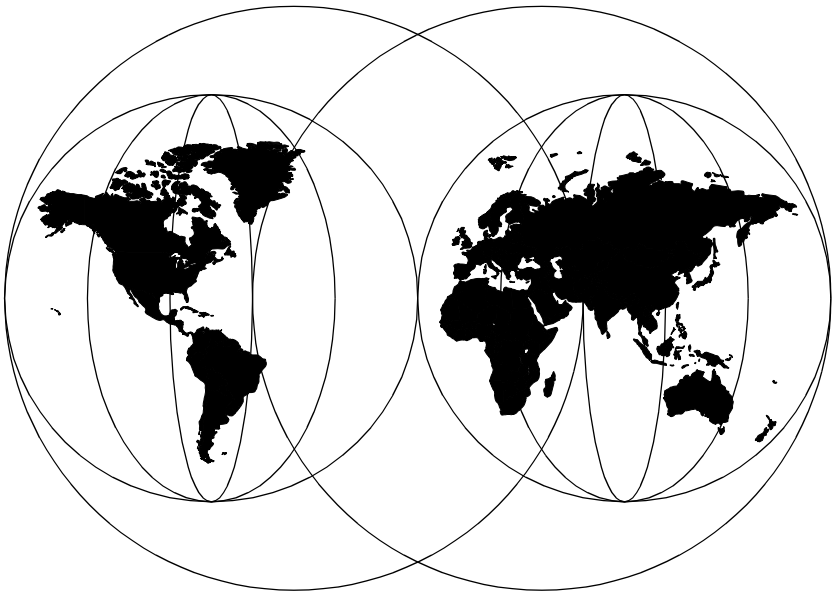
IBM

Designing Java Applications for Network Computers

Oscar Cepeda, Jens Andexer, Craig Grossi, Timothy Luke



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5111-00

SG24-5111-00

International Technical Support Organization

Designing Java Applications for Network Computers

August 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 141.

First Edition (August 1998)

This edition does not apply to any specific operating system or application other than those described within this document.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. DHHB Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

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.

© Copyright International Business Machines Corporation 1998. All rights reserved
Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1998

Figures

© Copyright IBM Corp. 1998

Tables

© Copyright IBM Corp. 1998

Preface

Although the term thin client is relatively new, Network Computers in various forms have been in the marketplace for years now, along side traditional Intel-based PCs. However, the Java application designer and developer will quickly discover that writing Java applications for Network Computers introduces certain challenges and issues that must be addressed in order to maintain a functional, robust and user-friendly application environment.

This redbook is intended for the reader who is comfortable with concepts and implementation of client/server applications using Intel-based clients. With this base, this redbook describes many of the issues related to the design and implementation of Java applications on Network Computers. In addition, we describe the Network Computing Framework (NCF) and its relationship with Network Computers, as well as basic software maintenance issues in the context of thin clients.

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 Austin Center.

Oscar Cepeda is an Advisory Software Engineer at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide in areas including OS/2, Warp Server and WorkSpace On-Demand. Before joining the ITSO in 1995, Oscar worked in IBM US Availability Services as an I/T Specialist.

Jens Andexer graduated with a degree in Business Administration in 1985 and since that time has been working in Canada, England, Scotland, Germany, Australia and the US in capacities ranging from Programmer to Project Leader. He joined IBM in January 1998. Jens is concerned about issues surrounding the maintainability of software and is completing his Master's in Computer Science, specializing in design for software maintainability in the procedural and OO paradigms.

Craig Grossi is a Senior Consultant in the NC Java Services Practice of IBM Global Services. He has 8 years of experience in the Object Technology field. He has worked for IBM for the past 4 years as an OO Mentor and Team Lead for Smalltalk and Java projects. He holds a B. S. degree in Electrical Engineering from Tufts University. His areas of expertise include OT Methodology, Architecture and System Design.

© Copyright IBM Corp. 1998

Timothy Luke is a Senior Consultant in the NC Java Services Practice of IBM Global Services. Tim has 17 years of experience in manufacturing automation and process control. He has worked at IBM for 3 years. His areas of expertise include object-oriented analysis, design and development. Tim has a Master's degree in Computer Science and a Master's degree in Music.

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

Peter Bahrs
IBM Network Computing Software Division (NCSD), Austin

Barry Feigenbaum
IBM NCSD, Austin

Steve Heracleous
IBM Network Computing Division (NCD), Austin

Gary Huber
IBM NCD, Austin

David Kaminsky
IBM Pervasive Computing Division, Raleigh

Kris Lichter
IBM NCSD, Cupertino, CA

Tim Thatcher
IBM NCSD, Austin

Scott Vetter
IBM International Technical Support Organization, Austin Center

Laura Werner
IBM Center for Java Technology, Cupertino, CA

Jason Woodard
IBM NCSD, Armonk, NY

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 157 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@us.ibm.com`

Chapter 1. Network Computing and Java Environment

If you enter a room filled with IS professionals and ask how many are familiar with the term Network Computing, nearly everyone would raise their hand. However, if you ask each person to define Network Computing, no two definitions would be the same. In today's environment, the adoption of Network Computing is increasing significantly. Because of the variety of hardware and software implementations, it means something different to everyone.

If you ask this same group about their interest in Java, again, you would see many hands raised. While not all of those interested are actually implementing production Java applications, interest (and implementation) in Java has skyrocketed over the past few years.

Network Computing and Java are increasingly being used to solve the problem of providing the latest information to end users and clients in a manner that minimizes the administrative costs associated with supporting this infrastructure. As software vendors and I/S departments begin to adopt network computing and Java applications, they invariably make comparisons between programming for their existing Intel-based client/server platforms, and the network computing paradigm.

This chapter gives the reader an understanding of how Network Computers work by using the IBM Network Station as an example. We also describe the major components of the Java programming language. These topics are the basis for further discussion in the chapters that follow.

If you are not comfortable or familiar with some of the fundamental concepts of networking, please read Appendix A, "Network Basics" on page 127, before continuing with this chapter. This Appendix describes some of the networking components and terms used throughout this redbook.

1.1 Network Computing

The client/server paradigm provided many benefits over the host-based, centralized paradigm, including increased computing power at the desktop, a graphical user interface, and local data storage. However, these benefits came at a cost. That is, the overall cost of ownership of computers increased significantly compared to a centralized, mainframe/terminal environment. One solution to this increasing cost is the Network Computer. After a basic definition of the network computer, the operation of a network computer, the IBM Network Station, is used as an example.

© Copyright IBM Corp. 1998

1.1.1 Network Computer

The Network Computer (NC) is a general end-user device designed at a lower cost point than the average Personal Computer (PC). Its main characteristic is that it gets nearly all its resources from other systems on the network, including its operating system, applications and application data, because there is no local permanent storage, such as a hard disk. In contrast, a PC has a hard disk with an operating system installed on it, such as OS/2, Windows 95, WIndows NT or DOS. A PC may also have applications and user data stored on the local hard disk. Since the hard disk can also be used as a paging device for the operating system, the PC can allow for a virtual memory-based operating system. Because a PC can have all these traits, it is sometimes called a fat client.

Although NCs are much newer than PCs, there are many hardware vendor implementations of NCs. They are often called *thin clients* and usually have the following characteristics:

- Display device
- Text input device, such as a keyboard
- System unit with processor (RISC, or Intel)
- Connection to a TCP/IP-based network
- No permanent local storage, such as a hard disk

The key motivation in network computing is to provide cost effective user platforms by accessing resources across the network. The trend is to place less resources on the user platform, centralizing them instead on servers for all NCs to use. This makes more effective use of the common resources and can reduce the overall cost of ownership of end user platforms. As an example of how a network computer works, we describe the IBM Network Station in some detail in the following sections.

1.2 A Network Computer—The IBM Network Station

This section gives a simple overview of the IBM Network Station and its software and hardware architecture.

1.3 IBM Network Station Hardware

The IBM Network Station is a thin client end-user hardware platform that consists of a keyboard, a mouse, a CPU, RAM, and a network adapter. The

network adapter on each Network Station has a unique hard-coded media access control (MAC), or burnt-in, address.

General Network Station specifications are listed in Table 1 on page 3, while a more detailed description can be found in the redbook titled *RS/6000 - IBM Network Station Companion Guide*, SG24-2016.

Table 1. IBM Network Station Hardware Specifications

Model	CPU	RAM	Network Connection
Series 100 Model 100	PowerPC	8 MB	Ethernet
Series 100 Model 200	PowerPC	8 MB	token ring
Series 300 Model 110	PowerPC	16 MB	Ethernet
Series 300 Model 210	PowerPC	16 MB	token ring
Series 1000 Model A52/53	Power PC 603 at 200 MHz	32/64 MB	Ethernet
Series 1000 Model A22/23	Power PC 603 at 200 MHz	32/64 MB	token ring

1.3.1 IBM Network Station Server Software

The server that provides boot and configuration support for the IBM Network Station runs an application called Network Station Manager (NSM). NSM can run on several operating system platforms, including AIX, Windows NT, OS/400, and OS/390. Client access to other resources on the server requires additional server software. This takes the form of communications programs, such as Trivial File Transfer Protocol (TFTP) and Network File System (NFS) that allow clients to make requests for resources on the server.

1.3.2 How the IBM Network Station Works

When an IBM Network Station is switched on, it has no software. All it has available are its local hardware resources. So how does the Network Station go from a hardware device with no operating system to a useful end user system?

The answer is all in the software, or more precisely, the firmware. When the IBM Network Station is switched on, it actually does have some software. Burned into one of its chips is a startup program called the boot monitor, that allows the Network Station to start itself up. The following sections describe how the Network Station starts up.

1.3.2.1 Network Station Power-On

Powering up the IBM Network Station causes the following basic operations to be carried out.

1. POST: The power-on initiates a hardware self-test, the power-on system test (POST), that ensures the hardware components in the Network Station are functioning properly.
2. Obtain Operating System: After the POST sequence, the Network Station must get a copy of the operating system. To obtain this, the Network Station accomplishes the following:
    1. Find Boot Server: Before the Network Station can request a copy of its kernel, it must know where to find a copy of one. To find its kernel, the boot monitor sends a message out on the network to which boot servers reply. The first boot server to reply to the boot request becomes the boot server for the Network Station.
    2. Request Kernel: Once the Network Station has established communications with a boot server, the Network Station sends a request for its kernel.
    3. Download Kernel: When the kernel request is received, the boot server initiates the download of the Network Station's operating system kernel. The Network Station receives the kernel and once the transaction is complete, the boot monitor passes control to the kernel.

1.3.2.2 Network Station Operation

Once the kernel is in control of the Network Station, it configures the Network Station's background color, mouse speed and other basic characteristics. This is done by reading the configuration files that are on the boot server and following their instructions.

Next, after the environment for the Network Station is configured, the initial applications are loaded. The actual applications loaded are determined by another configuration file, which is set up by the administrator. After this is accomplished, the Network Station is ready for end user operations.

1.4 Servers

Servers are systems that allow access to resources in their control. To be a resource provider, a server requires an operating system and application software that enables resource sharing, such as OS/2, AIX, or Windows NT. Servers can be placed into three broad categories, where a single server could belong to more than one category:

Boot Server Provides the kernel and remainder of the operating system

Internet Server Provides access to Internet-based resources

Application Server Provides access to shared applications for users

1.4.1 Boot Server

A boot server has all the software needed to load and start a network computer. It also has the facilities to make it possible for this software to be downloaded to the network computer on demand. The rules that allow the boot server and the client to communicate and get the client stared up are in protocols, but the protocols are also embodied in programs. Therefore, the following are protocols implemented within programs:

DHCP The Dynamic Host Configuration Protocol (DHCP) is a boot protocol that supports the dynamic allocation of IP addresses by the server. In this protocol, the client broadcasts that it needs a boot server and establishes communications with the first boot server to respond. The kernel is downloaded and its execution is initiated. DHCP is a superset, that is, it is based on, BOOTP. More details of this process can be found in the redbook titled *Beyond DHCP: Work your TCP/IP Internetwork with Dynamic IP*, SG24-5280.

BOOTP The bootstrap protocol (BOOTP) is a boot protocol used to download and initiate the execution of an operating system on the BOOTP client, in this case, the IBM Network Station. The client requests its kernel from a specific server and that server assigns an IP address based on the client's MAC, or hard-coded, address.

These two boot protocols can set up communications, but they also need to move files from the server to the client. For the purpose of moving files, therefore, they rely on the following two file transfer protocols:

TFTP The Trivial File Transfer Protocol (TFTP) is a simple protocol that allows files to be transferred between two systems. There is no

security authentication in TFTP, but its simplicity allows for support on a variety of systems.

NFS The Network File System is a more robust implementation of a file sharing mechanism compared to TFTP, with user authentication and performance improvements. From a client view, the files that actually reside on the NFS server appear to be local to the client.

1.4.2 Internet Server

Internet servers are a group of servers that provide resources to clients using the TCP/IP protocols. Examples of Internet servers are these two subcategories:

HTTP server A Hyper Text Transfer Protocol (HTTP) server, or a Web Server, responds to client requests for files with rich text formatting tags using the HTTP protocol. If an application is run using a Web browser, HTTP provides for the download of the software and the data from the server to the client. Examples of HTTP servers are the Lotus Domino Go Webserver and the shareware Apache Server.

FTP server A File Transfer Protocol (FTP) server satisfies request for files from FTP clients. It provides a basic file transfer mechanism, independent of the content of the file.

1.4.3 Application Server

Application servers provide application programs to requesting clients. In addition to the application software, they need nothing other than the programs described above to provide these services. How the applications programs are identified and how they are transferred from the application server to the client are topics dealt with in later sections.

1.5 Networks

A computer network consists of two or more computers connected in a way that allows them to communicate. The computers that communicate across a network have been described as clients and as servers. But, facilitating communications also requires an agreement on how the interchange will take place. This agreement is called a protocol.

1.5.1 Network Function

To allow two computers to communicate requires a set of rules called protocols. A communications protocol is a set of rules that describe how

communications will take place across a network. But, because the task of interacting with another computer is complex, a single protocol cannot do the whole job.

In order to control the complexity associated with communications, the problem must be broken down. Just breaking the problem down is not enough, though. The pieces must be organized in a way that distances the program that needs communication from the implementation of the communications medium.

Hiding the implementation of something is called abstraction. An abstraction hides the details of how a problem is solved and provides a set of externally visible services to the user of the abstraction. The problem of communications is solved this way. Communications protocols are layered, and each layer corresponds to a part of the overall communications problem. How protocols are layered is describe in more detail in Appendix B, “Protocol Layers” on page 131.

1.6 Java—A Good Language for Network Computing

Now that we have a better understanding of the Network Computer, and of basic function of the IBM Network Station in particular, we change our focus to discuss the Java programming language.

