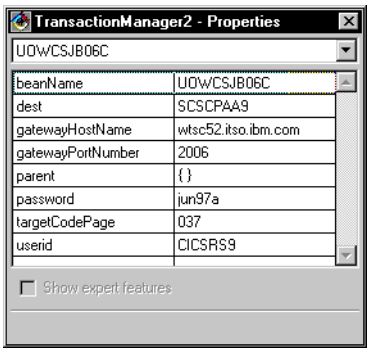


Figure 30. Unit of Work Bean Properties

3. Program the CICS bean.

   Figure 31 shows how we used the CICS bean in our program.

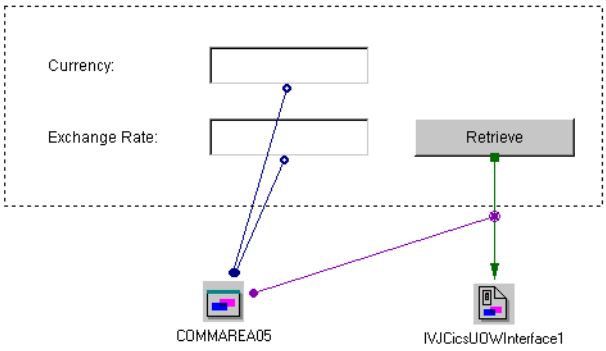


Figure 31. Visual Programming with the CICS Bean

   To run the transaction we use the invokeTxn method of the IVJCicsUOWInterface with COMMAREA05 bean as a parameter.

7.7 JDBC DB2 Access

We used the Data Access Builder tool of VisualAge for Java to create data access classes customized to our relational database tables.

Data Access Builder generates beans that access database tables using JDBC. Both DB2 and ODBC JDBC drivers are supplied. We used the DB2 JDBC driver.

The applet needs to access a DB2 table to provide the values for a drop-down list box. We had to decide when the JDBC call should be made. As this data is required when a window is displayed, we have issued the JDBC call when the window is opened. For performance reasons, it may be better to issue the JDBC call upon logon and cache the result so that when it is accessed a read of the cached version only is required.

As with CICS transactions, it is necessary to set run-time parameters at execution time. The parameter values are held in the router class and are set for run-time invocation in the DataManager class.

As a sample, we describe in the following how we created the Java bean to access the branch table. The columns of the DB2 table are shown in Figure 32.

```
CREATE TABLE NCM.BRANCH
       ( BRANCH          CHAR(02) ,
         BRANCH_NAME     CHAR(40) ,
        BRANCH_ADDRESS   CHAR(80) )
```

Figure 32. Definition of BRANCH Table

The steps are the following:

1. Select the table.

   After selecting the package in which the JDBC data access bean is to be created, complete the Smart Guide to create a schema to the database. This defines the type of database (DB2 or ODBC) and the database that is accessed. When selecting the tables in the schema, a DB2 CAE is required on the VisualAge for Java development machine. A schema for the selected tables is displayed in the Data Access Builder window. In our example we are using the BRANCH table (see Figure 33).

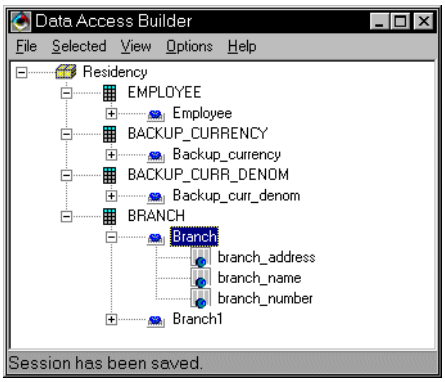


Figure 33. Schema Mapping

   A mapping is created for the schema upon its creation. The location of the database and the driver used are defined in the properties of the mapping (see Figure 34). These must be set up before testing the connection.

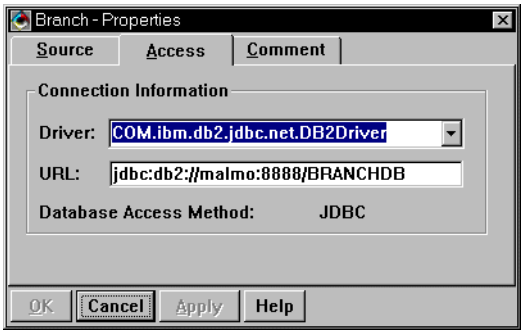


Figure 34. Branch Bean Properties Window

   The driver depends on the type of database being accessed and whether the code is run from an applet or an application. For JDBC access to DB2 the

driver is COM.ibm.db2.jdbc.net.DB2Driver for applets and COM.ibm.db2.jdbc.app.DB2Driver for applications.

2. Generate the bean.

   Access to the table can be generated from this mapping. This is achieved by using the Save and Generate function. The generated classes provide basic access to the database:

3. Program the data bean.

   Figure 35 shows how we used the data bean in our program. The generated BranchAccessApp can be used to access the database.

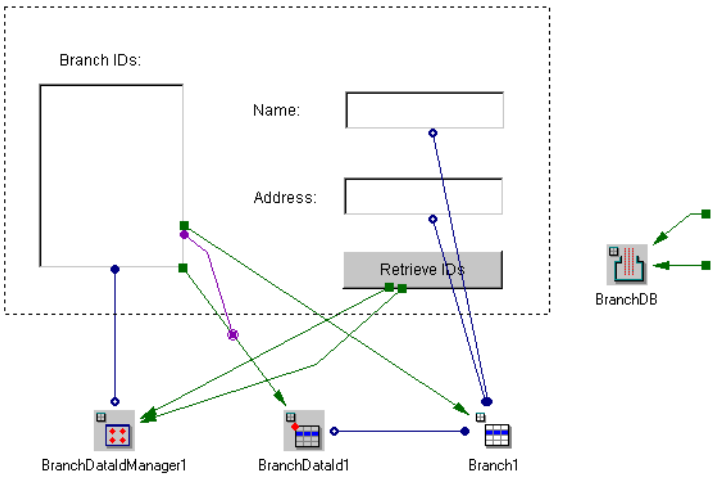


Figure 35. Visual Programming with the Data Bean

7.8 Net.Data

While designing the migration to NC97, we wanted to show that we could add value to the application in an easy and inexpensive way, by leveraging the strengths of the various products in our infrastructure.

We produced a new data warehouse function that is accessed from the bank home page. This function satisfies the business need of both branch cashiers and central office staff, to know about the demand profile for a certain currency, for a certain branch, during a certain period in the past. This knowledge helps them to make more accurate forecasts of future demands.

This type of application requires an approach that entails the creation of specific data structures separated from the operational data (sometimes in the form of the so-called *data marts),* and access to that data from the Web.

7.8.1 Implementation

We achieved the development of this new function using Net.Data to access our data and an existing piece of code, a Java applet, to display it. This example combines the strengths of Net.Data and Java and illustrates two of Net.Data's main features:

- Easy plug-in of SQL statements

- Interface to other language environments

It also illustrates two main Java features:

- Code portability (platform independence)
- Code reuse

The Net.Data macro issues SQL statements and then invokes a Java applet to display a user-selectable type of chart.

To stress the concepts of portability and code reuse in Java, we used an existing applet that was available on the Web. This is a simple Java applet available through the Net.Data home page on the Web.

This piece of code is capable of displaying charts when invoked with the proper parameters. Within our implementation, this has two important advantages:

- We had no need to modify the code of the applet, we just had to provide it the correct parameters.
- We also did not install the applet on our Web server; NC97 downloaded it as needed from the Net.Data Web server.

Figure 36 shows the complete picture of our Net.Data implementation.

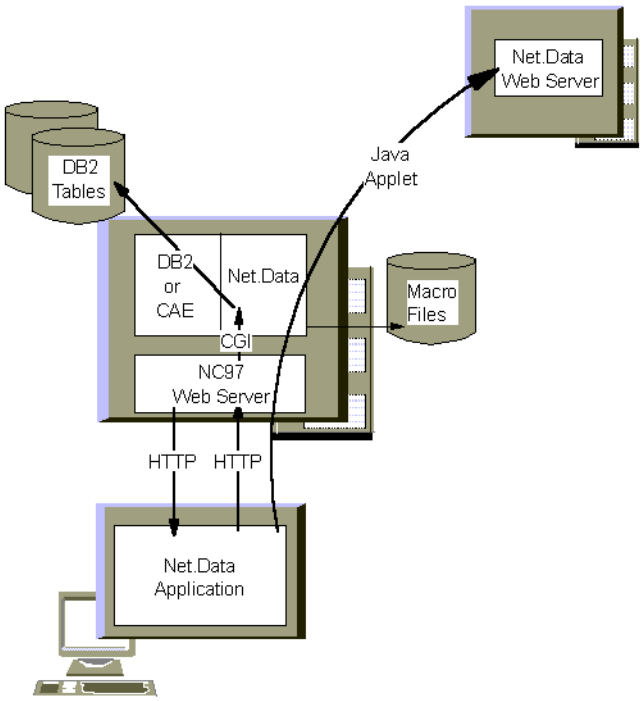


Figure 36. Net.Data Implementation

7.8.2 Net.Data Macro

Net.Data simplifies writing interactive Web applications by using macros to add logic, variables, program calls, and reports to HTML. A macro is a text file containing Net.Data macro language, HTML, and statements needed to work with data, such as SQL or PERL. These macros combine the simplicity of HTML with the dynamic functionality of Web server programs, making it easy to add live data

from local or remote databases, flat files, applications, and system services to static Web pages.

The Web server starts Net.Data as a CGI process or as a Web server API thread by calling Net.Data as a DLL or shared library when it receives a URL that refers to the Net.Data macro. The URL includes information for Net.Data, including which macro file to process. When Net.Data finishes processing the macro file, it sends the resulting HTML to the Web server, which passes it on to the Web client, where it is displayed on the browser.

Refer to Appendix D, “Net.Data Macro” on page 103 to get the complete environment to run the macro.

Step 1: Request (the URL).
To start the application, the user enters the URL where the Net.Data macro is stored:

```
http://china.almaden.ibm.com/cgi-bin/db2www.exe/ncmndb.mac/input
```

You can break the syntax of the URL into four pieces:

- Hostname:*china.almaden.ibm.com*
- Program to be executed: *.../cgi-bin/db2www.exe*
- Macro to be invoked: *ncmndb.mac*
- Section to be processed within the macro: *input*

Here we are using Net.Data in its CGI-style fashion (the URL would slightly differ in case of using the Web server’s native API). The Web server locates the executable in its appropriate CGI-bin programs directory. This directory contains the executable module, db2www.exe, and has be specified during the Net.Data installation process. Net.Data locates the macro as for the MACRO_PATH in its configuration file, db2www.ini, and then executes its *input* section shown in Figure 38

```
%{*********************************************************************/
/*  HTML section: INPUT                                                */
/*  Description: Provides input form for user to choose the information*/
/*               for the database query                                */
/*********************************************************************%}
%HTML_INPUT{
<html>
<head>
<TITLE>$(sampleTitle)</TITLE>
</head>
<body>
<H3>$(sampleTitle)</H3>
<b>
<p>
Inquiry arguments: BRANCH/CURRENCY/PERIOD OF TIME<br>
<ul type=square>
 <li>Select Branch from the drop down list box
 <li>Select Currency from the drop down list box
 <li>Enter start/end time you are interested
</ul>
</b>
<p>
<FORM METHOD="POST"
ACTION="report">
<HR>
Branch:
<P>
@qryDB_B()
<P>
Currency:
@qryDB_C()
<p>
Date from (YYYYMM):
<INPUT TYPE="text" NAME="DATE_FROM" SIZE=6>
Date to (YYYYMM):
<INPUT TYPE="text" NAME="DATE_TO" SIZE=6>
<p><HR>
<P>
<INPUT TYPE="submit" VALUE="Submit Query">
<INPUT TYPE="reset" VALUE="Reset input">
</FORM>
<P>
</body>
</html>
```

Figure 37. Net.Data Input Section

The SQL sections qryDB_B and qryDB_C return the branch and currency lists. The DATE_FROM and DATE_TO, specifying the time frame, are simple text entry fields

Step 2: Inquiry (the input form)

The Input section generates an input form, which is sent back to the user. As shown in Figure 38, the form allows the user to enter four values:

- Branch, selected in drop-down listbox
- Currency, selected in a drop-down listbox
- Period of inquiry start, in an entry field
- Period of inquiry end, in an entry field

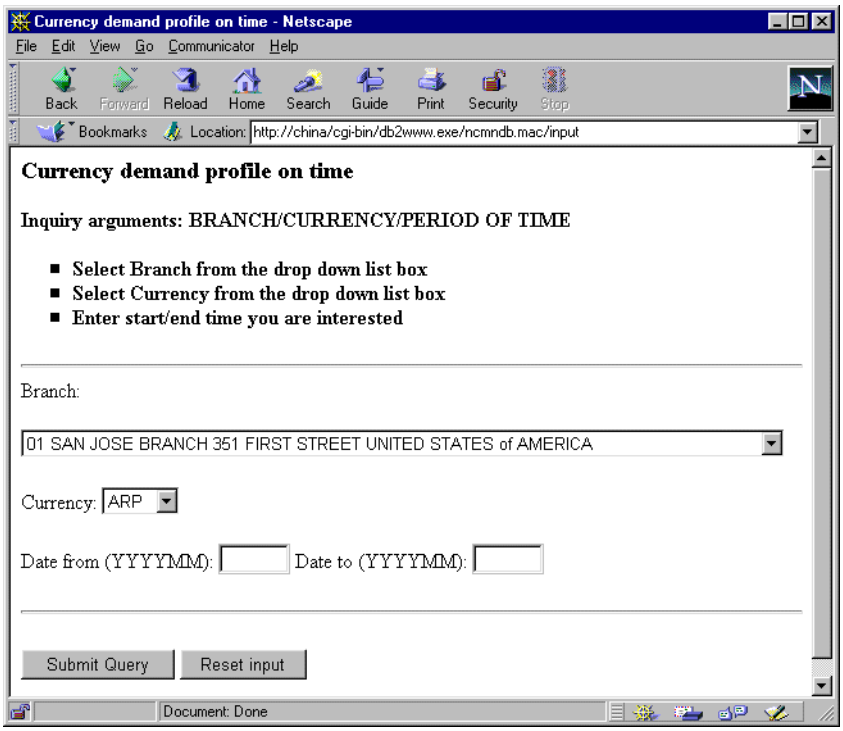


Figure 38. Net.Data Function Input Form

After making his/her selections, the user clicks on the **Submit Query** button. This causes the execution of what is specified as ACTION in the input form; in our case, this means to process the *report* section of the macro.

Step 3: Delivery of the results, Part I (report form, tabular output)
As shown in Figure 39, the report section starts the SQL section *Process*.

```
%{*************************************************************************/
/*  HTML section: REPORT                                                 */
/*  Description: Queries the database to create the pie chart via a JAVA  */
/*               applet                                                  */
/*************************************************************************%}
%HTML_REPORT{
<html>
<head>
<TITLE>$(sampleTitle)</TITLE>
</head>
<body>
<H3>$(sampleTitle)</H3>
<p>
Following are the results of your query.
@Process()
<P>
<hr>$(mainfooter)
</body>
</html>
%}
```

Figure 39. Net.Data Report Section

Figure 40 on page 90 shows the content of SQL section *Process*, which:

1. Executes the actual SQL statement to get the data

2. Displays the result set in a tabular format.
3. Stores the result set in table *currtable* for subsequent applet processing.
4. Calls the applet to display the chart.

```
%{*************************************************************************/
/*  Function:  Process              Language Environment: SQL              */
/*  Description: Queries the database and invokes the Java applet          */
/*               within the report section                                 */
/*************************************************************************%}
%function(DTW_SQL) Process() {
select yyyymm_order, sum (amount) from $(TABLEQLFR).ORD_HIST_VIEW
   where branch_number    = SUBSTR('$(BRANCH)',1,2)
   and branch_type_curr   = '$(CURRENCY)'
   and yyyymm_order      >= '$(DATE_FROM)'
   and yyyymm_order      <= '$(DATE_TO)'
 group by yyyymm_order
 order by 1
%REPORT{
   %IF (RETURN_CODE == "0")
    <table border=2 cellspacing=40 cellpadding=0>
    <tr> <th colspan=2 bgcolor='a7a7ff'>Currency: $(CURRENCY) From: $(DATE_FROM) To:
$(DATE_TO) Branch: $(BRANCH)
    <tr> <th bgcolor='ffaacc'>Tabular Data <th bgcolor='ffaacc'> Java Applet
    <tr>
    <td align=center>
    <table border=1 >
    <tr><td>Month</td> <td>Requested</td>
  %ENDIF
  %ROW{
   <tr><td>$(V1)</td> <td align=center>$(V2)</td>
  %}
  %IF (RETURN_CODE == "0")
    </table>
    <td>
    @DTWA_ChartUI2(ChartUI2.codebase,ChartUI2.width, ChartUI2.height,ChartType, numrow,
currtable, name)
    </table>
  %ENDIF
 %}
%}
```

Figure 40. Net.Data Process Section

Note that *currtable* is not a relational table; it is just a Net.Data data structure that contains the result of the query.