Java is a programming language designed to run in both a stand-alone and a network environment. Since a network is an interconnection of many different hardware and software platforms, Java must be able to run on all these platforms.

Allowing Java applications to run on different hardware and software platforms requires two key features. They are described below.

First, the Java application is insulated from the specific hardware and software platform upon which it is running by a standard interface, a component of the *Java Virtual Machine* (JVM). A JVM is a virtual environment inside which Java applications run. A JVM can be written for each unique operating system platform. Because each JVM presents the same, standard interface to a Java application, the same Java code itself can run unchanged regardless of the hardware platform. All it needs is a compatible JVM implemented on that platform.

Second, the translation of source code into a running application has been split into a two-step process. First, the Java source code is translated into a device independent language called Java *bytecode*. In the second step, the

bytecode are interpreted on the target platform by the JVM to perform the specified behavior. Since the bytecodes are standard, they should behave the same on every platform that has a compatible JVM (which is part of the Java Development Kit, see below).

What makes Java so versatile is its platform independence. But what makes Java platform independent is the encapsulation of its interfaces and the standardization of the bytecodes it interprets.

The Virtual Machine

The term virtual machine (VM) was coined in 1959 by IBM to describe the VM operating system that provided a software environment that was independent of the underlying hardware.

1.6.1 Java Development Kit

The Java Development Kit (JDK)provides resources in the form of files and tools for both the development and the Java Run-Time Environments. When a Java program is compiled, the compiler generates bytecodes for the source code files specified during compilation. If the source code relies on classes that are part of the JDK, these are assumed to be on the target platform. Therefore, it is important that the JDK on the platform where the program was compiled be compatible with the JDK on the platform where the program is to be run.

For the purposes of this chapter, knowing that the JDK provides resources to both the development and the run-time environments is sufficient. Below is a short description of the parts of the JDK that are relevant to this chapter. Section 5.2.1 on page 72 describes the contents of the JDK in slightly more detail.

- **Files:**
  - **Java core classes:** These are the compiled (bytecode) classes that constitute the code classes the JDK provides.
  - **Java source files:** These are files that supply the source code for the classes in the Java core class files.
- Java Tools:
  - javac: Java compiler - Translates Java source into bytecodes.
  - java: Invokes the JVM and runs the application code

1.6.2 Java Development Environment

The Java development environment consists of the Java Development Kit and a collection of programs that allow Java programs to be written more effectively.

Part of the development environment is specialized software that allows programers to develop programs more effectively, such as IBM Visual Age for Java. There are many fine books and descriptions of these tools, so they will not be discussed here.

1.6.2.1 Java Compiler

The translation of source code text (.java files) into files containing device independent language, called bytecodes (.class files) is done by the Java compiler that is part of the Java Development Kit (JDK). Although the bytecode is independent of the hardware and software platforms on which it runs, it is dependent of the release of the JDK. See Figure 1 on page 9 for a representation of the relationship between the files.

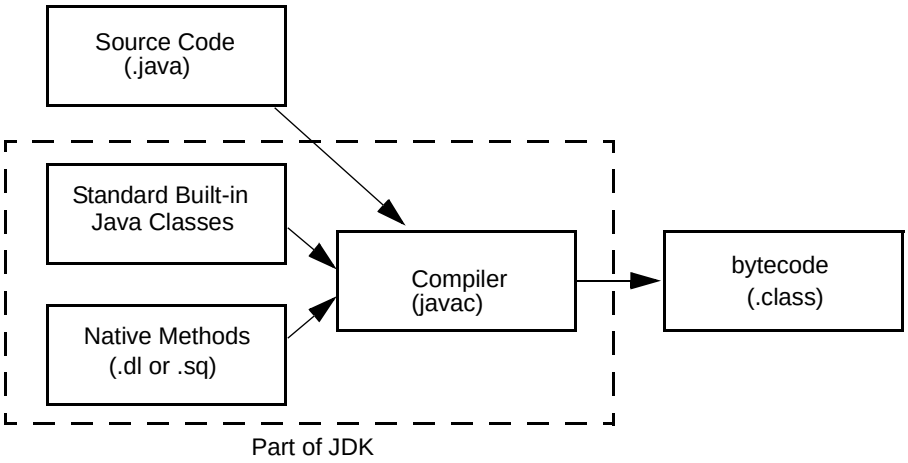


Figure 1. Java Compile Time Environment

1.6.3 Java Run-Time Environment

The Java Run-Time Environment (JRE) provides a run-time environment inside which the Java Virtual Machine can function and make requests for resources to the operating system. The JRE consists of the Java Virtual Machine, the system memory available to the JVM, the files in the JDK that support the JVM and the applet and application class files.

1.6.3.1 Java Virtual Machine

The JVM provides an encapsulated environment inside which the bytecodes can be interpreted. The JVM consists of an execution engine and services that support the execution engine. The components of the JVM can be seen in Figure 2 on page 10 and are described as follows.

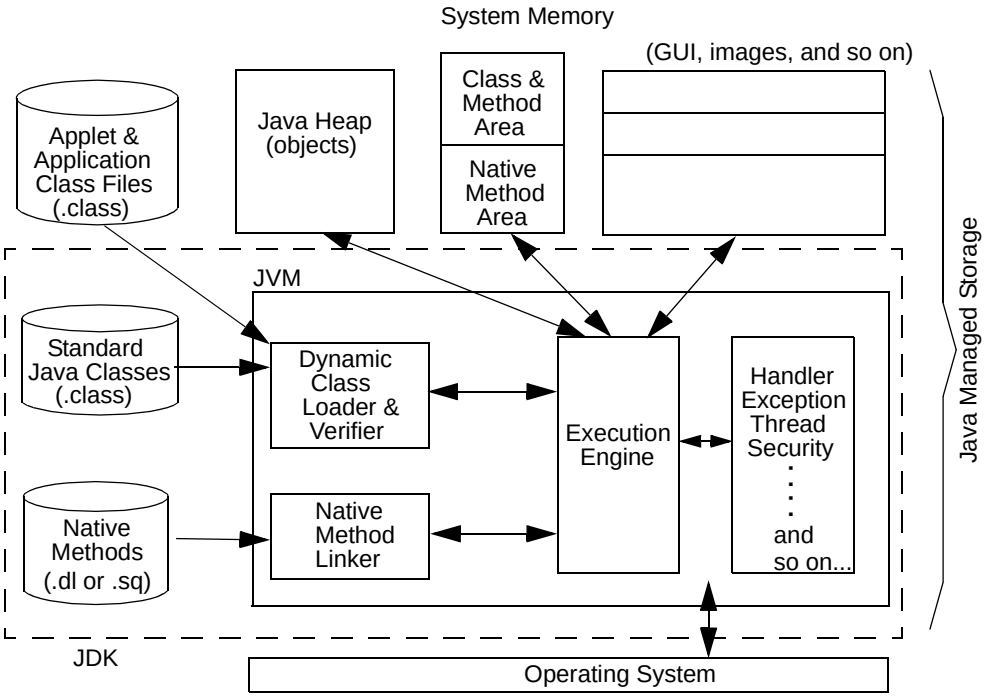


Figure 2. Java Run-Time Environment

Execution Engine The Execution Engine is a piece of software, called an interpreter, that carries out the bytecode instructions contained in Java class files.

Memory Manager When objects or arrays are created in Java they are allocated from free memory in the Java heap. The heap is allocated at start-up time and can be allowed to default or can be specified in parameters to the java command.When the allocated entities are no longer needed, the garbage collection component returns the storage to the unused portion of the *heap*. The total size of the Java heap for an application must be large

enough to provide space for the working set of objects needed by the application. When the limit of available space is approached, the JVM initiates processing that will free any unused space. This is intended to have a minimal effect on the performance of the host, since garbage collection usually runs as a low-priority background thread.

Native Method Support In Native Methods, the body of the method is not supplied as Java bytecodes in a class file. Instead, the method is written in some other language and compiled into native machine code and stored in a separate Dynamic Link Library (DLL) in Windows, or shared library, in UNIX. These methods cannot be included in the execution in the same way as bytecodes can and must be linked during execution time. The native methods are the platform-dependent part of Java. Java uses these to interface with the operating system or with other platform-dependent resources.

Class Loader Since Java programs are broken into classes, an important aspect of the JRE is to load, link and initialize classes. The class loader must, therefore, locate a class locally or across a network and make it available to the Execution Engine.

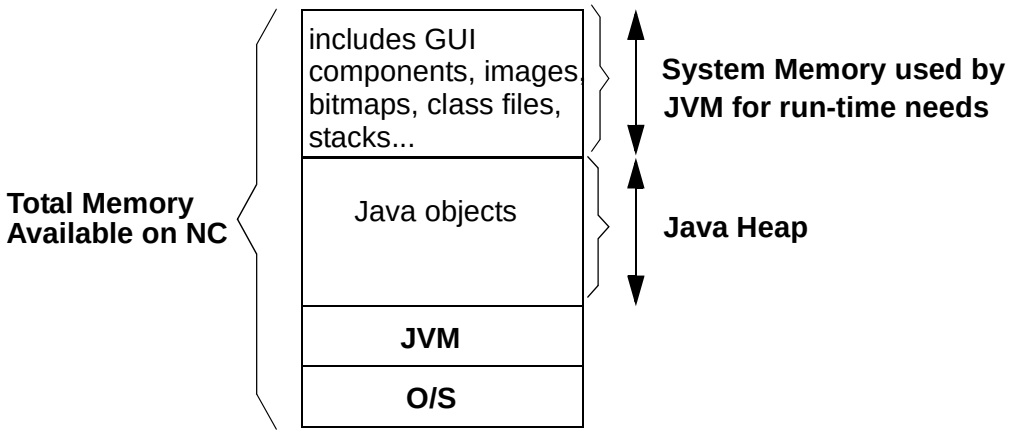
Error and Exception Manager

Exceptions are the mechanisms Java uses to signal that something out of the ordinary has happened. Each method defines an exception handler table that specifies which exception the method is able to catch. The JRE includes an exception manager that is responsible for processing the exceptions each method catches.

Thread Interface Since Java is a multi-threaded language, it must support multiple threads running at the same time. Each thread must execute as if in its own execution engine with its own call stack and local data state.

Security Manager

The Java Security Manager makes it more difficult for hostile programs to do mischief. The Security Manager is an outer shell program that protects the JVM by limiting the bounds that a Java program must reside within and limits the program's access to resources that might be misused.



Note: The actual run-time memory layout and size of data areas are decided by the JVM implementation.

Figure 3. Java Storage Management

1.7 Java Program Types

An executable entity is a program that is in a form that can be invoked in the run-time environment. A context is the run-time environment in which the executable entity runs. In Java, when the executable entity is in control of its own context, it is referred to as an application. When a program requires a context to be provided within which it can execute, then this program is referred to as an applet.

1.7.1 Java Applications

Java applications are stand-alone programs that run independently within a Java Virtual Machine (the JVM is described in Section 1.6.3.1 on page 10). When starting a stand-alone Java application, the operating system passes control directly to the JVM, which loads the program and starts its execution. Although requests are made to the operating system, control is not relinquished by the application to the operating system until the application terminates, which also stops the JVM.

Since control is passed directly from the JVM to the application, control remains there until termination. Only one application can run in one JVM at any one time. There can, however, be multiple JVMs running on a single computer if the platform Java implementation allows it. The Network Station Java implementation, however, does not allow for multiple JVMs.

The mechanism that identifies a Java program as a stand alone application is the inclusion of a routine called main(). This statement identifies the entry point for the application and tells the Java compiler to create a program start point which tells the JVM that the program can be invoked directly.

Java applications are always referred to as trusted code. This means that they have all facilities the JVM can provide available to them.

1.7.2 Java Applets

An applet is not a stand-alone Java program and does not take control of the context. An applet can only be run from “within” another piece of software like a browser or applet viewer.

The JVM receives control from the browser when it interprets the applet’s entry point in the HTML file. It then invokes the JVM, which passes control to the applet. The applet then simply responds and control is passed back to the JVM and back to the browser, which then retains control. As a result, any number of applets can run within the same JVM, or, indeed, within the same application at the same time.

Java applets are not trusted code. When an applet is loaded over the network it is considered untrusted, and its access to resources the JVM can provide is restricted. These restrictions are:

- Untrusted code cannot access the local file system.
- Untrusted code cannot perform networking operations.
- Untrusted code cannot use System.exit() or RunTime.exit() to exit the JVM.

- Untrusted code cannot spawn new processes.
- Untrusted code has restricted access to system properties.
- Untrusted code cannot create or access a thread outside the thread group in which it is running.
- Untrusted code has restriction in the use of classes it can load and define.
- Untrusted code has restrictions on the use of the security package.

1.7.3 Servlets

In the same way that an applet is associated with a client system, a servlet is associated with a Java-enabled Web Application server. A servlet is a server-based Java module that helps the server receive requests from clients, process the request, and return information back to the client, usually a browser. End users usually don't see servlets, but their applets often interact with servlets, depending on how the application they are executing was written.

Servlets can often handle multiple client requests simultaneously. In processing information, servlets can communicate with other servlets on the same system, or with servlets on other servers as well.

1.7.4 Basic Program Constructs

This section briefly describes Java classes, Java objects and communications between objects.

1.7.4.1 Java Classes

A class is the specification for an object. A class specifies an object's domain, all possible data states, and the behavior needed to change from one state to another. The domain is specified in the data definition and the behavior is specified in the methods.

1.7.4.2 Java Objects

An object is the instantiation of a class. When an object is created in Java, the class specification is referenced for the instructions on how to create the object.

When creating an object, the JVM first creates objects for any classes the new object inherits from. To create the new object, the JVM allocates sufficient space in the heap for the object's data and the control characteristics. The control portion of the allocated space is populated with the entry points of the object's methods. Finally, control is passed to the

appropriate constructor method. When the constructor method completes, control is passed back to where the creation of the object was initiated.

1.7.4.3 Communications between Objects

Objects need to communicate. In Java, this is easiest when both objects are on the same computer. In this case, a method in one class can simply call a method in another class. But what if the objects are on different computers?

An object can have both a state and behavior. To change the state, control must be passed to one of the objects methods. Invoking an object's methods is also called messaging.

As with all messages, there must be information and an address to where the information is to be sent. When objects communicate, the information is the data passed as parameters to the method. The message's address is the fully qualified name of the method to be invoked.

The fully qualified name, however, introduces the issue of scope. Scope is the part of a program within which a name can be seen. In inter-object communication, this name must be unique. But more than one method can have the same name. Therefore, the fully qualified name must identify the name of the method that is to be invoked uniquely within the method's scope.

Beyond identifying the object with which communications are to take place, the physical location of the object within a network environment is an important issue that will be dealt with later in this book.

1.7.5 Java in the Network Environment

This section introduces some details that must be considered when implementing Java in a Network environment.

1.7.5.1 Class Creation

In order to create an object, the JVM needs the class to specify how the object is to be created and to supply the bytecode for each methods implementation. Making this available in a thin client environment means importing the class from the server, since there is no persistent local storage. (See Figure 4 on page 16).

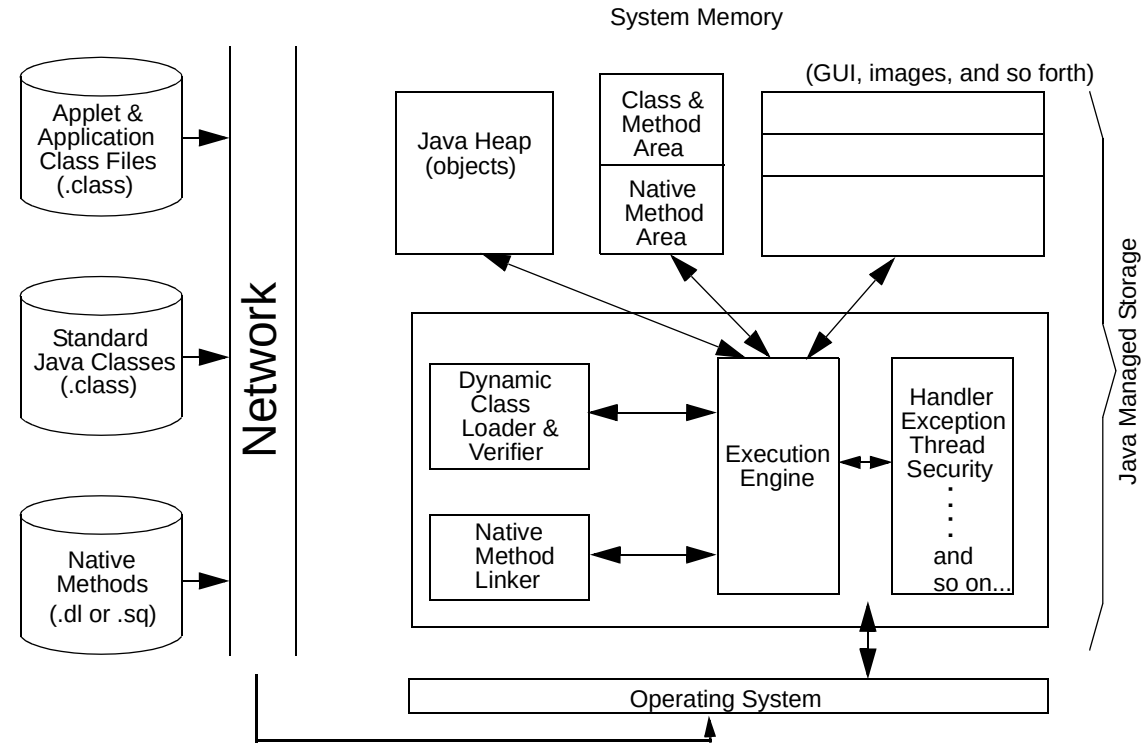


Figure 4. Java in the NC Environment

1.7.5.2 Storage Management

The network computer has a hard limit to the amount of RAM storage that is available since there is no caching. Therefore, if the available space is always near its limit, there will be a persistent overhead associated with freeing heap storage due to garbage collection.

1.7.5.3 Communication

In a network environment, objects that are part of the same Java program can be spread across different physical devices. Therefore inter-object message passing in a network environment means the object to which a message is being directed, in fact, the object whose method is being invoked, may be on a different computer. This means the message must be passed over the network in order to invoke the remote method.

Network Station Operating Environment

On the Network Station, the RAM in which the kernel is already executing is the only resource available locally. It is not possible for more than one Java Virtual Machine (JVM) to be executing locally in the Network Station at a time; however, this is dependent on the operating system and JVM implementation. It is also possible to run an applet (which is dependent on a browser) concurrently with a Java application (which runs independently of a browser). For example, the Navio browser can share the JVM with other applets.

Programs that are executing locally on the Network Station have only the local RAM available as a resource. Therefore, any data or programs must be brought across the network before execution can begin.

1.7.5.4 The Future of the IBM Network Station

The IBM Network Station provides a low cost of ownership and excellent flexibility as a client platform. IBM intends to extend the domain of the Network Station into the Intel space specifically for Network Computing. The IBM announcement below is from
`http://www.pc.ibm.com/networkstation/news/intelproc.html`

IBM Network Computers based on Intel Processors

AUSTIN, April 23, 1998... “IBM* today announced that it is tuning the JavaOS(TM) for Business(TM) product for network computers (NCs) based on Intel processors with assistance from Intel*. IBM and Intel are also working together so that the JavaOS for Business product supports Intel's Lean Client Guidelines. By bringing together the Intel Architecture with the JavaOS for Business software, IBM and Intel are cooperating to expand the choices available to customers seeking network computing solutions.

The JavaOS for Business software will support both the Network Computer Reference Platform and the Intel Architecture Lean Client Guideline. The initial release will support Intel processors and will be available to OEMs mid year.

In addition, IBM will develop a version of the IBM Network Station network computer utilizing an Intel microprocessor, for the high end of the Network Station family. The Intel-based NC from IBM is expected to be available in the first half of 1999, and will take advantage of the JavaOS for Business product. IBM's Network Station will be compliant with both the Intel Architecture Lean Client Guideline and the Network Computer Reference Platform.”

1.8 Network Computing using WorkSpace On-Demand

The IBM Network Station is not the only platform to boast the benefits of Network Computing. Companies with a large investment in OS/2 Warp Server and Intel-based clients were given a a new Network Computing solution in October 1997, when IBM announced WorkSpace On-Demand, an program product that uses OS/2 Warp Server as its foundation.

It is important to note that, as of the writing of this redbook, WorkSpace On-Demand does not directly support the Network Station as a client. This is a complementary solution for companies with Intel-based client systems.

Intel-based desktop and server systems have become pervasive in today's world. Some estimates indicate that there are more than 150 million systems based on Intel processors in the marketplace. In most large corporations, an Intel-based system tends to be a *fat client*. That is, they are computers with local, persistent storage, such as a hard disk drive. The operating system, such as OS/2, Windows 95, or Windows NT, is often stored locally. User applications might also be stored locally as well. Although this allows some degree of flexibility and independence from problems with the network, this fat client configuration can result in increased administrative and support costs, especially for a large corporation with thousands of PCs in their departments.

Companies are looking for ways to control the costs associated with the management of operating systems, data and application software stored on local hard disks of end-user systems. One solution is an IBM software solution, released in 1997, called WorkSpace On-Demand.

WorkSpace On-Demand is a network-centric operating system which allows the client to be an Intel-based PC, yet have characteristics like RISC-based network computers (such as the IBM Network Station). For example, a WorkSpace client gets its operating system, applications and data from servers on the network. This can result in great cost savings because many clients can use the same operating system and application image that resides on a particular server. Saving data and updating operating systems and applications become much easier because there only one image to update, instead of updating each system individually, as in a fat client environment, which can take weeks if you have thousands of clients to update. Another benefit is that customers can run their existing applications on this enhanced platform.

In addition to a centralized source for operating systems and applications, WorkSpace On-Demand allows the administrator to restrict end-user

functions such as desktop bitmaps, colors, and color selection. The tendency for users to engage in unneeded tailoring and configuration is decreased, and the consistency of the user interface is increased.

WorkSpace On-Demand allows companies to continue using their DOS, Windows 3.x, OS/2 and Java applications in a structured environment, which can save money through reduced support costs. For additional information about WorkSpace On-Demand, refer to the redbook titled *WorkSpace On-Demand Handbook*, SG24-2028.

Chapter 2. Network Computing Framework

So now that you have a better understanding of the NC, what solution are you trying to build with it? The application of the Network Computer and the Java language to your solution can be much better understood by stepping back and taking a look at the big picture of Internet and Intranet business applications: e-business. This powerful combination is the integration of the easy access and reach of the Web with the strength, reliability and security of your existing IT. To address the complex issues of developing e-business applications, in April of 1997, IBM and Lotus announced IBM's Network Computer Framework (NCF). This chapter highlights many of the powerful ideas, detailed in numerous NCF white papers, that you should understand before developing an e-business solution. These white papers can be found from the following URL:

`http://www.software.ibm.com/ebusiness/`

2.1 Overview

The Network Computing Framework is a dynamic, proven model built from experience with customers that provides a strategy and technology vision for e-business. It describes many of the tools, methods and standards you can implement in order to deploy Web-based business application solutions quickly, effectively and with a lower cost of ownership than the traditional client/server paradigm. Many companies are including new platforms, such as thin clients and mobile clients, and they are connecting more of their extended enterprise, such as customers and suppliers. This requires cross-platform technology that handles a variety of requirements and that is manageable.

The NCF is based on open, industry Internet standards that allow you to leverage the best technology available today, but also remain open to quickly using new emerging technology.

Most companies have a large number of legacy systems where most of the data are stored, such as mainframe databases. NCF also allows you to connect to these existing databases, such as DB2 and IMS. Methods of including transaction processing are also described.

NCF helps you define your environment as you transform from client/server to network computing. See Figure 5 on page 22 for a graphical representation of NCF Architecture.

© Copyright IBM Corp. 1998

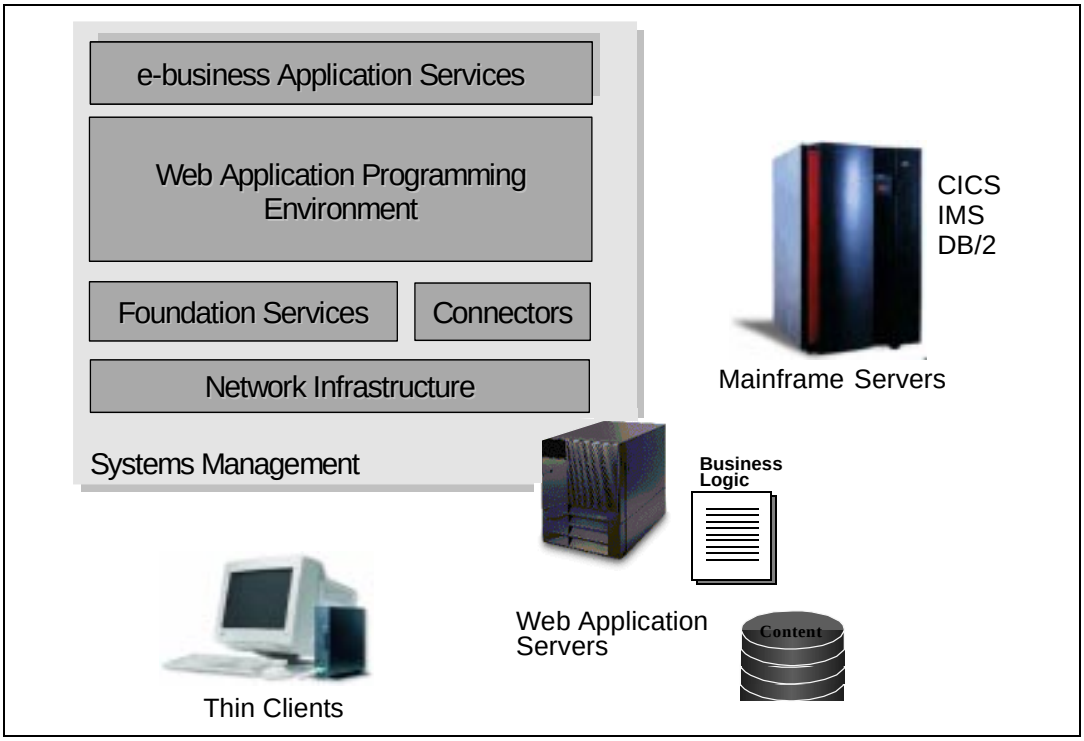


Figure 5. Network Computing Framework

The NCF helps developers because it addresses many of the issues confronting you in developing e-business applications:

- Which technology should I choose?
- How exactly do I build multi-platform applications?
- How do I plan for a growing system?
- How do I deal with security?

The NCF helps companies because it addresses these IT questions confronting your systems management group:

- How are new and existing applications connected?
- How can I extend my intranet to include my supplier?
- How can we maintain consistent application availability to our end users?

IBM created the NCF to help developers tackle the same issues you will address when developing Java-based applications for the NC.

2.2 Basic NCF Principles

NCF was designed to handle a very complex network computing environment. Fundamentally, it provides a way to generate new applications that compliment and extend existing applications using industry adopted standards such as HTTP, IIOP, TCP/IP, CORBA, and SSL with Java and technology.

NCF is based on these principles:

- Connect people to processes and information efficiently.

  This means that the business processes involving a human being are taken into account when designing a system. Applications like on-line catalogs need to connect to processes like order entry.

- Distribute work between clients and servers, lowering cost of ownership.

  Instead of a single monolithic application, the bulk of the business logic is designed to be processed by a powerful server. This allows the use of much thinner clients such as the NC, or even hand-held devices, which can lower the overall cost of deployment. These clients do not need local software installation or data backup.

- Link network computing solutions to existing applications and data.

  Even though you are about to tackle a Java project for the NC, you probably work in a company that already has an existing IT application suite with one or more databases. Most likely it will be necessary to connect to these systems. The NCF relies on Java servlets to connect to these back end systems. In addition, NCF relies on Java EnterpriseBeans to be the workhorse for these connections and to integrate into Java visual development tools.

- Provide a consistent programming model that empowers development teams while reducing costs.

  Developing applications for multiple platforms can be a complex task. The NCF relies on Java, which is platform independent. As a developer, you can work in your favorite editor on your favorite tool without the concern for the run-time environment. In addition, since Java is object-oriented, and therefore based on the principles of re-use, it is easier to build a single enterprise-wide business object model which can address the needs of many IT projects, thus reducing costs.

- Measure e-business solutions with the traditional IT benchmarks: security, reliability, scalability, manageability.

  The difference between simple sandbox examples and full scale production lies in the system's robustness. NCF addresses the fact that the nature of e-business applications dictates that the applications will deployed and managed on a large scale and will be supporting the company's business process.

- Flexibility is important.

  As you already know, the pace of Java technology is screaming fast. This means that new technology is being announced everyday. We know that you must put a stake in the ground and start your development process, otherwise you will keep waiting and waiting for stability that never comes. You can expect that as you begin your development, you should pick the current technology which best fits your solution, but that you will need to make changes to support unplanned growth. The NCF addresses the fact that the flexibility is important.

- Open standards and interoperability for communication, data access, content, services and applications.

  Even though IBM has a complete suite of products that supports the NCF, the products use common internet standards that allow for the use of other open implementations. It is important that you have a choice from variety of providers for these products.

- Model for future development.

  We expect that as the Internet technologies emerge, the NCF will be able to incorporate them into this e-business model.