Step 4: Delivery of the results, Part II (report form, applet output)
The applet code is downloaded from the its origin (Web server in Boulder, Colorado) and then executed. Its input parameters comprise *currtable* and some information about the way to display the chart. Figure 41 on page 91 shows the final result. The negative value reported for January and March 1996 means that during that month, customers returned to the branch more Australian dollars than they requested from it.

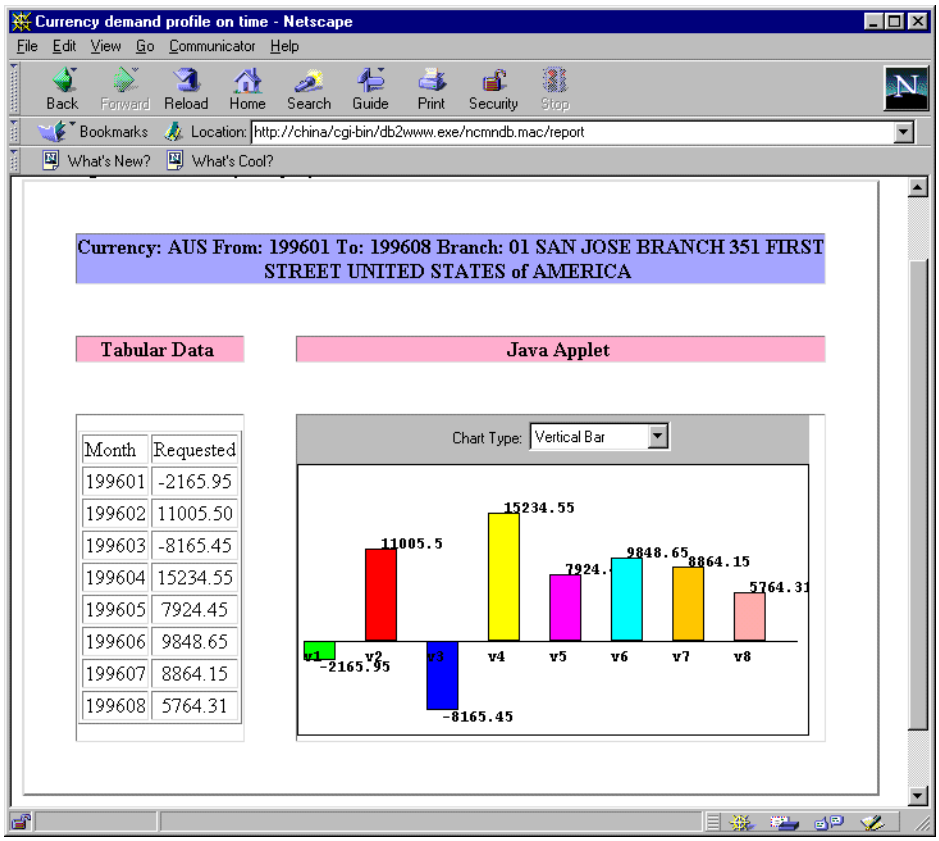


Figure 41. Net.Data Function Output

This process highlights how easily the Java applet, a single piece of code centrally stored and maintained, can be used to display a simple chart for any kind of two-column tables, without any kind of spreadsheet product actually installed on the client machine.

7.9 COBOL Changes

Previously, most of the COBOL programs resided on the client. In this migration, we placed the code on the same platform as the DB2 tables. The code was originally written using Micro Focus COBOL/2. We had to recompile all the modules using IBM VS COBOL II on the mainframe server and IBM VisualAge for COBOL on the local server.

Therefore, some changes to the COBOL code and copybooks were essential in the following areas:

- DB2-related changes
  - Additional columns

    The Currency and Currency Denomination tables were migrated from the client to the server machine. As a result, an additional column was added to the unique key of both tables, namely, CASHIER_ID. Any programs that accessed the tables by the unique key had to have the new column added into the SQL statement. Also, the copybooks associated with the changed programs had to be updated to include the new field.

- SQLGSTRD API

  The API calls to stop and start databases remotely are not needed when the code is on the same platform as the data, and only one database is accessed. See Chapter 4.4, “DB2” on page 41 for details on the reasons behind this. As a result the APIs and their related copybook were removed from both the mainframe and server programs.

- CICS/DB2 synch-point coordination

  A CICS user installable module (UIM), FAARMDBM had to be linked into all the programs that ran on the server, to coordinate the commit points between CICS and DB2. Otherwise the thread to DB2 would still be active when the CICS program finished. The module definition file for each of the server updated programs had the following import statement:

  ```
  IMPORTS SQLCALL=FAARMDBM.SQLGCALL
  ```

  as described in the CICS OS/2 manual.

- COBOL related changes

  The syntax of the redefines statement in the copybook had to be changed from the Micro Focus version (see Figure 42) to IBM COBOL on all platforms (see Figure 43).

```
04 ERROR-MESSAGE.

   06 MESSAGE-COMMAREA   PIC  X(80).
   06 SQLCODE-COMMAREAX  PIC  X(07).
   06 SQLCODE-COMMAREA   REDEFINES
      SQLCODE-COMMAREAX  PIC  9(07).
   06 SQLSTATE-COMMAREA  PIC  X(05).
```

Figure 42. Micro Focus COBOL Redefine Statement

```
04 ERROR-MESSAGE.

    06 MESSAGE-COMMAREA   PIC  X(80).
    06 SQLCODE-COMMAREAX  PIC  X(07).
     06 SQLCODE-COMMAREA   PIC  +999999
        REDEFINES SQLCODE-COMMAREAX.
    06 SQLSTATE-COMMAREA  PIC  X(05).
```

Figure 43. IBM COBOL Redefine Statement

7.10 Change of Platform

In the previous application, the final suffix letter of the program name (L,S,H) identified where the DB2 tables resided. In this implementation, we maintained the naming convention, so that the client programs that were previously suffixed with an L, are renamed to be suffixed with an S to reflect the move to the server platform.

One program had been coded in the client/server implementation to demonstrate that both server and mainframe DB2 tables could be accessed from the same

client based program. Because of our data and function placement decision we needed to change this program to remove all the logic that accessed and retrieved Exchange Rate data. Another program on the mainframe already provided this functionality.

Two programs implemented the create customer order function and demonstrated the difficulties of maintaining data integrity when updates were across platforms (the client and the server). Because we migrated the client data, to the server, the two modular update programs can both be called by the same CICS COBOL program, and so included in the same LUW. This also involved combining the two specific copybooks into one.

Appendix A. Domino Go Web Server for OS/390 Operations

In this appendix, we describe the use of the Domino Go Webserver for OS/390 to distribute the applet code to Web browsers. In our particular network computing configuration, the Web server is centrally located at the main office on the host server.

In order for a user (cashier) to start the application, a web browser must be directed to open an HTML page on this server. Once the page is accessed, the Java code is downloaded from the server and executed in the browser.

After installation, you start the server using the startup procedure IMWEBSRV member of the system PROCLIB data set. The OS/390 operator start command is:

```
s IMWEBSRV
```

This command starts the Web server using the configuration file httpd.conf. We added a Pass directive for the CICS Gateway for Java which references actual directories in the HFS. In order for the CICS Gateway for Java files and documentation to be served by the Domino Go Webserver, you need a Pass directive pointing to the installation directory in the HFS:

```
Pass /jgate/* /u/java/JGate/*
Pass /* /usr/lpp/internet/server_root/pub/*
```

This assumes that the CICS Gateway for Java is installed in the directory /u/java/JGate. The documentation is contained in the directory JGate/html/doc, with a file called index.html as the main starting page. Once the server has been started, you can use the URL:

```
http://your.server.address/jgate/html/doc/index.html
```

Under the JGate directory structure installed in the HFS is a classes directory. This contains the CICS Gateway for Java class files that are served by the Domino Go Webserver. You need an HTML page that contains an APPLET tag so that a Web browser can invoke the applet. There is a test Java applet that is provided with the CICS Gateway for Java and to use it you need the APPLET tag that follows:

```
<applet code="ibm.cics.jgate.test.TestECI.class" width=520 height=290
codebase="/jgate/classes">
</applet>
```

It is the codebase parameter that instructs the browser where on the server the applet class files are. When this tag is read by the browser, it forms a URL request using /jgate/classes/ as the base. On the server, this is mapped to /u/java/JGate/classes by the Pass directive. Java class files are also stored in a directory structure that matches their package naming convention. So in the above example, TestECI.class is in /u/java/JGate/classes/ibm/cics/jgate/test directory. The full URL the browser requests when it reads the applet tag is

```
http://you.server.name/jgate/classes/ibm/cics/jgate/test/TestECI.class
```

Appendix B. CICS Gateway for Java — Installation and Setup

We downloaded the CICS Gateway for Java from the CICS Web site.This site also contains detailed information about the CICS Gateway for Java and instructions for operation on workstation platforms. Using a Web browser, you receive a registration form, and you can download the product for the specific platform you require.

The product files are packaged as a single compressed tar file that needs to be copied to the hierarchical file system of the OS/390 OpenEdition. The tar file must then be expanded into a set of directories.

B.1 Configuration

Before the CICS Gateway for Java can be run, you must edit the startup script, in the JGate file, to reflect the CICS TS libraries on your system, your EXCI options table, and the HLQ variable:

```
EXCI_OPTIONS="CICSTS12.PROG.LOAD"
HLQ="CICSTS12.CICS"
EXCI_LOADLIB="${HLQ}.SDFHEXCI"
export STEPLIB=${STEPLIB}:${EXCI_OPTIONS}:${EXCI_LOADLIB}
fig: example of JGAte filr varables
```

You must specify the WEB=YES SIT option to enable the business logic interface. To support CICS Gateway for Java, the DFHJVCVT program definition must be installed on CICS TS.

Java applets and applications do not execute in an EBCDIC environment, even on the OS/390 Java virtual machine. Unless the applet can generate COMMAREA data for the CICS program in the correct format and code page, a DFHCNV table entry is required for the program.

In order for an OS/390 program to use the EXCI to communicate with CICS TS, definitions for the connection and sessions must be installed. The CICS Gateway for Java can use a specific or generic connection.

B.2 Running the CICS Gateway for Java

The CICS Gateway for Java is started from an OpenEdition shell prompt. It can also be started by submitting JCL that runs the OS/390-supplied program BPXBATCH that executes OpenEdition programs and shell scripts that reside in the HFS.

Figure 44 on page 98 shows the JCL that can be used to run the JGate script file. It is assumed that the CICS Gateway for Java is installed in the /u/java/JGate directory.

```
//JGATE    JOB (999,POK),'JGATE',CLASS=A,MSGCLASS=T,
//             NOTIFY=&SYSUID
//BPXJGATE EXEC PGM=BPXBATCH,
//           PARM='SH /u/java/JGate/bin/mvs/JGate
//                     -noinput',
//           REGION=28M
//STDIN  DD PATH='/dev/null',
//          PATHOPTS=(ORDONLY)
//STDOUT DD PATH='/u/java/JGate/jgateo.log',PATHOPTS=(OWRONLY,OCREAT),
//          PATHMODE=SIRWXU
//STDERR DD PATH='/u/java/JGate/jgatee.log',PATHOPTS=(OWRONLY,OCREAT),
//          PATHMODE=SIRWXU
//STDENV DD *
DFHJVSYSTEM_00=SCSCPAA9-ITSO System TS 1.2
DFHJVSYSTEM_01=BRANCH-Main branch server
/*
```

Figure 44. Java Gateway Startup JCL

Appendix C. Creating Signed Java Applets

This appendix describes the steps required to build a signed archive file for Netscape Communicator, HotJava, and Microsoft Internet Explorer. Signing the applet class files and placing them in an archive enables browsers to give additional privileges over unsigned applets.

C.1 The Netscape Tools

Two tools for signing Java code are available from Netscape: JAR Packager and zigbert. JAR Packager is a Java applet that is run within Netscape Communicator and has a graphical user interface. Zigbert is a command line tool, available for Windows NT. Both can be downloaded free of charge from the Netscape developer site. JAR Packager proved slow and unable to handle large numbers of files. In order to create a JAR file that included all the VisualAge support classes and our applet, we had to use zigbert.

Follow these steps to create a signed JAR file using zigbert. The steps assume that zigbert has been downloaded and installed onto your machine, and the directory is in the PATH. The steps are:

1. You must have a certificate installed in your Netscape Communicator that is enabled for object signing. See “Obtaining a Digital Certificate” on page 54 for ways to obtain this certificate. Make a note of the name by which the certificate is referred to when you installed it into the Netscape Communicator database.

2. Place all the files that make up your applet into an empty directory, making sure the directory structure is preserved. If you are using VisualAge for Java, use the export project option and specify the directory; however, do not select the export JAR file option.

   To sign all the files, issue the following command (all on one line):

   ```
   zigbert -d "c:\program files\netscape\users\default" -k "Stephen Longhurst's
   IBM ID" c:\build
   ```

   This assumes that the Netscape certificate database files are in the directory c:\program files\netscape\users\default. This is usually the case, the files are named cert*.db and key*.db. The name of the signing certificate is "Stephen Longhurst's IBM ID." You can check the names of your certificates by using the "Security Info" panel in Netscape Communicator and looking under the *Certificates->Yours* section. The directory where all the files have been expanded to is c:\build.

   You will be prompted for the password that protects the certificate you specify. After the command has completed, a new directory called META-INF is created under the top level (c:\build). This directory contains the manifest file for the JAR archive and the Netscape-specific signing information files.

3. Use the **zip** command that is supplied with zigbert to create the JAR file from the directory. Change to the top-level directory (c:\build) and use the command:

   ```
   zip -r ns4applet.jar .
   ```

A file is created called ns4applet.jar, this is the JAR file that is uploaded to the Web server and is referenced in the archive tag of the HTML page. You can use any name you like for the output file.

C.1.1 Create a JAR file signed for Netscape Communicator.

Figure 45 shows the script file used for NC97. It is quite simple to automate the process. You could also add a step to copy the resulting file to the Web server disk if it has been remotely mounted.

```
REM makens4.bat
REM Script to digitally sign a directory structure of files
REM and then package them up into a JAR file.
REM Use on Windows NT
REM
REM Parameter 1 = Directory to package up
REM Parameter 2 = Name of output file
REM
REM Example Usage : makens4 g:\build ns4applet.jar
REM

SET ZIGBERT_DIR=c:\JARPackager\zigbert
SET CERTDB_DIR="c:\program files\netscape\users\default"
SET CERTIFICATE="Stephen Longhurst's IBM ID"

%ZIGBERT_DIR%\zigbert -d %CERTDB_DIR% -k %CERTIFICATE% %1
cd %1
%ZIGBERT_DIR%\zip -r %2 .
```

Figure 45. JAR file creation

C.2 The Sun Java Development Kit Tools

Sun JDK Version 1.1 includes the *javakey* tool that allows you to create, display and save certificates. It is also used to sign JAR files for use with HotJava. Javakey manages a database of entities. These entities are either identities or signers and the user or administrator can declare certain entities to be trusted.

The Sun Microsystems Web site contains comprehensive instructions on using the *javakey* tool as well as a tutorial on signing applets. The page is entitled "Security and Signed Applets."

For the purposes of this project, actually signing a JAR file for use with HotJava was not necessary. HotJava can be configured to allow unsigned applets privileges beyond the sandbox. We used this mechanism when testing the applet with HotJava. You need to create a separate JAR file for use with HotJava because one signed with Netscape tools will fail to load properly. To create the JAR file, use the *jar* tool in the directory where your class files are expanded.

```
jar -cvf output.jar *
```

C.3 Microsoft Authenticode Technology

You can use Microsoft tools to create digitally signed CAB files, giving your applets access to system resources when running in Microsoft Internet Explorer. You need to have the Microsoft Java Software Development Kit. The SDK provides a tool called *makecert* that allows you to generate a test software publishers certificate to sign the CAB file with. The Java SDK is available from the Microsoft Web site.