2.3 Building NCF Solutions

The ultimate goal of the NCF is to enable you to develop business solutions that take advantage of the Internet technologies. The following discussion is intended to put your particular solution into a broader context as you probably have a specific problem to solve using Java and the NC. To help, we can sort the different types of solutions into three categories to highlight common requirements:

- Content Management Solutions
- Collaboration Solutions
- Commerce Solutions

2.3.1 Content Management Solutions

These solutions are generally focused on better ways to leverage your company's information:

- Applications that deliver general information that may change often but may have been distributed previously on paper medium through the company's internal mail. Corporate Intranets that provide useful information to employees, such as personal benefits programs, are a good example.
- Applications that advertise a presence on the Web. Random web-surfers will come into contact with these web pages by using powerful search engines that are widely available. The content on these pages can be static or it can be dynamically created for a specific user.
- Applications that provide access to a product line. These applications can often be used as a way for Internet users to order a specific product. Therefore, the application must be integrated with the company's order processing model.
- Applications that provide a means of managing a company's business operations such as defining product specifications, inventory planning, or shop floor production.
- Applications that manipulate multimedia files such as medical image records or CAD engineering plans.
- Applications that deliver encrypted, secure digital content

2.3.2 Collaboration Solutions

These solutions are generally focused on better ways for teams to work together:

- Any type of communication system that uses a messaging metaphor such as e-mail.
- Human resource solutions that require authorization like expense accounting.
- Project coordination solutions like task scheduling or document sharing.
- Intranets and Extranets that link development teams to external vendors in order to make the complete business process more efficient.

2.3.3 Commerce Solutions

These solutions generally focus on better ways of working with customers, suppliers and partners:

- Web sites that offer very personalized content based on extensive knowledge of a customer's preferences.
- Web sites that provide self-service buying or selling of products 24 hours per day.
- Web sites that provide online support to handle customer problem resolutions or a group discussion forum.
- Web sites that improve the coordination among businesses in a supply chain.

2.4 NCF Architecture

It should be clear that any solution for e-business (including yours) requires a flexible and extensible architecture that can scale to provide many users with simultaneous access to application services. The NCF addresses these issues by using an n-tier distributed application environment. In this model, the application is de-constructed into separate components that communicate with each other across the network. Each software component specializes in a particular role and is deployed on the appropriate hardware. A typical model defines three tiers as follows:

- A client is responsible for the presentation of data and for sending service requests to the application server. The Network Computer running Java applets would be positioned at this tier.
- An application server is responsible for the processing of business logic and for retrieving or saving data from the enterprise server. One application server should be able to handle many client requests simultaneously. Web Servers running Java servlets with EnterpriseBeans would be positioned at this tier.
- An enterprise server is responsible for the management of data and transactional applications. One server should be able to handle many application server requests simultaneously. Large data stores such as DB2 would be positioned at this tier.

The flexibility of the architecture lies in the fact that the components are interconnected through industry-standard Internet protocols. The extensibility of the architecture lies in the use of an object-oriented language such as Java. The scalability of the architecture lies in the distribution of the application onto several machines. To the end-user, the application and enterprise servers are being accessed through a single user interface on the client.

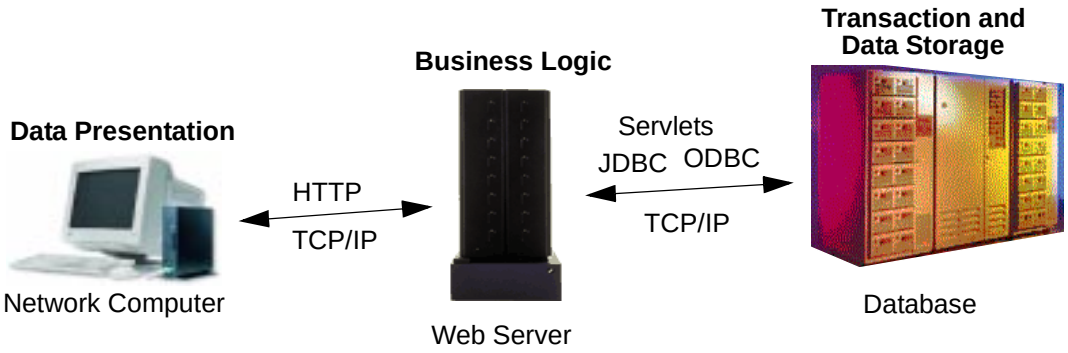


Figure 6. Logical Three-Tier Environment - NC to a Web Server to a Database

2.4.1 Key Elements

The industry-standard Internet protocols allow for a plug-and-play approach by using components from different vendors. The following are the key elements that compose the NCF:

- Clients
- Network Infrastructure
- Foundation Services
- Connectors
- Web Application Programming Environment
- Development Tools
- Systems Management
- e-business Application Services

2.4.2 Clients

The client element provides the end-user with universal access to application services using a thin client model, such as one that follows the Mobile Network Computer Reference Specification. These clients should rely on a just-in-time mechanism for delivery of software components. This can be accomplished using HTML, dynamic HTML, XML, Java applets, and through the use of protocols like HTTP, IIOP and RMI.

Web Browsers are fundamental in implementing the NCF client element because they are the ultimate thin client, and can be used to deliver content and services by downloading web pages with embedded applets from the

network when needed. This can be key to managing deployment from a central server. This approach also provides a lot of variety in user access and as such can provide a painless way to migrate from a simple web page using static HTML toward a complex web page using a Java applet with distributed objects. End users are generally unaware of these continuous deployment activities, other than the changes to the user interface as they see it.

A consistent programming model using Java does make your life easier. Since a distributed object application using Java employs the use of the same business object model on both the client and the server, you can save on both development and maintenance efforts.

Information exchange between application functions is encouraged through the use of JavaBeans and the Lotus Infobus technology. JavaBeans can participate on the bus by providing or receiving data. In this way, a spreadsheet application can collaborate with a charting application.

2.4.3 Network Infrastructure

The Network infrastructure plays an important role in the NCF, since networks today are increasingly global, and can span two or more companies, crossing insecure networks. It is built on TCP/IP, and provides services, such as:

- Network
- Security and Directory
- Host Integration
- Mobile Enablement
- Client Enablement

These services create the foundation of the NCF and are based on open Industry standards.

Network services include intranet and Internet access. To ensure security across the Internet, Virtual Private Networks (VPNs) can be established. This is very important in communicating securely with customers and suppliers. DHCP and DDNS services help minimize IP address and hostname administration, and implementation of the IPsec protocol assists in establishing VPNs. Web browsers can also provide high-level encryption of data across a network using SSL. To enable high-capacity Web site requests, routing mechanisms defined in the Network Infrastructure enable multiple systems to handle the incoming workload.

Security services are necessary for identification and authorization, enabling and controlling employee access to important company data. Security becomes even more important as people access information over insecure networks. Security Services support a Public Key infrastructure (PKI), which utilizes certificate-based authentication and encryption. Elements of security services include logon, secure communications, access control and auditing, data protection, non-repudiation, certificate services, and cryptography. The NCF Security services include interfaces for JCE 1.2, access controls based JDK 1.2, and JNDI.

Directory services are critical in the NCF, especially with the increase in distributed and Internet-based computing. It is important to minimize and eliminate multiple locations for the same information, such as employee data. Directory Services elements include APIs and protocols such as LDAP and Java Naming and Directory access (JNDI). Common schema, defined as a part of this service, enables multiple applications to share the same objects, allowing a single source for data. Meta-directory functions allow synchronization across LDAP and non-LDAP repositories to ensure data integrity and interoperability. LDAP repositories provide the actual storage and retrieval mechanisms for information on the network.

Host Integration services are important because much of the world's most critical data still reside on mainframe systems, such as the S/390. To enable access by Web-based users, such as employees and suppliers, Web Applications Servers need multiprotocol support (including SNA access), security interaction, universal client access, and a model for Web-to-Host applications. These are provided or defined by Host Integration Services.

Mobile Enablement services allow access by the increasing number of employees who work at multiple locations due to the dynamics of today's business requirements. Since relatively low-speed dial-up lines or wireless connections are implemented, bandwidth management, wireless services and network optimization are important. Agent services allow clients to work disconnected, improving user productivity. Management services enabling software distribution and remote command execution are important. Mechanisms for data replication using selective criteria are critical for mobile users.

Client Enablement services define a network-centric application model, allowing universal client access for platform-independent applications that are downloaded on demand as required by the client or the server. With this in mind, services such as application preference, configuration management, machine configuration management, user authentication, and systems

management, all enable full client access and administrative control, including the use of management architectures like Tivoli's TME.

2.4.4 Foundation Services

In the multiple tier architecture, the server is responsible for performing the work required for application service processing in handling the many client requests. Like clients, the server can be flexible in its growth over time. The NCF model provides the opportunity to integrate the server with additional enterprise servers as required.

The NCF services are defined as the Web (HTTP) server, mail and community, groupware, database, transaction and messaging services. All of these services are accessible through Java classes or JavaBeans.

The Web server plays the key role in this architecture. It serves as the integration point. Its main role is to accept requests and then invoke the appropriate business logic to satisfy the request. When you add ORB support to the Web server, objects on the client can send messages to remote objects on the server. This is critical in supporting distributed object applications. Through connectors, the Web server can link to existing enterprise applications, data and systems.

Mail and Community services provide the electronic communications that enable employees to share ideas quickly and to act as one unit. This includes e-mail, calendaring, group scheduling, chat and newsgroup discussions. Through the use of these technologies, businesses can establish closer relationships within their departments, as well as with customers and suppliers. The NCF supports these services through standard, open protocols, such as SMTP, X.400, POP3, IMAP4, IRC, NNTP, LDAP, X.509 and iCalendar object models.

Groupware services include any service that allows a group to cooperate in a virtual workspace by executing workflow processes. These services specialize in integrating content from many sources and result in increased team efficiency. Collaboration services like Lotus Notes integrate into the NCF by providing the Lotus Notes API.

Database services are crucial in that they provide mechanisms to access a database from the Web server environment. This allows you to use the existing relational or object database to provide new service through Web-based applications. These databases can contain traditional record-based data, or they may also contain complex data structures such as multimedia information. Access to these databases is provided by an API

using Java and JavaBeans, and also includes access using SQL, SQLJ and JDBC interfaces.

Transaction Services provide the robust, secure transaction environment needed by production systems. Scaling the system to handle a growing client base is done through replicating the server and load balancing. Keeping the application service accessible to authorized users is done through security.

2.4.5 Connectors

Much of a company's key data and large transaction-based applications typically reside on an enterprise system. Access from the Web server to these enterprise systems are provided by connectors. Since databases can run on a middle tier as well, the connection may be needed locally, or to a remote machine. The connector software, implemented as Java libraries, provides the integration needed to make the NCF a complete end-to-end model by linking the application services to the enterprise services.

Specifically, connectors can provide access to:

- Relational and hierarchical data stored in databases, like DB2 and IMS
- Application programs accessible through transaction systems such as CICS, IMS and TXSeries by Encina
- Object-oriented applications accessible through IIOP that support the OMG object services
- Industry applications from vendors such as SAP and Lotus
- External services outside the enterprise

Since the Web applications that users run often require fast access to data that may be on a host database, NCF includes support for open connections, connection pools and connection management. More information about connectors is available in Chapter 2.6.2, “Connectors” on page 39.

2.4.6 Web Application Programming Environment

The NCF describes a programming environment based on Java servlets and JavaBeans. These are some of the components from which the developer builds dynamic, transactional, secure Web-based business applications. Services are provided that help promote the separation of the business and presentation logic, which helps application developers dynamically adapt their solutions based on user input and client device types. Some of the more important aspects of the Web programming environment are described in more detail in Chapter 2.5, “NCF Development” on page 34.

2.4.7 Development Tools

Since the NCF is an open architecture, many tools can be used in the development and implementation of e-business applications. Some of the key tools offered by IBM are also described.

The IBM WebSphere Application Server Workbench provides a structured visualization of all components of a Web application. There is a Windows Explorer-like interface, with the application components shown on the left of the window. This workbench has the capability of launching other tools for viewing individual components, launching wizards to generate fragments or elements of a Web application, check-in and check-out support for library elements, and support for publishing elements of a Web application to a local or remote Web server so that end users can utilize the new applications.

VisualAge for Java is a comprehensive development environment for Java applications with support for both client-side and server-side development. It has visual programming and debugging facilities, and tools for constructing connector beans, which reduce the complexity in connecting Web applications to relational data and external services.

VisualAge Java Version 2.0, announced in July 1998, adds a High Performance Compiler (HPC), which is faster than other Just-In-Time (JIT) compilers. It creates a static executable after a Java application is ready for production, therefore bypassing the slower run-time conversion of bytecodes to the native platform code. Another feature, the Insight Profiler, allows you to see where your application is spending its time, allowing you to optimize the most critical components of your application.

Since NCF is based on standard Java libraries and interfaces, you can use VA Java or other development tools that use these standards.

NetObjects Fusion is currently the market leading visual development environment for Web pages. It facilitates the creation of dynamic web pages with HTML fragments and ScriptBuilder, a text editor with support for Dynamic Server Pages (DSP).

2.4.8 Systems Management

Management of the many components in a distributed application architecture that spans both an Intranet and Internet can be complicated. In addition to traditional IT administration, tools are needed to provide feedback and control of the run-time environment for servers, clients and applications. The NCF model can provide a foundation to define and integrate these tools into the system management process. The NCF Systems Management

services define management agents, frameworks, applications and consoles for the monitoring and execution of management activities.

In gathering information and resolving system issues, the aglet may provide an interesting venue for automated systems management. Written in Java and deployed onto the network, aglets are software agents that can move from machine to machine, performing needed authorized services or gathering data.

Servlets loaded by Web servers need to be managed and monitored. Secure web administration tools in the form of Applets can be delivered to an authorized client to provide universal access to systems management (IBM WebSphere, formerly known as Servlet Express, has an administration tool applet that manages and monitors servlets).

Named services provided by objects need to be managed in a distributed object application architecture. Tools that control the deployment and management of ORB-based objects are needed.

Tivoli is pioneering a collaborative management approach to distributed systems management, where management across multiple enterprises introduces new issues and levels of complexity.

2.4.9 e-business Application Services

As more businesses develop robust, secure application services, high-level object-based commerce APIs built on these industry standard Internet protocols will be needed. e-business application services will be defined over time as companies develop even closer relationships with business partners, vendors and suppliers.