The followings steps are detailed on the Microsoft web site in a document entitled "Signing a Cabinet File with Java Privileges Using Signcode." It is assumed that the Java SDK is installed on your machine with the bin directory in the PATH. The steps are:

1. Create a certificate with the makecert program using the following command:

   ```
   makecert -sk DeveloperKey -n "CN=Company Development" TestCert.cer
   ```

2. Use the cert2spc program to turn the certificate into a test software publishers certificate:

   ```
   cert2spc TestCert.cer TestPublish.spc
   ```

3. Extract all your Java applet class files into an empty directory. This step is equivalent to Step 2 in “The Netscape Tools” on page 99. Use the *cabarc* tool to create a CAB file containing all the files:

   ```
   cabarc -r -p -s 6144 n output.cab *
   ```

4. Use the *signcode* program to sign the CAB file with your software publishers certificate (this is all one command).

   ```
   signcode -j javasign.dll -jp low -spc TestPublish.spc -k DeveloperKey
   output.cab
   ```

   The -jp options specifies the security level at which the CAB file is signed. If your code needs to do anything beyond the sandbox other than access the "scratch pad," it must be signed with low security.

5. Create a CAB file signed for Microsoft Internet Explorer

   Figure 46 on page 102 shows the Windows NT script file used to create the NC97 applet signed CAB file. It does not include the steps to create the certificate because this only needs to be done once.

```
REM Script to digitally sign a directory structure of files
REM and then package them up into a CAB file
REM Use on Windows NT
REM
REM Parameter 1 = Directory to package up
REM Parameter 2 = Name of output file
REM
REM Example Usage : makeie g:\build iecashier.cab
REM
SET SDK_DIR=c:\sdk-java.20\bin
SET CERT="g:\ie_certs\itsokey.spc"
SET NAME=ItsoKey
cd %1
%SDK_DIR%\cabarc -r -p -s 6144 n %2 Residency\* Domain\* COM\* ibm\*
%SDK_DIR%\signcode -j javasign.dll -jp low -spc %CERT% -k %NAME% %2
%SDK_DIR%\chkjava %2
```

Figure 46. CAB File Creation

Appendix D. Net.Data Macro

In this appendix, we give the complete environment needed to run the Net.Data macro.

D.1 The Tables

We created a dedicated database, named DWDB, on a different machine (tonga.almaden.ibm.com). The purpose of the database is to hold the history data related to customer orders of foreign currencies. Figure 47 shows the definition for table NCM.Ord_Hist.

```
create table ncm.Ord_Hist
       branch_number char (2),
       branch_order_no int,
       order_date date,
       order_time time,
       completion_date date,
       completion_time time,
       branch_ord_status char (10),
       tran_type char (4))
```

Figure 47. Ord_Hist Table Definition

Figure 48 shows the definition for table NCM.Ord_Detail_Hist.

```
create table ncm.Ord_detail_hist
       branch_number char (2),
       branch_order_no int,
       branch_type_curr dchar(3),
       branch_denom int,
       branch_amount dec (11,2))
```

Figure 48. Ord_Detail_Hist Table Definition

Both tables exactly replicate the columns of the operational tables. We have not undertaken any data remodeling, which would usually be needed for a real-life data warehouse implementation. The tables are meant to keep history—possibly for years—of customer orders, which are unlikely to be maintained in the operational databases. Having a second database is typical of a data warehouse application. An operational database must periodically be subject to file cleaning operations (removal of old data), for performance reasons or because such data is no longer relevant. We also created tables NCM.Branch and NCM.Exchange_rate, which contain information relevant to our sample application.

Finally, we created a view that proves useful for aggregating data in a way that keeps the actual SQL statement in our sample macro very simple (see Figure 49).

```
create view ncm.ord_hist_view
(branch_number, branch_type_curr, yyyymm_order, amount)
as select b.branch_number, b.branch_type_curr,
cast (year (a.order_date) as char (4)) ||
substr (digits (cast (month (a.order_date) as smallint)),4,2),
case
when a.tran_type = 'BUY'  then   (b.branch_amount * b.branch_denom)
when a.tran_type = 'SELL' then - (b.branch_amount * b.branch_denom)
end
from ncm.branch_order a,
ncm.branch_ord_detail b
where a.branch_number = b.branch_number
and a.branch_order_no = b.branch_order_no
and a.branch_ord_status = 'COMPLETE'
```

Figure 49. Ord_Hist_View View Definition

D.2 Net.Data Macro

This is the list of the Net.Data macro used in our migration

```
%{*************************************************************************/
/*   FileName: ncmndb.mac                                                  */
/*                                                                         */
/*   SQL function blocks included in this file are:                        */
/*       sql_A()                                                           */
/*       qry_DB_B()                                                        */
/*       qry_DB_C()                                                        */
/*                                                                         */
/*   Java applet used by this file:                                        */
/*       ChartUI2()                                                        */
/*                                                                         */
/*   HTML blocks included in this file are:                                */
/*       INPUT                                                             */
/*       REPORT                                                            */
/*************************************************************************%}
%{*************************************************************************/
/*   GLOBAL DEFINEs - general                                              */
/*************************************************************************%}
%define {
 DATABASE  = "DWDB"
 TABLEQLFR = "NCM"
 LOGIN     = "CICSRS3"
 PASSWORD  = "CICSRS3"
 DTW_HTML_TABLE    = "yes"
 DTW_SAVE_TABLE_IN = "currtable"
 SHOWSQL   = "NO"
 mainfooter  = {<a href="/cgi-bin/db2www.exe/ncmndb.mac/input">Return to Input
form</a>%}
 sampleTitle = "Currency demand profile on time"
%}
%{*************************************************************************/
/*   Applet Defines for Chart                                              */
```

```
/*************************************************************************%}
%define {
 ChartType = "Vertical Bar"
 ChartUI2.codebase="http://testcase.boulder.ibm.com:8081/java"
 ChartUI2.height = "250"
 ChartUI2.width = "400"
 name = "$(N2)"
 numrow = "$(ROW_NUM)"
%}
%{*************************************************************************/
/*  Function:     qryDB_B           Language Environment: SQL              */
/*  Description:  returns branches in drop down list box                  */
/*************************************************************************%}
%FUNCTION(DTW_SQL) qryDB_B() {
  SELECT * FROM $(TABLEQLFR).BRANCH ORDER BY 1
  %REPORT{
  %IF (RETURN_CODE == "0")
  <select name=BRANCH>
  %ENDIF
  %ROW{
   <option>$(V1)
   $(V2)
   $(V3)
  %}
  %IF (RETURN_CODE == "0")
  </select>
  %ENDIF
 %}
%}
%{*************************************************************************/
/*  Function:    qryDB_C              Language Environment: SQL            */
/*  Description: returns currencies in drop down list box                 */
/*************************************************************************%}
%FUNCTION(DTW_SQL) qryDB_C() {
  SELECT TYPE_OF_CURRENCY FROM $(TABLEQLFR).EXCHANGE_RATE ORDER BY 1
  %REPORT{
  %IF (RETURN_CODE == "0")
  <select name=CURRENCY>
  %ENDIF
  %ROW{
   <option>$(V1)
  %}
  %IF (RETURN_CODE == "0")
  </select>
  %ENDIF
 %}
%}
%{*************************************************************************/
/*  Function:    ChartUI2                  Language Environment: Applet   */
/*  Description: Call to the Applet Language Environment for ChartUI2     */
/*************************************************************************%}
%function (dtw_applet) ChartUI2();
%{*************************************************************************/
/*  Function:  Process               Language Environment: SQL            */
/*  Description: Queries the database and invokes the Java applet         */
/*               within the report section                                */
/*************************************************************************%}
%function(DTW_SQL) Process() {
```

```
select yyyymm_order, sum (amount) from $(TABLEQLFR).ORD_HIST_VIEW
   where branch_number    = SUBSTR('$(BRANCH)',1,2)
   and branch_type_curr   = '$(CURRENCY)'
   and yyyymm_order      >= '$(DATE_FROM)'
   and yyyymm_order      <= '$(DATE_TO)'
 group by yyyymm_order
 order by 1
%REPORT{
   %IF (RETURN_CODE == "0")
    <table border=2 cellspacing=40 cellpadding=0>
    <tr> <th colspan=2 bgcolor='a7a7ff'>Currency: $(CURRENCY) From:
$(DATE_FROM) To: $(DATE_TO)
  Branch: $(BRANCH)
    <tr> <th bgcolor='ffaacc'>Tabular Data <th bgcolor='ffaacc'> Java Applet
    <tr>
    <td align=center>
    <table border=1 >
    <tr><td>Month</td> <td>Requested</td>
  %ENDIF
  %ROW{
   <tr><td>$(V1)</td> <td align=center>$(V2)</td>
  %}
  %IF (RETURN_CODE == "0")
    </table>
    <td>
    @DTWA_ChartUI2(ChartUI2.codebase,ChartUI2.width,
ChartUI2.height,ChartType, numrow, currtable, name)
    </table>
  %ENDIF
 %}
%}
%{*************************************************************************/
/*  HTML section: INPUT                                                   */
/*  Description: Provides input form for user to choose the information   */
/*               for the database query                                   */
/*************************************************************************%}
%HTML_INPUT{
<html>
<head>
<TITLE>$(sampleTitle)</TITLE>
</head>
<body>
<H3>$(sampleTitle)</H3>
<b>
<p>
Inquiry arguments: BRANCH/CURRENCY/PERIOD OF TIME<br>
<ul type=square>
 <li>Select Branch from the drop down list box
 <li>Select Currency from the drop down list box
 <li>Enter start/end time you are interested
</ul>
</b>
<p>
<FORM METHOD="POST"
ACTION="report">
<HR>
Branch:
<P>
```

```
@qryDB_B()
<P>
Currency:
@qryDB_C()
<p>
Date from (YYYYMM):
<INPUT TYPE="text" NAME="DATE_FROM" SIZE=6>
Date to (YYYYMM):
<INPUT TYPE="text" NAME="DATE_TO" SIZE=6>
<p><HR>
<P>
<INPUT TYPE="submit" VALUE="Submit Query">
<INPUT TYPE="reset" VALUE="Reset input">
</FORM>
<P>
</body>
</html>
%}



%{*********************************************************************/
/*  HTML section: REPORT                                              */
/*  Description: Queries the database to create the pie chart via a JAVA  */
/*               applet                                               */
/*********************************************************************%}
%HTML_REPORT{
<html>
<head>
<TITLE>$(sampleTitle)</TITLE>
</head>
<body>
<H3>$(sampleTitle)</H3>
<p>
Following are the results of your query.
@Process()
<P>
<hr>$(mainfooter)
</body>
</html>
%}
```

Appendix E. Special Notices

This publication is intended to help technical professionals to migrate client/server applications to network-computing applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by CICS Transaction Server for OS/390 Version 1.2, Domino Go Webserver Version 4.6, VisualAge for Java Version 1.0, DB2 for MVS/ESA Version 4, and DB2 Universal Database version 5.0. See the PUBLICATIONS section of the IBM Programming Announcement for each of those products for information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
CICS
CICS OS2
CICS/ESA
DB2
IBM
IMS
MVS
System/390

Appendix F. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

F.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 115.

- *From Client/Server to Network Computing, A Migration to Domino,* SG24-5087 (to be published in 1998)
- *Accessing CICS Business Applications from the World Wide Web,* SG24-4547, published by Prentice Hall
- *CICS Transaction Server for OS/390: Version 1 Release 2 Implementation Guide,* SG24-2234
- *VisualAge and Transaction Processing in a Client/Server Environment*, GG24-4487
- *Network Computing Framework Component Guide,* SG24-2119
- *Programming with VisualAge for Java*, Marc Carrel-Billard, published by Prentice Hall, ISBN 0139113711
- *JavaBeans by Example,*SG24-2035, published by Prentice Hall
- *Java Network Security*, SG24-2109, published by Prentice Hall
- *Application Development with VisualAge for Java Enterprise*, SG24-5081

F.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

F.3 Other Publications and Web Sites

These publications and Web sites are also relevant as further information sources:

F.3.1 Network Computing Framework

- Web Sites

- *http://www.software.ibm.com/openblue*, for information about IBM Open Blueprint
- *http://www.software.ibm.com/ebusiness/ncf*, for information about the Network Computing Framework

F.3.2 JAVA

- Publications
  - *Teach Yourself Java 1.1 in 21 Days (2nd Edition)*, Laura Lemay and Charles L. Perkins, Sams.net Publishing, ISBN 1575211424
  - *Java Database Programming with JDBC,* Pratik Patel and Karl Moss, published by The Coriolis Group, ISBN 1-57610-056-1
- Web Sites
  - *http://www.ibm.com/java*, for information about IBM support for Java, including developers tools (*/tools*) and a library of Java applications and applets, and servlets (*/apps*)
  - *http://www.internet.ibm.com/commercepoint/registry*, for information about IBM registry
  - *http://java.sun.com*, for information about Sun support for Java, including beans (*/beans*), security (*/security*), and JDK (*/product/jdk*)
  - *http://www.microsoft.com/java* for information about Microsoft support for Java, including security (*/security*) and SDK (*/SDK*).
  - *http://developer.netscape.com*, for information about developing with Netscape
  - *http://www.verisign.com*, for information about VeriSign, Inc.a provider of digital authentication services
  - *http://www.thawte.com*, for information about Thawte, a provider of security and authentication services.
  - *https://certs.netscape.com*, for the Netscape list of Certificate Authorities that issue digital certificates

F.3.3 Domino Go Web Server

- Publications

  Portable document format (PDF) versions of *Planning for Installation,* the *Webmaster's Guide*, the *Webmaster Programming Guide*, and the *Webmaster search engine documentation* are available from the Domino Go Webserver Web site.

  Hardcopy versions of *Planning for Installation* and the *Webmaster's Guide* can be ordered through Lotus.
- Web Site
  - *http://www.ics.raleigh.ibm.com*, for information about Domino Go Web Server

F.3.4 CICS

- Publications

- *CICS Transaction Server for OS/390 V1R2 Planning for Installation*, GC33-1789
- *CICS Transaction Server for OS/390 V1R2 Release Guide*, GC33-1570
- *CICS Transaction Server for OS/390 V1R2 Migration Guide*, GC33-1571

- Web Sites
  - *http://www.software.ibm.com/ts/cics*, for information about CICS and the CICS Gateway for Java

F.3.5 DB2

- Publications
  - *IBM DB2 Universal Database Administration Getting Started*,S10J-8154
  - *IBM DB2 Universal Database Administration Guide Version 5*, S10J-8157
  - *IBM DB2 Universal Database Messages Reference Version 5*, S10J-8168
  - *IBM DB2 Universal Database Roadmap to DB2 Programming*, S10J-8155
  - *IBM DB2 Universal Database SQL Reference Version 5*, S10J-8165
- Web Sites
  - *http://www.software.ibm.com/data/db2/udb*, for information about DB2 Universal Database version 5
  - *http://www.software.ibm.com/data/net.data*, for information about Net.Data
  - *http://www.software.ibm.com/data/db2/java*, for information about DB2 Java enablement.

F.3.6 VisualAge for Java

- Publications
  - *VisualAge: Concepts and Features*, GG24-3946
- Web Sites
  - *http://www.software.ibm.com/ad/vajava*, for information about VisualAge for Java

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** – to order hardcopies in United States
- **GOPHER link to the Internet** – type `GOPHER WTSCPOK.ITSO.IBM.COM`
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/redbooks`

- **IBM Direct Publications Catalog on the World Wide Web**

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) – send orders to:

	IBMMAIL	**Internet**
In United States	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** – send orders to:

IBM Publications Publications Customer Support P.O. Box 29570 Raleigh, NC 27626-0570 USA	IBM Publications 144-4th Avenue, S.W. Calgary, Alberta T2P 3N5 Canada	IBM Direct Services Sortemosevej 21 DK-3450 Allerød Denmark

- **Fax** – send orders to:

United States (toll free)	1-800-445-9269
Canada	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** – ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** – send note to `softwareshop@vnet.ibm.com`
- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name | Last name

Company

Address

City | Postal code | Country

Telephone number | Telefax number | VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date | Card issued to | Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

A

application programming interface (API). A set of calling conventions defining how a service is invoked through a software package.

applet. A Java program designed to run within a Web browser. Contrast with application.

application. In Java programming, a self-contained, stand-alone Java program that includes main() method. Contrast with applet.

bean. A definition or instance of a JavaBeans component.

browser. An Internet-based tool that lets user browse Web sites.

call level interface (CLI). A callable application program interface (API) for database access, which is an alternative to an embedded SQL application program interface. In contrast to embedded SQL, CLI does not require precompiling or binding by the user, but instead provides a standard set of functions to process SQL statements and related services at run time.