One example of this type of service is the Lotus InfoBus. The InfoBus is a certified 100% Pure Java implementation of a data exchange mechanism for JavaBeans. It is a small Java API which allows applets, servlets or other components to connect to a bus and exchange information with any other entity on the bus. It is a mechanism for components within the same JVM to communicate with each other. The users consist of data producers and data consumers. The data can be spreadsheet information, the result of a database query, or nearly any data type. For more information on the Lotus InfoBus, refer to the following URL: `http://esuite.lotus.com`

IBM's Net.Commerce software allows businesses to transform static product information into dynamic, easy-to-use, searchable electronic catalogs. All the tools needed to start selling products on the Web are included, such as a store manager, where the catalog is created and maintained, a site manager,

where you can create and manage entire online malls, a merchant server, where customers can shop online and make queries about orders, and a template designer, a Java-based Web page design tool with customizable templates.

Another offering, Domino.Merchant, enables small and medium-sized company to take advantage of electronic commerce. Domino.Merchant includes all the tools necessary to develop an e-commerce application with a minimal amount of Internet programming experience. This product includes the base; the Domino Webserver.

2.5 NCF Development

Until now, we have described the NCF from an architectural perspective. Let's move this discussion from the abstract to the concrete. The NCF architecture defines the specifications of key element types, but any one element does not require a specific vendor's implementation or technology. Therefore, as a developer, you have a variety of options to choose from for each component. Multiply this out to include integration with exiting IT systems, and you can see that there is a large list of possible combinations of web-based technologies to create a single e-business solution. The goal of this section is to define the strengths of particular NCF-based Java technologies. Any of these technologies can help you implement your Java and network computing-based solution.

2.5.1 Dynamic Web Applications

You are probably familiar with static HTML web pages, where every user sees the same information. Perhaps it gets updated periodically by the owner, but essentially your Web browser is loading a file stored on a remote Web server. If you have ever used a web page containing an HTML form, the results of your 'submit' action are displayed as an HTML-based web page that was generated immediately by a process on the Web server, and then loaded by your browser as if you had followed a link there yourself. This is dynamic HTML. The web page that is produced will vary per request because the information on the form will likely vary from user to user.

The design pattern is that of a factory that produces an HTML product to order. In the past, the mechanism for assembling the HTML pieces has been to write CGI-BIN programs, but the preferred NCF method is to use server-side include (SSI). In this method, HTML servlet tags are used to indicate that the section of the web page between the tags will be produced by a specified servlet identified by an URL. In the case of a form, the servlet

is specified as the recipient of the form's action event. The servlet URL is invoked via HTTP by the browser with the form parameters the user had entered using either a GET or POST method.

The real advantage is that the servlet is written in Java and therefore has access to all of the NCF services available as Java APIs and JavaBeans that we have discussed. As Java objects, servlets are also capable of using RMI or ORB-based object services. It is clear that combining these elements together can satisfy the requirements for developing an e-business application that is integrated with the enterprise system.

2.5.2 JavaBeans

Within the Network Computing Framework for e-business (NCF), there is an NCF architecture that includes the concept of using a consistent programming model based on Java and associated technologies. JavaBeans is one of those major technologies used to implement the Java programming model.

Put simply, JavaBeans are reusable software components for use in the Java programming environment, introduced with the availability of the JDK 1.1. Because Java is platform-independent, JavaBeans are thus platform neutral. Beans are "drag and drop" components that can be deposited into an application during the development process. This places a requirement on JavaBeans to be structured in such a way that they cooperate with the development tools that will manipulate them.

JavaBeans must be developed according to the requirements specified in the JavaBeans 1.01 specification. This specification was developed with contributions from an industry-wide group that included IBM, Lotus, Sun, Netscape, Oracle, Corel, and many others. (Note that, at the time of this writing, a later release of the JavaBeans component model specification, code-named "Glasgow", was nearing completion. Search the Sun Microsystems Web site, `http://www.java.sun.com` for more information about Glasgow.)

JavaBeans have three important foundation pieces:

1. Properties (discrete named attributes)
2. Normal Java methods
3. Events generated to inform other beans of a change to this bean's internal state

By following the design and programming conventions outlined in the JavaBeans specification, JavaBeans developers create beans that can interact with each other (via event notifications) according to an application-specific predefined protocol. Strict adherence to the conventions (outlined in the JavaBeans specification) is what provides the ability for developers to combine beans that come from multiple sources into a single application.

This book will not teach the reader in detail about how to do JavaBean programming. Many fine books exist on that subject already, and the reader should consider reading one or more of those as a required prerequisite for reading this book (for example, the IBM redbook, *Cooking with Beans in the Enterprise*, SG24-7006). We will briefly discuss some of the important IBM VisualAge Java programming procedures used to develop our examples that use JavaBeans.

2.5.2.1 How Do JavaBeans Relate to NCs?

JavaBeans are created using Java classes. The JavaBean classes are just designed following the JavaBean guidelines. "Normal" classes can communicate using RMI or CORBA, and so can JavaBean-based classes. Just like "normal" Java classes that can be downloaded to an NC for execution as part of an applet running on the NC, JavaBean classes can also run on the NC. An applet can be designed to use its JavaBeans to communicate with other JavaBeans across the network using RMI or CORBA.

2.5.2.2 NCF Programming Using JavaBeans

The NCF programming model focuses on object re-use and as such relies heavily on the use of JavaBeans. The JavaBean programming model is so useful that IBM designed all NCF connectors and services to be exposed as Java classes and/or JavaBeans. Any NCF development tool should accommodate the creation of JavaBeans and access to exposed NCF services through NCF-supplied JavaBeans.

2.5.3 Enterprise JavaBeans

2.5.3.1 What Are Enterprise JavaBeans?

The Enterprise JavaBean (EJB) architecture is a cross-platform architecture that specifies conventions for building distributed, multi-tier Java applications. EJBs are not a product. Instead, EJB support must be built into software development tools to help in the creation of individual EJBs. EJB is new, having been announced in March 1998, and the V1.0 specification is available. At the time of this writing, VisualAge for Java Version 2.0 does not

support EJBs, but a future version is expected to add support for this important new architecture. Being able to support new functions and architectures is very important for the NCF.

2.5.3.2 Enterprise JavaBeans and CORBA

JavaSoft has adopted the CORBA distributed object model, per its June 1997 announcement. The Enterprise JavaBean architecture is compatible with CORBA. The EJB architecture specifies mappings to the CORBA APIs, so the EJB developer uses Java APIs, not CORBA APIs.

JDK 1.2 is expected to include a CORBA/IIOP ORB as a standard extension. JDK 1.2's development environment will provide the capability of generating Java-based CORBA stubs and skeletons from IDL. This is also mentioned in Chapter 5.3.3, "RMI over IIOP" on page 117.

2.5.3.3 Do Enterprise JavaBeans Relate to NCs?

Enterprise JavaBeans (EJBs) don't directly relate to NCs. EJBs provide a model for the development of server-side applications. A frequently used analogy is that JavaBeans are used in the development of client-side applications, while EJBs will be used in the development of server-side applications.

The EJB architecture provides a standard mapping of the EJB protocols onto CORBA, thus allowing for the use of CORBA-based servers. So, applets that run in a CORBA environment on the NC will eventually be able to communicate with servlets that do contain EJBs.

2.5.4 Developing e-business Applications

Developing an e-business solution requires a variety of roles and tools. A complete web-based application will involve authoring web pages, developing JavaBeans, applets and servlets and managing the assembly of these components into a working system.

As your solution development scales in resources, managing the repository of e-business elements will require traditional IT administration that handles version control and change management.

2.5.4.1 Content Authoring Tools

Content authoring tools create and manage multi-media content like graphics, image, audio, animation and video. The NCF supports tools that are based on the industry-standard formats like HTML, GIF, JPEG, and MPEG.

2.5.4.2 Integrated Application Development Environment

Integrated development environments (IDEs) allow a developer to build and test complete distributed, Java-based, networked computing applications. Team IDE environments allow many people to collaborate on the same project in a more efficient way. Enterprise IDEs provide additional services for developing and testing access to data stores and transactional applications.

2.5.4.3 Content Assembly and Management

Content assembly tools bind the NCF elements together. Web pages can be constructed from a palette of Applets and Servlets that implement the JavaBean interface. Using a visual construction method, the NCF elements can be dropped onto an abstract space and connected using exposed services.

2.6 Software

IBM and other vendors offer a portfolio of products that support the complete NCF model. These products include the key elements that the NCF is built upon such as servers, connectors, network infrastructure, systems management, tools and clients. The following sections describe some of IBM's NCF software products for e-business.

2.6.1 Servers

This section describes some of the software servers related to the NCF model.

2.6.1.1 Domino Go Webserver

This product provides Web services that integrate well with network computer applications. It includes a Java Virtual Machine to run sServlets and an ORB to provide IIOP access to CORBA objects. Its infrastructure is based on HTTP, LDAP and SSL protocols.

2.6.1.2 Apache HTTP Server

The Apache HTTP server is one result of a collaborative software development effort aimed at creating a powerful, enterprise-ready implementation of an HTTP (Web) server. One unique item is that the source code is freely available, which contributes to the fact that it is widely used. In fact, a June 1998 survey by Netcraft found that Apache is the most widely implemented Web server in production.

2.6.1.3 IBM WebSphere

Formerly known as ServletExpress, this product is a plug-in to Domino Go Web Server and provides a Servlet engine that supports HTTP session tracking and some extensions to the Java IDL ORB. According to a June, 1998 announcement, IBM is including the Apache HTTP Server in WebSphere, in addition to the Domino Go Web Server.

2.6.1.4 Lotus Domino Mail

This product provides e-mail, newsgroups, chat, calendars and scheduling services. Its infrastructure is based on LDAP, X.509 and SSL protocols.

2.6.1.5 Lotus Domino

This product integrates with Lotus Domino mail and provides workflow and collaboration services. Lotus Domino also provides an application framework that supports Web content management, commerce and push technologies.

2.6.1.6 IBM DB2 Universal Database

This product supports enterprise services for storing data and multimedia objects and services for querying and online transaction processing. The database supports Net.Data, Java and JDBC.

2.6.1.7 IBM Transaction Series

This product add CICS and Encina enterprise services for transaction monitoring. This also includes Component Broker Connector, which provides a full set of OMG object services.

2.6.2 Connectors

This section describes some of the software connection mechanisms pertinent to the NCF model.

2.6.2.1 IMS

These connectors provide a link to transactions in IMS from the Web server.

- WWW Templates
- IMS Web
- IMS TCP/IP OTMA Connection
- IMS Client for Java

2.6.2.2 e-Network Host on Demand

This product is an Internet-to SNA connectivity solution that provides standard Telnet 3270 application access through a Java Applet.

2.6.2.3 CICS Internet Gateway

This product translates the output from CICS into HTML through the use of CGI scripts from a Web server.

2.6.2.4 CICS Internet Gateway for Java and CICS Clients

This product provides access to CICS from any Java-enabled Web client.

2.6.2.5 DCE Encina Lightweight Client

This product extends a company's DCE and Encina-based installations to include the Web.

2.6.2.6 MQSeries Internet Gateway

This product links Internet-connected Web browsers with MQSeries applications.

2.6.2.7 MQSeries Client for Java

This product provides a Java API to access MQSeries applications.

2.6.2.8 Net.Data

This product allows Web developers to build dynamic Internet applications using Web macros. It also provides database connectivity to information stored in flat and relational files.

2.6.2.9 Lotus Domino.Connect

This product provides access from Lotus Notes applications to existing data, transactions and applications such as IBM DB2, CICS, IMS, MQSeries and SAP R/3.

2.6.3 Clients

There is an extremely wide variety of clients available that support the NCF client model. IBM offers the following:

2.6.3.1 Network Station

As described in Chapter 1.2, "A Network Computer—The IBM Network Station" on page 2, the IBM Network Station meets the requirements of the NCF client model.

2.6.3.2 Workspace On-Demand

This product includes Java Virtual Machine Version 1.1, which can run Java applets and applications and can utilize the standard application communications in a distributed network. For a brief discussion of

WorkSpace On-Demand, see Chapter 1.8, “Network Computing using WorkSpace On-Demand” on page 18.

Chapter 3. Application Example - WorldWide Trucking Company

This chapter provides background information about an imaginary business environment that is used to help illustrate the concepts outlined in this redbook.

3.1 Programming Introduction

This chapter describes a small collection of simple Java-based applications to demonstrate some of the key concepts of this redbook. The examples are purposely kept very simple so that the code excerpts can focus on these key concepts. Of course, real applications would be much more complex in terms of their business logic, design and error handling capabilities. The code relating to this book's core concepts (use of JavaBeans, CORBA, and so on) is coded just like it would be for production applications. Your code will definitely need to contain these lines to function in a Java-based distributed object environment.

3.2 WorldWide Trucking Company

WorldWide Trucking Company (WWTC) is a multi-national transportation company that owns a fleet of trucks that deliver cargo across state and national boundaries.

WWTC is a modern company that needs to keep their operation costs low in order to remain number one in their field. In the past, they relied heavily on doing business over the telephone to accomplish their daily business operations. This approach required personnel to cover the phones and perform the required duties. To cut costs, WWTC has been asking their Information Technology (I/T) department to develop computerized solutions to automate portions of their daily business operations.

In the past, WWTC had been using PCs in their company running applications on the IBM OS/2 and Microsoft Windows NT platforms. WWTC's programmers in the I/T department were knowledgeable object-oriented programmers who could create distributed client -server applications. The servers were often IBM RS/6000 systems running AIX (IBM's UNIX implementation). WWTC had been big users of IBM's CORBA-compliant Distributed System Object Model (DSOM) and IBM's VisualAge C++.

WWTC's I/T department made a business decision two years ago to move their application design and development efforts into an approach that emphasized:

- Using Network Computers (NCs) for base hardware platforms
- Using Java as the development environment
- Emphasizing the use of open industry standards (CORBA, Enterprise JavaBeans, and so on)

The I/T department decided that the environment outlined above (illustrated in Figure 7 on page 44) was both robust (use of open standards guaranteed many choices for software vendors) and cost-effective (NCs were expected to lower WWTC's total cost of ownership). They chose to use components from IBM's Network Computing Framework (NCF) for e-business because many of its components were based on the open industry standards (CORBA, Java, and so on) developed by IBM and other industry leaders.