Customer Information Control System (CICS). A distributed online transaction processing system designed to support a network of many terminals. The CICS family of products is available for a variety of platforms ranging from a single workstation to the largest mainframe.

CICS Access Builder. A VisualAge for Java Enterprise tool that generates beans to access CICS transactions through the CICS Gateway for Java and CICS Client.

CICS Client. A server program that processes CICS ECI calls, forwarding transaction requests to a CICS program running on a host.

CICS Gateway for Java. A server program that processes Java ECI calls and forwards CICS ECI calls to the CICS Client.

class. An aggregate that defines properties, operations, and behavior for all instances of that aggregate.

client. As in client/server computing, the application that makes requests to the server and, often, handles the necessary interaction with the user.

client/server. A form of distributed processing, in which the task required to be processed is accomplished by a client portion that requests services and a server portion that fulfills those requests. The client and server remain transparent to each other in terms of location and platform. See *client* and *server*.

commit. The operation that ends a unit of work to make permanent the changes it has made to resources (transaction or data).

Common Gateway Interface (CGI). A standard protocol through which a Web server can execute programs running on the server machine. CGI programs are executed in response to requests from Web client browsers.

Common Object Request Broker Architecture (CORBA). A middleware specification which defines a software bus—the Object Request Broker (ORB)—that provides the infrastructure

communications area (COMMAREA). In a CICS transaction program, a group of records that describes both the format and volume of data used.

conversational. A communication model where two distributed applications exchange information by way of a conversation; typically one application starts (or allocates) the conversation, sends some data, and allows the other application to send some data. Both applications continue in turn until one decides to finish (or deallocate). The conversational model is a synchronous form of communication.

Data Access Builder. A VisualAge for Java Enterprise tool that generates beans to access and manipulate the content of JDBC/ODBC-compliant relational databases.

database. (1) A collection of related data stored together with controlled redundancy according to a scheme to serve one or more applications. (2) All data files stored in the system. (3) A set of data stored together and managed by a database management system.

database management system (DBMS). A computer program that manages data by providing the services of centralized control, data independence, and complex physical structures for efficient access, integrity, recovery, concurrency control, privacy, and security.

DB2 Call Level Interface (CLI). The DB2 call level interface is an alternative SQL interface for the DB2 family of products and takes full advantage of DB2 capability.This implementation closely follows industry standards, such as X/OPEN, to enhance application portability. Currently, the DB2 Call Level Interface functions are compatible with ODBC 2.0, and contain DB2-specific APIs to help exploit DB2 capability.

DB2 for MVS/ESA. An IBM relational database management system for the MVS operating system.

DCE. Distributed Computing Environment. Adopted by the computer industry as a de facto standard for distributed computing. DCE allows computers from a variety of vendors to communicate transparently and

share resources such as computing power, files, printers, and other objects in the network.

distributed processing. Distributed processing is an application or systems model in which function and data can be distributed across multiple computing resources connected on a LAN or WAN. See *client/server computing*.

distributed program link (DPL) enables an application program executing in one CICS system to link (pass control) to a program in a different CICS system. The linked-to program executes and returns a result to the linking program. This process is equivalent to remote procedure calls (RPCs). You can write applications that issue RPCs that can be received by members of the CICS family.

dynamic link library (DLL). A file containing executable code and data bound to a program at run time rather than at link time. The C++ Access Builder generates beans and C++ wrappers that let your Java programs access C++ DLLs.

e-business Either (a) the transaction of business over an electronic medium such as the Internet or (b) a business that uses Internet technologies and network computing in their internal business processes (via intranets), their business relationships (via extranets), and the buying and selling of goods, services, and information (via electronic commerce.)

external call interface (ECI). An API that enables a non-CICS client application to call a CICS program as a subroutine. The client application communicates with the server CICS program using a data area called a *COMMAREA*.

external presentation interface (EPI). An API that allows a non-CICS application program to appear to the CICS system as one or more standard 3270 terminals. The non-CICS application can start CICS transactions and send and receive standard 3270 data streams to those transactions.

Enterprise Access Builders (EAB). In VisualAge for Java Enterprise, a set of code-generation tools.

See also CICS Access Builder and Data Access Builder.

file transfer protocol (FTP). The basic Internet function that enables files to be transferred between computers. You can use it to download files from a remote, host computer, as well as to upload files from your computer to a remote, host computer. See Anonymous FTP.

gateway. A host computer that connects networks that communicate in different languages. For example, a gateway connects a company's LAN to the Internet.

graphical user interface (GUI). A type of interface that enables users to communicate with a program by manipulating graphical features, rather than by entering commands. Typically, a graphical user interface includes a combination of graphics, pointing devices, menu bars and other menus, overlapping windows, and icons.

HotJava A Java-enabled Web and intranet browser developed by Sun Microsystems, Inc. HotJava is written in Java.

hypertext markup language (HTML). The basic language that is used to build hypertext documents on the World Wide Web. It is used in basic, plain ASCII-text documents, but when those documents are interpreted (called *rendering*) by a Web browser such as Netscape, the document can display formatted text, color, a variety of fonts, graphic images, special effects, hypertext jumps to other Internet locations, and information forms.

hypertext transfer protocol (HTTP). The protocol for moving hypertext files across the Internet. Requires an HTTP client program on one end, and an HTTP server program on the other end. HTTP is the most important protocol used in the World Wide Web (WWW). See also Client, Server, WWW.

HTTP request. A transaction initiated by a Web browser and adhering to HTTP. The server usually responds with HTML data, but can send other kinds of objects as well.

hypertext. Text in a document that contains a hidden link to other text. You can click a mouse on a hypertext word and it will take you to the text designated in the link. Hypertext is used in Windows help programs and CD encyclopedias to jump to related references elsewhere within the same document. The wonderful thing about hypertext, however, is its ability to link—using HTTP over the Web—to any Web document in the world, yet still require only a single mouse click to jump clear around the world.

Internet Inter-ORB Protocol (IIOP). An industry standard protocol that defines how General Inter-ORB Protocol (GIOP) messages are exchanged over a TCP/IP network. The IIOP makes it possible to use the Internet itself as a backbone ORB through which other ORBs can bridge.

integrated development environment (IDE). A software program comprising an editor, a compiler, and a debugger. IBM's VisualAge for Java is an example of an IDE.

interface. A set of methods that can be accessed by any class in the class hierarchy. The Interface page in the Workbench lists all interfaces in the workspace.

Internet. The vast collection of interconnected networks that all use the TCP/IP protocols and that evolved from the ARPANET of the late 1960's and early 1970's.

intranet. A private network inside a company or organization that uses the same kinds of software that you would find on the public Internet, but that is only for internal use. As the Internet has become more

popular, many of the tools used on the Internet are being used in private networks. For example, many companies have Web servers that are available only to employees.

Internet protocol (IP). The rules that provide basic Internet functions.

See TCP/IP.

Java. Java is a new programming language invented by Sun Microsystems that is specifically designed for writing programs that can be safely downloaded to your computer through the Internet and immediately run without fear of viruses or other harm to your computer or files. Using small Java programs (called *applets*, Web pages can include functions such as animations, calculators, and other fancy tricks. We can expect to see a huge variety of features added to the Web using Java, since you can write a Java program to do almost anything a regular computer program can do, and then include that Java program in a Web page.

Java archive (JAR). A platform-independent file format that groups many files into one. JAR files are used for compression, reduced download time, and security. Because the JAR format is written in Java, JAR files are fully extensible.

JavaBeans. In JDK 1.1, the specification that defines the platform-neutral component model used to represent parts. Instances of JavaBeans (often called *beans*) may have methods, properties, and events.

Java Database Connectivity (JDBC). In JDK 1.1, the specification that defines an API that enables programs to access databases that comply with this standard.

Java Development Kit (JDK) The Java Development Kit 1.1 is the latest set of Java technologies made available to licensed developers by Sun Microsystems. Each release of the JDK contains the following: the Java Compiler, Java Virtual Machine, Java Class Libraries, Java Applet Viewer, Java Debugger, and other tools.

Java Foundation Classes (JFC) Developed by Netscape, Sun, and IBM, JFCs are building blocks that are helpful in developing interfaces to Java applications. They allow Java applications to interact more completely with the existing operating systems.