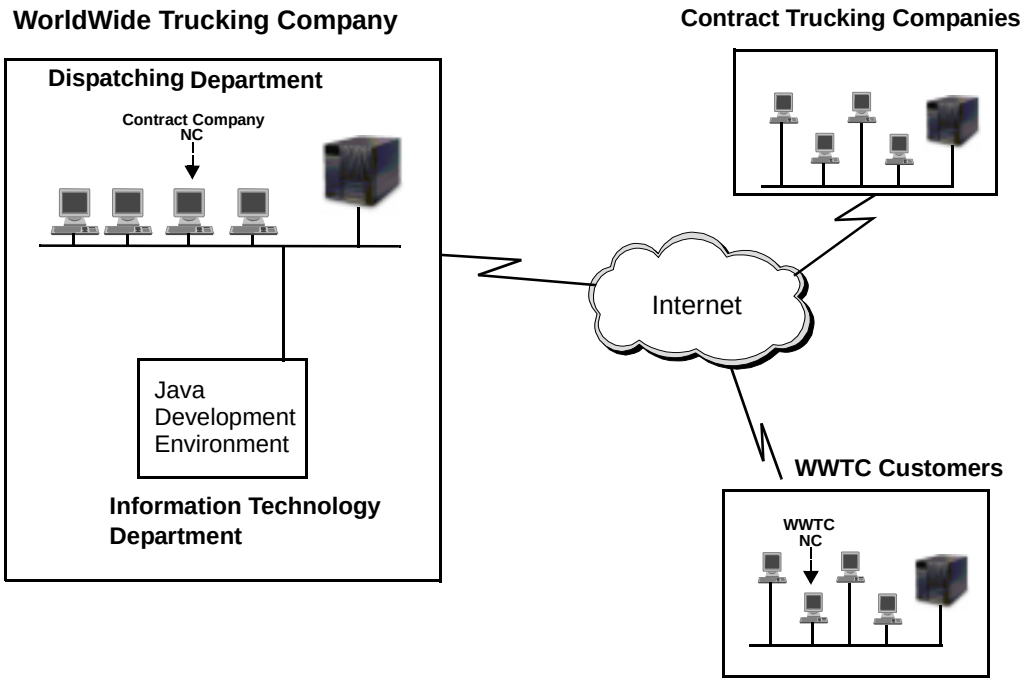


Figure 7. WorldWide Trucking Company's NC-Based Computing Environment

WWTC uses NCs internally in their business operations, and they often give one or two free NCs to important customers as an incentive for those customers to do business with WWTC. The customers use the NCs as gateways into WWTC's business software environment. WWTC feels that it makes good business sense to focus the customer on WWTC as the supplier, so WWTC adds physical advertising labels to the NC box identifying WWTC as the source for all their transportation needs. See Figure 8 on page 45 for a representation of this environment.

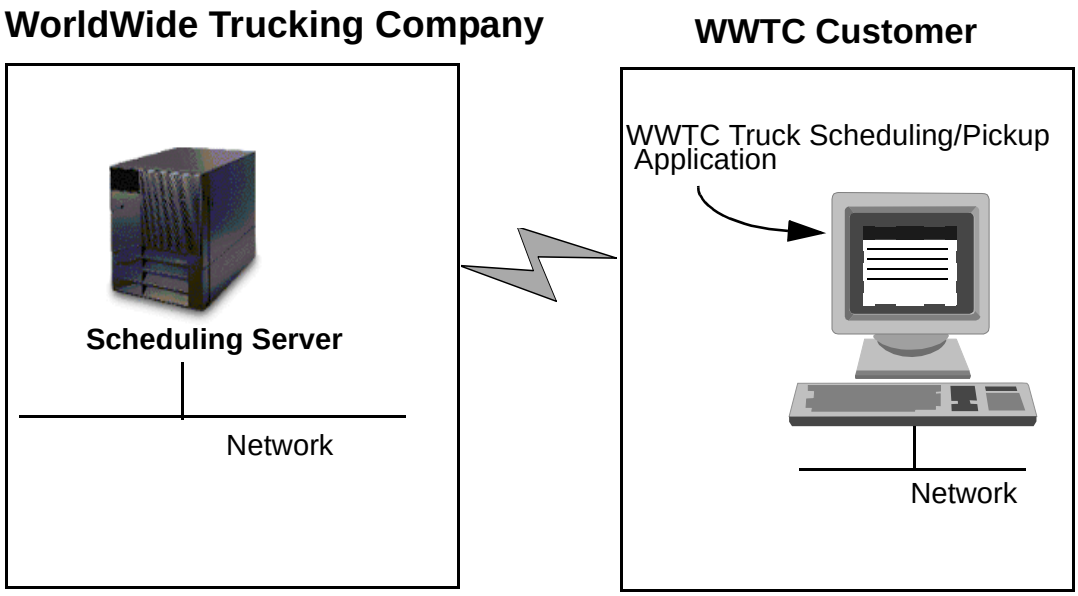


Figure 8. WWTC Customer NC Connectivity

NCs can be configured to download an aApplet as part of the download of the NC's boot software when the NC is turned on. This helps simplify the use of the NC at the customer site, because the desired Applet is started without user intervention. Furthermore, WWTC often embeds their corporate logo into the main splash screen for many of their Applets.

WWTC also felt that there was good technical justification for giving away some free NCs. By supplying these seed units, WWTC helped set technical standards for the customers regarding the technical requirements (processor speed, RAM amount, and so on) needed for NC power. This practice also helped WWTC software designers and developers with a technical profile for the client machines that their software might run on.

This chapter describes some of the applications that were created as a result of the I/T department's strategic decisions. An overview of these results is shown in Figure 9 on page 46.

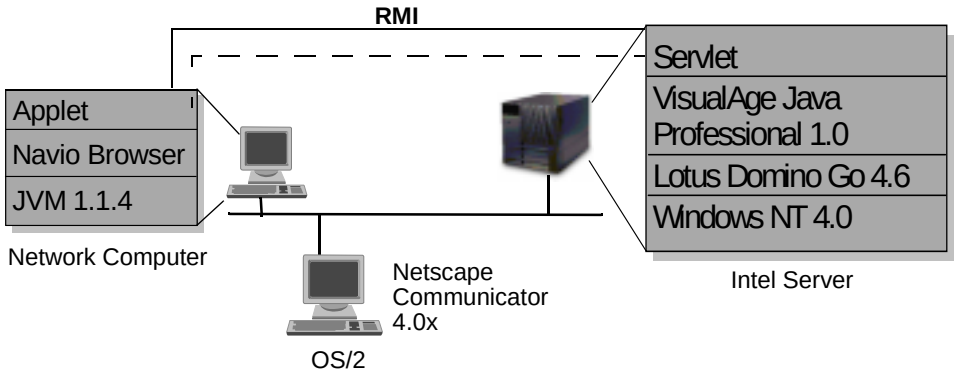


Figure 9. WWTC's General Business Software Application Architecture

3.3 Transport Request Application

WWTC wanted to give customers an opportunity to use a software application to request pickup and delivery of cargo. The transport requests are routed to WWTC's Scheduling department for execution.

WWTC created an Applet that could be run on the customer's NC. The Applet communicates with a server application on a computer in WWTC's Dispatching department. Using this application (see Figure 10 on page 47), the customer requests a number of trucks to come to a specified location at a specified time.

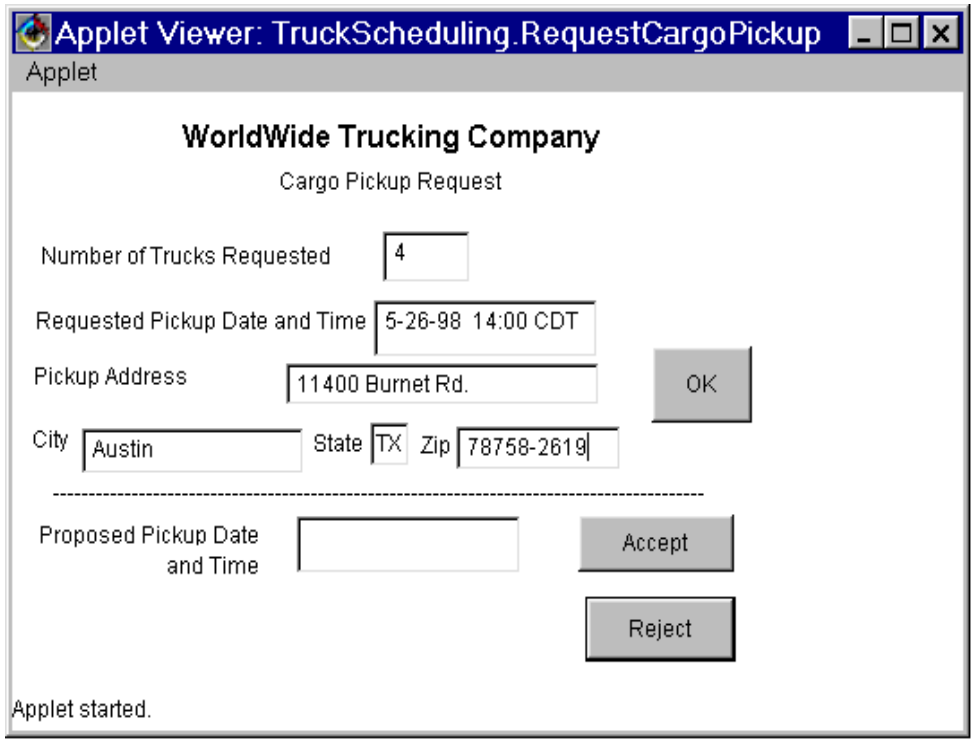


Figure 10. Customer Pickup Request

WWTC's dispatchers use viewing software (not described in this book) to look at the request. For simplicity, we assume that WWTC always has enough trucks available (either trucks they already own or trucks they can quickly rent) to always approve the customer's request for the specific number of trucks. The real issue is time. When can WWTC get the trucks to the customer? Can they get them there by the time requested by the customer?

The dispatchers look at the truck fleet's current state (truck locations, driver availability, and so on) to determine the time that they will commit to delivering trucks to the address requested by the customer. The dispatchers send that projected arrival time back (see Figure 11 on page 48) and wait to see if the customer finds it acceptable and gives it final approval or if the customer rejects the proposed time.

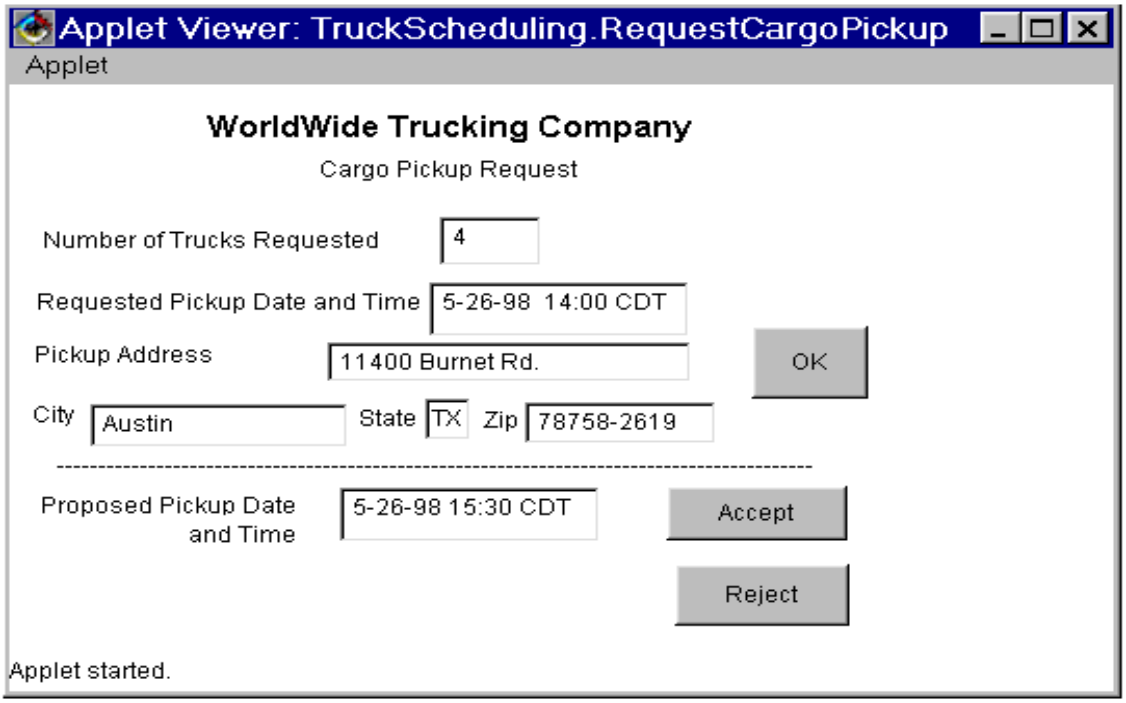


Figure 11. WWTC Dispatcher Response to Pickup Request

Because WWTC gives great service, they usually agree to meet the customer's requested time or propose something very close to it. In the unusual circumstance that the time is totally unacceptable, the customer can just use the `Reject` button.

3.3.1 Application Re-Use

WWTC's Dispatching department had previously indicated that they needed a similar transport request capability when they rented additional trucks from other trucking companies (see Chapter 3.4, "Truck Tracking Application" on page 50 for an explanation of why WWTC sometimes needs to rent additional trucks).

The I/T department deliberately designed the Transport Request Application so that it could also be used by WWTC's Dispatching department. WWTC's computer that runs the application communicates with a server on the truck supplier's computer. For example, WWTC dispatchers can run the application to request trucks from the RedRyder U-Rent Truck Company. They simply use the same pickup request information (customer address, requested

pickup time, and so forth) that WWTC's customer supplied in the original pickup request.

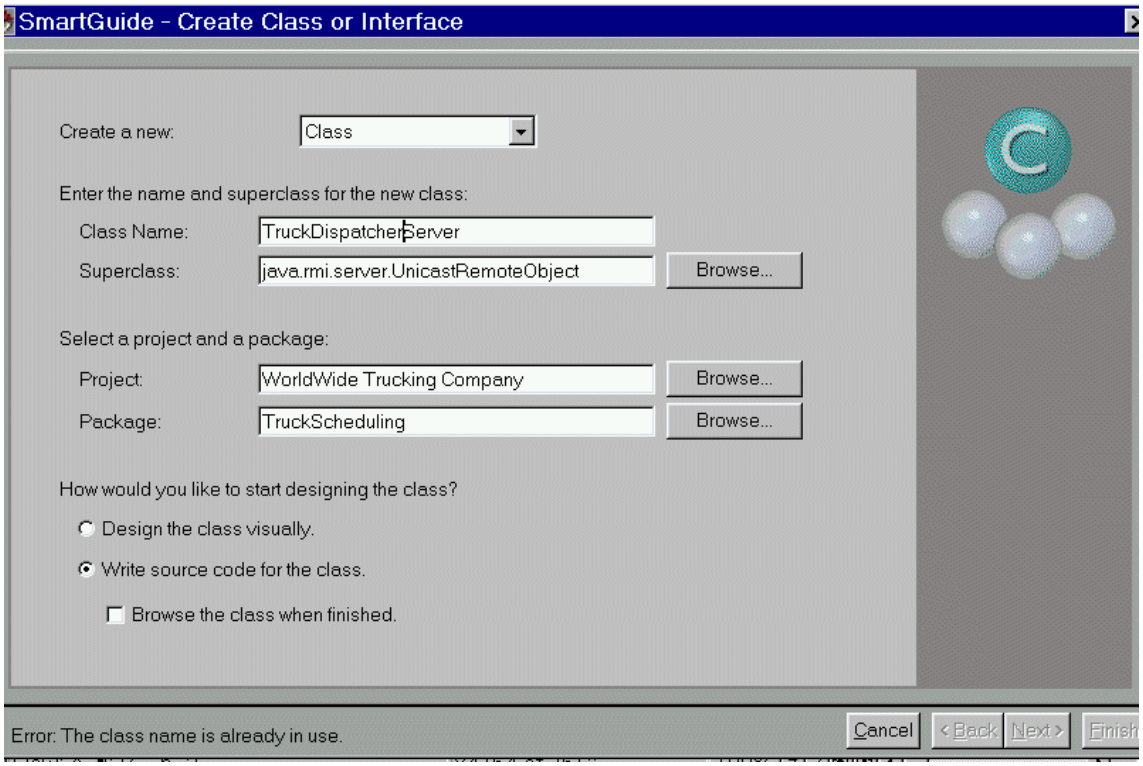


Figure 12. WWTC's Suppliers Use TruckDispatcher Bean for their Server

WWTC chose to share their bean design for the TruckDispatcher with some of their preferred truck suppliers. This allowed WWTC to re-use the pickup request Applet without changes. The preferred truck supplier used the TruckDispatcher bean as the foundation for their server. They were free to implement their own business logic regarding scheduling.

In the future, WWTC is considering automating the dispatching function by capturing the business rules that control the scheduling of trucks. Then, when requests come in from customers that require renting outside trucks, the request will be evaluated and automatically routed electronically to companies like RedRyder. Of course, the role of the physical dispatching staff will not completely go away, since it is likely that some WWTC customers and some truck rental companies may not choose to use WWTC software. Doing business with those business partners will still require using the telephone.

3.4 Truck Tracking Application

No trucking company can afford to own enough trucks to handle peak shipping days, so WWTC only owns enough trucks for their usual shipping requirements. When WWTC has more business than usual, WWTC will rent or lease trucks from some other trucking company and contract with that company for the delivery of the excess load.

These outside trucking companies have a contractual obligation to report the location and status of their trucks to WWTC at least once per hour. WWTC would prefer to receive updates more frequently.

In the past, the outside trucking company personnel simply called WWTC's Dispatching department and verbally told them where the trucks were. Now, WWTC has developed a software application that is made available to the outside trucking companies to let them update truck status via computer.

WWTC's Dispatching department also uses the application to update the status and location of WWTC's own trucks. In the future, The I/T department plans to extend this application to automatically accept radio telemetry transmissions from WWTC trucks in order to remove the human data entry activities.

Also, WWTC Customer Service personnel frequently use this application to view truck status and location in order to advise WWTC customers regarding truck arrival times. WWTC makes this viewing software available to its customers, so that the customers can check the projected arrival times themselves. WWTC customers appreciate the extra help that this viewing capability provides as they plan their shipping dock activities.

Chapter 4. Application Design Issues

As the WorldWide Trucking Company (described in Chapter 3, “Application Example - WorldWide Trucking Company” on page 43) develops its network-computing based application set and infrastructure, they must inevitably resolve some of the issues described in this chapter.

The sections in this chapter describe potential problems that the application designer faces when writing an NC-based application. In most (if not all) cases, the designer of a PC-based application could conceivably face the same challenges (limited RAM, limited or non-existent persistent storage, network interruptions, power fluctuations, and so so). The key point is that the NC application designer must face these issues every day for each and every single application, and must plan for handling all of them.

The table below lists some of the issues when writing Java applications for NC clients, and some of the possible resolutions to these issues. They are discussed in more detail throughout this chapter.

Issue	Possible Resolution
Network-based Execution	Keep software/data local to NC
Limited Local Resources	Minimize Footprint
Limited RAM	Download & present only needed data
Multiple JDK Versions Platform-specific behavior	Test, test, test
Transaction rollbacks	Store state data on server
Maintaining session context	User validation

Table 2. Application Design Issues for Network Computers

4.1 Thin Client

A thin client hardware architecture is one that has a minimum of local resources. Notable by its absence in the NC configuration is a persistent local storage medium, such as a hard disk.

The absence of local storage requires the NC to go to the network for its resource needs. In the absence of persistent local storage, the NC must go to the network to obtain, for example, its application software.

Java application software is made up of class files that contain the software and supporting files that contain the screen images, the beans and other control information that contribute to the screen's behavior. The cumulative size of all these pieces when running in the client's memory is referred to as the Applet's footprint. However, the amount of data (class files) that flows across the network can differ from the applet's footprint, depending on the behavior of the application.

4.1.1 Issue: Limited Local Resources

When an Applet executes, all the files that provide its screens and functionality are downloaded as required, one at a time, to the NC where they will be used. Clearly, downloading these files one at a time can be a time consuming operation.

There are other options available to the application designer that we will discuss in the following sections.

4.1.2 Principle: Minimize Footprint Size

One easy way to reduce an Applet's footprint, that is, the size of what needs to be downloaded, is to compress the files into a more space efficient format and then transfer the smaller files across the network when the applet begins execution. A Java Archive (JAR), introduced in Java 1.1, is a compressed archive of java class files used by an Applet. The mechanism used is similar to PKZIP, and some versions of PKUNZIP may be able to view and decompress JAR files, but we don't recommend this method. A JAR file can be decompressed by using the `jar xf <jarfile>` command.

The JAR file also includes checksum and manifest information, that is, the list of files contained within it. During Applet execution on the NC, these compressed file members of a JAR file are found quickly and are downloaded in their compressed format to the NC, where the JAR file is then dynamically decompressed, and requires no special interaction by the user. The time savings of sending a compressed file across the network just once, instead of for each class file, can be significant, especially as the class files get larger.

A JAR file can be created using the `jar` tool and must be prepared before the run time of the application. At run time, the JAR file is downloaded through the specification of the ARCHIVE= tag in the HTML.

4.1.3 Principle: Keep Software Local

The executable code for a Java application is found in the .class files. This means that the JVM has frequent need for these files during execution. For

example, each time a new object is created by the JVM, it needs the class specification found in the .class file to create the object.

Since all the software that runs on the NC must be brought over from the application server, it is sensible to keep locally at least those parts of the software that are used repeatedly.

Here, again, the JAR file is a good solution to the problem of continuously downloading fragments of the application at run time. If the JAR contains all parts of an application that a user is likely to need, then the JAR needs to be downloaded only once. When a new subsection of the application is entered, it already exists locally. If the subsection of functionality needed by a user is clearly defined, the JAR can be downloaded when the application is started up and remain on the NC for the duration of the user's session. This minimizes the need for the NC to go to the network during execution of the application for software resources.

4.2 Thin Data

Not all files used by an Applet are software. The software captures data from the user, manipulates data, provides for data to be stored in a persistent form and presents data to the user. But much of the data used by the NC software must be brought across the network at run time in the form of files. In the thin data concept, only the data needed by the NC are downloaded to reside in memory on the NC.

4.2.1 Issue: Limited Local Space

Depending on the application, some data files can get large. The issue of limited space on the NC can become an issue. With no persistent storage, it is important to have sufficient RAM in the NC. Fat client solutions for data manipulation are not optimal, and in some cases, will not work here.

4.2.1.1 Principle: Download Only What is Needed

To minimize network traffic and to avoid unneeded download of data to the NCs, available storage requires a scheme where only the data needed by the NC to perform its current task are downloaded. This requires identifying exactly what data are needed for each task the Applet must perform. This definition of the data usage will serve as a blueprint to determine what data must be downloaded for each of the applet's tasks.

For example, an Applet might be responsible for providing the functionality for a user to capture data from the image of a scanned document. From the

scanned document, three pieces of information must be keyed: Customer Name, Customer Address, and Customer Number.

In this example, the scanned image is the source data that support the entire operation. This data must be downloaded to the NC at the beginning, to make the image available to the user. The Customer Name and Address can be retrieved directly from the image and this data should be uploaded to the server and put into persistent storage as soon as possible.

Let's assume the customer name must be verified before writing it to a persistent medium, such as a data base. If the name fails the validation, the operator might get a list of close matches to choose from. This list must then be downloaded to the NC. For example, If only five names can be displayed at one time, then one option is to download only five or six names at one time. To be consistent with the thin data scheme, only the data to be displayed should be downloaded.

4.2.1.2 Principle: Persist Reusable Data

While an Applet is executing on the client system, not all data are useful at all times. Here again, a clear definition of what data are needed at what time during the fulfillment of an Applet's tasks is vital. If data are needed intermittently during the execution of an Applet, there is no reason to keep downloading it repeatedly. Reusable data should persist on the NC as long as its persistence does not interfere with other operations, and it does not utilize an excessive amount of memory.

In the data entry example described above, the scanned image is a case of data that should persist on the NC. Because images represent a large piece of data that are used intermittently by the applet, the image should be retained on the NC until it is no longer needed. Even if the applet passes control to another applet, the image file should persist until all the application software is finished with it. When the data entry operator has completely finished with the image, it can be discarded. But even when the image is discarded, the class it was instantiated in can still persist and the next image can be loaded into it. This saves the overhead of creating a new object for the next image. Object persistence is covered in detail later in this chapter (in Section 4.7, “Remote Object Persistence and Serialization” on page 62).

4.3 Portability

One of the Java programming language's most salient features is its platform independence. Although this is mostly true, Java is not independent of

changes to its language releases or differences in the behavior of the operating system the JVM runs on.

4.3.1 Issue: Different Versions of the JDK

Each new version of the JDK brings with it new functionality. The changes included in each new version of a JDK are documented in a file called “Changes” that is associated with each new JDK release.

The functionality provided by JDKs resides in classes. These classes can then be used in application programs. A problem, however, arises when a program that relies on new functionality is executed on a platform where the JDK does not include the new classes. Clearly, if the classes do not exist in the JDK, the application program cannot use them, and a "Class not found" exception may occur. In such a case, the developer should handle the error and gracefully inform the user. Downward compatibility in Java should always be tested, not assumed.

4.3.1.1 Principle: JDK Version Control

When Java programs rely on functionality provided by specific versions of the JDK, then this functionality must be provided on all the platforms on which the programs are to run. This is a clear version control problem. When Java programs begin to rely on parts of a JDK that do not exist in previous versions, then the new JDK must be provided on the target platforms. These means providing the new JDK on the application servers from which the programs will be accessed by client NCs. Those personnel responsible for development and maintenance of the Java environment for end users must ensure that the JDK levels match between the development and the target platform.

4.3.2 Issue: Different Behavior of Platforms

A Java program will run on any platform that has a JDK that is consistent with the JDK under which the program was developed. Java provides this platform independence by using a set of standard modules through which the JVM performs all its interactions with the operating system. There is a standard set of interface routines for each operating system platform on which a JVM can run. The standard routines are referred to as standard because regardless of the OS platform, they provide the same interface to the JVM. This standard interface encapsulates the operating system and hides its implementation from the JVM. The standard interface allows the JVM to be OS platform independent.

Although the interface to the JVM is the same across OS platforms, the implementation of the standard routines is platform dependent and, therefore, varies slightly from platform to platform. In addition, the behavior of the underlying OS itself also differs for a given operation.

The variations in behavior due to the implementation of the standard routines and due to variations in the behavior of the OS can account for a significant difference in the behavior of Java applets on different OS platforms.

4.3.2.1 Principle: Code Once, Test Everywhere

Testing is the only means of identifying variations in behavior and determining if these variations are sufficient to constitute a deviation from the behavior defined in the program's specification. The discussion in Section 5.2.4.3, "Considerations for Applet Programming" on page 79, is a prime example of the need for this principle.

The Java standard routines for interfacing with the OS are part of the JDK. This is significant because this allows the Java-based code files to be platform independent. But, as described above, this platform independence is somewhat debilitated by the differences in the behavior of standard routines and the differences in the underlying operating system.

The Java code must be thoroughly tested for each operating system platform on which it is to run. Because the implementations of the standard routines vary between OS platforms, a Java program must be tested for possible unacceptable behavior that results from these implementation differences.

4.4 Soft Failure

A soft failure is a software problem that occurs at run time, and where the application is able to recover from the problem and eventually resume normal operation, without significant user intervention. A common example of a soft failure is when a Web browser is unable to access a desired Web page at a particular time due to an inability to access the corresponding Web server, but then later successfully makes connection to the server and displays this page when the user requests it again.

In each of the following sections, the reader is advised to look at the potential solutions as representing soft failures. There are no easy answers to adverse operating conditions. The design goal should be to reset the application to a known state or (when reset is not possible) to gracefully terminate the application in a manner that encourages the user to restart the application.

However, failures and bottlenecks will still occur. Soft failures will be affected by certain immutable NC characteristics (by today's standards):

1. No persistent local storage
2. Dependence on network availability (a dead network means no place to load applications from)

It is the responsibility of the application designer and programmer to determine a suitable course of action when these inevitable events occur.

4.5 Transaction Processing

WorldWide Trucking Company (see description in Chapter 3, "Application Example - WorldWide Trucking Company" on page 43) is a transaction-oriented business. Virtually everything that happens during their daily business operations is structured as a transaction. Examples include:

- Updating truck status and location
- Scheduling cargo pickup
- Transmitting customs documents

As WWTC became a heavily object-oriented design and development shop, they found themselves operating on object states. They needed a way to reliably control the state of an object. Direct manipulation of relational database records became less important as the records became "wrappered" by objects.

4.5.1 Issue: Transaction Rollback in a Diskless Environment

As WWTC started developing applications to run on NCs, they noticed some differences between developing transactions for PC-based applications versus NC-based applications.

The chief difference involved the lack of persistent storage on the NC. NCs simply have no hard disks. Although many NCs do include a PCMCIA card slot, and there are PCMCIA cards available that are essentially small 100 MB hard disks, developers can not count on these slots being on a particular NC.

As long-time database programmers, the WWTC staff was accustomed to issuing `begin_transaction`, `commit_transaction` and `rollback_transaction` commands inside their programs. Of course, these transactions were taking place on a remote database server machine. If the client application stopped functioning, then eventually the database management software on the server would time-out the transaction and automatically roll-back the

in-progress transaction. No problems here with migrating to NCs, because all the rollback work is done on the database server. The use of an NC didn't change that situation.

The real issue was how to do rollbacks on object states when those objects resided directly on the NC. In the last few years, WWTC programmers made frequent use of CORBA Object Services, particularly the Common Object Specification's Transaction and Concurrency Services. These services provided standard methods for starting and completing transactions on an actual object. CORBA doesn't require that the object represent something in a physical database. It can just as easily be a RAM-based object on the local client computer. That is the beauty and strength of the CORBA transaction model. You can execute transactions on any object that provides transaction commit and rollback support. It is up to the object's designer to determine what a transaction commit or rollback means for that particular class of objects.