LAN. Local area network. A computer network located at a user's establishment within a limited geographical area. A LAN typically consists of one or more server machines providing services to a number of client workstations.

LU type 6.2 (LU 6.2). A type of logical unit used for CICS intersystem communication (ISC). LU 6.2 architecture supports CICS host-to-system-level products and CICS host-to-device-level products. APPC is the protocol boundary of the LU 6.2 architecture.

logical unit of work (LUW). An update that durably transforms a resource from one consistent state to another consistent state. A sequence of processing actions (for example, database changes) that must be completed before any of the individual actions can be regarded as committed. When changes are committed (by successful completion of the LUW and recording of the synch point on the system log), they do not need to be backed out after a subsequent error within the task or region. The end of an LUW is marked in a transaction by a synch point that is issued by either the user program or the CICS server, at the end of task. If there are no user synch points, the entire task is an LUW.

messaging. A communication model whereby the distributed applications communicate by sending messages to each other. A message is typically a short packet of information that does not necessarily require a reply. Messaging implements asynchronous communications

method. A fragment of Java code within a class that can be invoked and passed a set of parameters to perform a specific task.

Multipurpose Internet Mail Extension (MIME). The Internet standard for mail that supports text, images, audio, and video.

online transaction processing (OLTP). A style of computing that supports interactive applications in which requests submitted by terminal users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. An online transaction-processing system supervises the sharing of resources to allow efficient processing of multiple transactions at the same time.

object. (1) A computer representation of something that a user can work with to perform a task. An object can appear as text or an icon. (2) A collection of data and methods that operate on that data, which together represent a logical entity in the system. In object-oriented programming, objects are grouped into classes that share common data definitions and methods. Each object in the class is said to be an instance of the class. (3) An instance of an object class consisting of attributes, a data structure, and operational methods. It can represent a person, place, thing, event, or concept. Each instance has the same properties, attributes, and methods as other instances of the object class, though it has unique values assigned to its attributes.

ODBC Driver. An ODBC driver is a dynamically linked library (DLL) that implements ODBC function calls and interacts with a data source.

ODBC Driver Manager. The ODBC driver manager, provided by Microsoft, is a DLL with an import library. The primary purpose of the Driver Manager is to load ODBC drivers. The Driver Manager also provides entry points to ODBC functions for each driver and

parameter validation and sequence validation for ODBC calls.

Open Database Connectivity (ODBC). A Microsoft-developed C database application programming interface (API) that allows access to database management systems calling callable SQL, which does not require the use of a SQL preprocessor. In addition, ODBC provides an architecture that allows users to add modules called *database drivers* that link the application to their choice of database management systems at run time. This means applications no longer need to be directly linked to the modules of all the database management systems that are supported.

Object Request Broker (ORB). A CORBA term designating the means by which objects transparently make requests and receive responses from objects, whether they are local or remote.

protocol. (1) The set of all messages to which an object will respond. (2) Specification of the structure and meaning (the semantics) of messages that are exchanged between a client and a server. (3) Computer rules that provide uniform specifications so that computer hardware and operating systems can communicate. It's similar to the way that mail, in countries around the world, is addressed in the same basic format so that postal workers know where to find the recipient's address, the sender's return address and the postage stamp. Regardless of the underlying language, the basic protocols remain the same.

proxy. An application gateway from one network to another for a specific network application such as Telnet of FTP, for example, where a firewall's proxy Telnet server performs authentication of the user and then lets the traffic flow through the proxy as if it were not there. Function is performed in the firewall and not in the client workstation, causing more load in the firewall. Compare with socks.

Remote Method Invocation (RMI). In JDK 1.1, the API that allows you to write distributed Java programs, allowing methods of remote Java objects to be accessed from other Java virtual machines.

Remote Procedure Call (RPC). A communication model where requests are made by function calls to distributed procedure elsewhere. The location of the procedures is transparent to the calling application.

sandbox. A restricted environment, provided by the Web browser, in which Java applets run. The sandbox offers them services and prevents them from doing anything naughty, such as doing file I/O or talking to strangers (servers other than the one from which the applet was loaded). The analogy of applets to children led to calling the environment in which they run the *sandbox.*

schema. In the Data Access Builder, the representation of the database that will be mapped. In the Data Access Builder, the mapping contains a set of definitions for all attributes matching all the columns for your database table, view, or SQL statement, information required to generate Java classes.

server. A computer that provides services to multiple users or workstations in a network; for example, a file server, a print server, or a mail server.

Socket Secure (SOCKS). The gateway that allows compliant client code (client code made socket secure) to establish a session with a remote host.

Transmission Control Protocol/Internet Protocol (TCP/IP). The basic programming foundation that carries computer messages around the globe via the Internet. The suite of protocols that defines the Internet. Originally designed for the UNIX operating system, TCP/IP software is now available for every major kind of computer operating system. To be truly on the Internet, your computer must have TCP/IP software.

thin client Thin client usually refers to a system that runs on a resource-constrained machine or that runs a small operating system. Thin clients don't require local system administration, and they execute Java applications delivered over the network.

transaction. A unit of processing (consisting of one or more application programs) initiated by a single request. A transaction can require the initiation of one or more tasks for its execution.

transaction processing. A style of computing that supports interactive applications in which requests submitted by users are processed as soon as they are received. Results are returned to the requester in a relatively short period of time. A transaction processing system supervises the sharing of resources for processing multiple transactions at the same time.

Uniform Resource Locator (URL). Standard to identify resources on the World Wide Web

virtual machine (VM) A software program that executes other computer programs. It allows a physical machine, a computer, to behave as if it were another physical machine.

Web server The server component of the World Wide Web. It is responsible for servicing requests for information from Web browsers. The information can be a file retrieved from the server's local disk or generated by a program called by the server to perform a specific application function.

workstation. A configuration of input/output equipment at which an operator works. A terminal or microcomputer, usually one that is connected to a mainframe or a network, at which a user can perform applications.

World Wide Web (WWW or Web). A graphic hypertextual multimedia Internet service.

List of Abbreviations

API	application program interface
ARM	application request manager
ASCII	American National Standard Code for Information Interchange
AWT	abstract windowing toolkit
CA	certification authority
CAB	cabinet (Microsoft)
CAE	client application enabler
CGI	common gateway interface
CICS	Customer Information Control System
CICS TS	CICS Transaction Server for OS/390
CLI	call level interface
COMMAREA	communication area (CICS)
CORBA	Common Object Request Broker Architecture
DBMS	database management system
DB2	Database 2
DCE	Distributed Computing Environment
DDCS/2	Distributed Database Connection Servives/2
DLL	dynamic link library
DPL	distributed program link (CICS)
DRDA	distributed relational database architecture
EBCDIC	extended binary coded decimal interchange code
ECI	external call interface (CICS)
EPI	external presentation interface (CICS)
ESA	Enterprise Systems Architecture
EXCI	external CICS interface
FTP	file transfer protocol
GUI	graphical user interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	integrated development environment
IIOP	Internet inter-ORB protocol
IMS	Information Management System
IS	information system
ITSO	International Technical Support Organization
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java development kit
JFC	Java foundation classes
JVM	Java virtual machine
LAN	local area network
LDAP	lightweight directory access protocol
LUW	logical unit of work
MIME	multipurpose Internet mail extensions
MVS	Multiple Virtual Storage
NC	network computer
NCF	network computing framework
NetBIOS	Network Basic Input/Output System
NNTP	NetNews transfer protocol
NT	Microsoft Windows NT (new technology)
ODBC	open database connectivity
OLTP	online transaction processing
ORB	object request broker
OS/2	Operating System/2
OSF	Open Software Foundation
PC	personal computer
POP	Post Office Protocol
RACF	Resource Access Control Facility
RAD	rapid application development
RMI	remote method invocation
SDK	software developer's kit
SET	secure electronic transaction

SHTTP	secure hypertext transport protocol
SIT	system initialization table
SMTP	simple mail transfer protocol
SNA	Systems Network Architecture
SNMP	simple network management protocol
SQL	structured query language
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TSO	time sharing option
URL	uniform resource locator
XA	extended architecture

Index

Numerics

A

B

C

D

E

F

N

O

P

R

S

T

U

V

W

X

Y

Z

ITSO Redbook Evaluation

From Client/Server to Network Computing A Migration to Java
SG24-2247-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-2247-00
Printed in the U.S.A.