The problem is that most developers use the hard disk to store object state while a transaction is in progress. Then, if the transaction hangs (for example, somebody turns off the computer), the object's previous state can be restored from the disk when the application is restarted. On an NC, there is no disk, so this approach doesn't work here. So, how do we address this problem?

4.5.1.1 Principle: Don't Depend on Rollbacks for NCs

The rollback problem, where it depends on a hard disk on the local client computer, can't be directly solved. However, there are other options:

1. Store object state on a remote node that does offer persistence.

   Request storage of object state data in a persistent environment. The client application that wants to receive the restored state must explicitly ask the storage node for the stored data. In most cases, this will be a Java application server, possibly the same server conducting the transaction.

   Disadvantages:

   - Increased network traffic
   - Dependence on remote node availability

2. Respond to externally initiated rollbacks

   The client application on the NC receives a rollback notice that includes needed state data to restore object state. This is a variation on the first alternative. Here, you send (publish) object state data to any interested

party that has registered (subscribed) to receive object state data updates. Any of these registered parties (subscribers) can then ask the NC client application to rollback the object to the state specified in the rollback request.

Disadvantages:

- Increased network traffic
- Increased overhead in managing subscribers
- Increased application complexity (can be mitigated somewhat by using event notification software, such as IBM's MQSeries)
- Need to coordinate application's rollback activities with ongoing application user's interaction with application. Must notify user when an external rollback has occurred.

3. Prompt user for correct state verification

As you can see, implementing rollback capability in an NC environment is not a trivial matter, and avoiding rollbacks, if possible, is preferred.

4.6 Session Context

As users interact with a software application (Web-based or otherwise), they expect the application to progress in a logical fashion that maximizes user efficiency and minimizes wasted time. At any given time, the sum of the users' actions in running the applications up to that time have put the application into a certain state. This combination of user actions and current application state is session context.

Applications should empower the user. In this regard, the user expectations are the same when running on an NC or a PC. Are there any problems meeting these expectations on an NC?

4.6.1 Issue: Providing Session Context for NC-Based Applications

Web-based applications and data are frequently accessed by browser software (for example, Netscape Navigator or Internet Explorer). If a browser user is willing to type in URL locations, then that user can choose to directly jump from one site to another without any thread of logic connecting the jumps between sites.

Usually, for an application to help the user along, and to present data to the user in an organized fashion, the designers of the application will choose to

track the user's actions and to anticipate logical user responses to certain situations.

When something interrupts a user session (loss of network, local node failure, and so on) a typical user may expect to resume work where it was left off before the interruption. However, NCs generally do not resume an application after a power failure, since everything must be loaded from the server, including the boot files. They can be configured to automatically load and start a particular application whenever they are turned on (which would start an automatic application download as part of the NC bootup).

Usually, session context is maintained by saving it to the local hard disk during a user session. NCs can't do this because they have no hard disk. In many ways, this session context issue is just a more complicated variation of the transaction issue described in Chapter 4.5.1, "Issue: Transaction Rollback in a Diskless Environment" on page 57. Again, if a session context is required, this must be saved at a server, which means that the developer must take this into consideration when developing the Java application.

4.6.1.1 Cookies: A Common, but Unacceptable Solution

As Web-based applications became more interactive, using HTTP with CGI allowed developers to create a more interactive environment. CGI let Web servers connect your client applet with back-end server-based applications that could perform some needed task and return information regarding the results.

But still, HTTP and CGI are stateless environments. The server essentially sends the response and then "forgets" about the client as it gets ready to respond to any request from any client out on the Web. How can we maintain some semblance of session context? One solution was to have the clients maintain some knowledge of their current state and to pass that state information to the server on each communication via cookies.

A cookie is an invisible piece of data maintained by the client and passed to the server to help the server identify pertinent information about the particular client that sent the cookie. The cookie contents influence the actions that the server software will take. The user may never see the cookie. There are other "hidden field" variations on this approach that hand data back and forth (for a discussion of this, refer to Orfali, Robert and Harkey, Dan, *Client/Server Programming with Java and CORBA*, 2nd Edition, John Wiley and Sons, Inc., New York, NY, 1997, pp. 36-37).

This type of approach bothers some users because they have security concerns about their personal data being sent around the Web. Further, some

cookies are stored on the client's hard disk to be used the next time that particular client application is started. Some users may have concerns about such data being stored on their machine. Also, the NC doesn't have a fixed disk to store the cookie locally.

Cookies could also be server-based. Upon initial contact, the server could attempt to identify a client and see if the server already holds a cookie for the client from some previous session. The task of correctly identifying the client is up to the implementation of the developer. Cookies are not a preferable solution for providing session context.

4.6.1.2 Principle: Verify and Validate Current Session Context

In addition to adapting some of the transaction-related alternative ideas outlined in Chapter 4.5.1.1, "Principle: Don't Depend on Rollbacks for NCs" on page 58, NC-based applications can help maintain session context by implementing the following recommendations:

1. Have more user verification.

   This approach puts some responsibility back onto the user to detect a problem (or at least observe a warning message and respond to it). The application needs to "train" the user to frequently respond to prompts and verify each step as correct before the application executes it. This attempt to focus the user makes it easier for him/her to remember and re-enter data if a failure occurs. The results of this approach, of course, will vary by user.

2. Use more data validation tests.

   If data are validated more often, it can be possible to create a larger number of small transactions instead of one large one. More frequent transaction commitments means less loss of data and user effort if a failure occurs.

3. Inform user of session context interruption.

   Maintain a simple binary session state (good or bad) in the external server that launched the application. Have the client application automatically send an "I'm OK" message periodically to the server. If the server doesn't receive the next message before the time-out period expires, then the server will reload and restart the application, including displaying a prominent message to the user that there has been a restart.

   Note that this approach will require having the client run a separate timer-based "I'm OK" thread to keep updating the server. This thread is separate from any thread(s) interacting with the user (no waiting on

user input). See Chapter 4.9.1, “Threads in Java” on page 67, for further discussion regarding potential thread issues for NC-based Java applications. The timing of the thread is implemented by the developer. Although this thread probably would not have much impact at the client, implementing "I’m OK" threads on hundreds of clients to the same server could result in some performance degradation, so this timing mechanism should be tunable by the network administrator.

4.6.1.3 Principle: Practice User Anticipation

Data and application byte code is loaded across the network to the NC. On slow networks and busy servers, users easily become impatient. These applications represent the face of your company on the Internet. For such important situations, it is worthwhile to conduct usability testing to profile your user software and verify that it is user-friendly and intuitive to use.

Once software user interface patterns are determined, anticipate user actions by preloading the most frequently used program parts. This may require a separate thread to keep the download going while the user is looking at some other part of the application. See Chapter 4.9.1, “Threads in Java” on page 67 for additional design considerations when using threads.

Recall that NCs have no concept of memory page files. Obviously, RAM limitations affect how much can be downloaded. One can imagine a program and data set so large that the entire image could never be loaded. If a program is designed never to have all parts of it visited by a user in a single session, then the application should include monitoring logic to guard against memory overload.

Note: All of the principles outlined in this section also apply to Java-based applications running on PCs with limited RAM and disk, although to a lesser degree. The JVM implementation may be running on an operating system that supports paging to local disk, which mitigates many of the issues described.

4.7 Remote Object Persistence and Serialization

Object persistence lets you save away the state of a bean and then restore it at a future point in time, either during the current run of the application or at a future run.

Some people use the terms "object persistence" and "object serialization" interchangeably. Persistence can be defined as the saving and recovery of an object’s state. This is an abstract concept, which could be implemented in a

number of ways. We define serialization as the process of converting the state of an object into a data stream.

In Java, serialization is one mechanism used to achieve persistence. The Java Remote Method Invocation (RMI) API uses serialization to pass state information across Java programs or over a network. A client application running on one node of a network invokes a method on an object that is resident on a different network node. So, how do these concepts work on NCs?

4.7.1 Issue: JavaBean Serialization vs. Externalization

If your JavaBean is going to be persistent, it must implement either the serialization or externalization interface. However, serialization typically uses persistent storage, which is usually a local disk. Since NCs, as we have defined them, don't have hard disks, we can't use the default serialization mechanism.

VisualAge for Java, for example, does all the code generation work for the programmer who uses the Serialization interface. Of course, you could always have your JavaBean override Serialization's default `readObject` and `writeObject` methods by creating private versions of them, but then you would have to do something meaningful with the bean state information (for example, write it out to a remote disk drive somewhere else on the network, so that it can be read back at a later time).

We can also use the Externalization interface. As Java developers, we must explicitly implement Externalization's `readExternal` and `writeExternal` methods. For the Java NC programmer, the differences between overriding Serialization's `readObject` and `writeObject` and implementing Externalization's `readExternal` and `writeExternal` are fairly negligible (except for security, see Chapter 4.7.1.1, "Principle: Know Your Security Model" on page 63).

4.7.1.1 Principle: Know Your Security Model

The Serializable interface's `writeObject` is private. Security-related data (for example, my annual salary, WWTC's contract data, and so forth) is more easily protected.

The Externalizable interface's `writeExternal` is a public method, so bean state information can be extracted by anybody's program that successfully invokes `writeExternal` on my sensitive bean. In general, sensitive classes should not be serialized at all.

4.7.1.2 Event State Persistence Limitation

The following discussion is based on the JDK structure used in JDK 1.1.

JavaBeans can be programmed to notify registered listeners when the bean fires an event. To fire an event, the event must be defined. You can use any of the AWT-specific event object types or create your own event type by extending `Java.util.EventObject`.

In either case, because `EventObject` implements the Java Serializable interface, Java events can be stored persistently. But, not on an NC, which has no local persistent storage.

4.7.2 Can NC Applications Use JavaBeans and Generate Events?

Designing and developing for the NC may be a new experience for the reader. Often, when working in a new (and often poorly documented) environment, the designer needs to know if previously learned techniques are still valid.

4.7.2.1 JavaBean Use in Applets Running on the NC

According to Laurence Vanhelsuwe, author of *Mastering JavaBeans* (see reference in Appendix E.3, “Other Publications” on page 143), "A bean lives in a multiple threaded environment and can therefore be addressed by multiple threads at the same time."

When designing JavaBeans for use in applets that run on an NC, the issues associated with running multiple threads on an NC should be examined. See Chapter 4.9.1, “Threads in Java” on page 67 for further details.

Basic testing done during the development of this document proved that JavaBeans can successfully be used in applets and applications that run on an NC. JavaBeans generate events, and this functionality works fine on the NC.

4.7.2.2 Principle: Distinguish between Applets and Applications

The designer for NC-based software must clearly distinguish between NC-based applets and NC client applications. An NC can run multiple applets simultaneously, but may run only one stand-alone application at any particular point in time.

Ordinarily, applets running on an NC can only talk to the server that downloaded them. Applets, unlike applications, cannot listen for incoming TCP/IP connections coming from remote objects across the network. IBM WebSphere includes CORBA ORB support that allows the remote server

application to invoke callbacks on the client applet. For the NC programmer, this creates a more robust environment.

4.8 Concurrency

Concurrency addresses issues that arise when multiple simultaneously running agents, producers and consumers, try to access the same resource.

4.8.1 Issue: Race Conditions

When there is more than one thread or agent trying to access the same shared resource, what is referred to as a race condition can occur. Race conditions are situations where two or more threads access a resource, and each thread thinks it is the only entity trying to inspect or change this resource. Since both threads update the resource, the actual outcome, that is, the new state of the resource, is unpredictable because the timing of the changes was not controlled by the programmer.

The following discussion assumes that the addition operation in a computer is not an atomic operation, but can be interrupted. Furthermore, it is assumed that there is no built-in regulation of a resource we will call R. In other words, any thread can access R regardless of the state of other threads.

The following discussion focuses exclusively on threads, but is equally applicable in theory to processes. In fact, more solutions have been developed over the years for process concurrency control than for thread concurrency control.

An example of a non-operating system-specific process concurrency control mechanism is the CORBA Concurrency Services that can be used to organize and regulate access to defined resources.

4.8.1.1 Scenario: Concurrent Access of a Common Resource

Consider the situation of two threads: T^p is a producer and T^c is a consumer. T^p simply counts upwards sequentially and puts the result of the counting into resource R. T^c gets the count and displays it. Therefore, both concurrently share a common resource R, where T^p puts the results of its processing into R and T^c gets the input for its processing from R.

If T^p is faster than T^c and produces numbers very quickly, then the input that the slower T^c gets from R will not appear continuous. If T^p executes twice, say

counting 2+1=3 and 3+1=4, in the time T^c executes only once, then T^c will get 2 and 4 but will miss 3. This is illustrated in the following example.

Time: t=	initial	1	2	3	4	5	6	7
process:		T^p	T^c	T^p	T^c	T^p	T^p	T^c
R count:	0	1	**1**	2	**2**	3	4	**4**

Output from T^c = 1,2,4

The converse problem occurs when T^c is faster than T^p, which might look like the next example.

Time: t=	initial	1	2	3	4	5	6	7
process:		T^p	T^c	T^p	T^c	T^c	T^p	T^c
R count:	0	1	**1**	2	**2**	**2**	3	**3**

Output from T^c = 1,2,2,3

In Java, a race condition can occur when T^p and T^c are independent threads, but Java provides some specific help (see Chapter 4.9.1, “Threads in Java” on page 67). But in the NC environment, these threads might be independent agents running in separate processes on different platforms.

In order to solve the situation of a race condition, we need a mechanism to ensure that updates to shared information are coordinated, or synchronized.

4.8.1.2 Principle: Concurrency Control

The key to solving the problem of multiple processes, or threads, competing for the same set of limited resource requires an understanding of two key concepts:

Locking

Locking allows only one process access to a particular resource at one time. If T^p places a lock on R, then T^c cannot read the contents of R. Conversely if T^c is reading the contents of R when it is locked, then T^p cannot update R.

Locking is sufficient to stop corruption of the shared resource that results from both processes accessing the resource at the same time, but it may also result in one process waiting a long time, called a busy wait, check the value of the lock continually until the value changes. In this case, the CPU is being used unnecessarily and can result in performance degradation of that particular task what is waiting.

Wait and Notify

Wait makes processes that want access to the locked common resource R wait until the resource is free. Typically, these waiting processes are organized into some type of FIFO or prioritized queue. Notify tells the first eligible waiting resource that the lock has been removed and access to the common resource is permitted for that first process.

Locking, when used along with Wait and Notify, is sufficient to bring an element of control to the race condition problem. The first process in, say T^p, locks R while updating it. If T^c wants to read the resource during the time the lock is in place on R, it is forced to wait. When T^p is finished updating R, it removes the lock from R and this results in a notification to proceed for the still waiting T^c. Once the resource is released by T^p, T^c then places its lock on R while it reads the contents secure from interference by T^p or any other waiting processes in the queue.

Together, the wait and notify mechanisms have the effect of synchronizing access to R by T^p and T^c so that output from T^c will be a continuous series of integers.

4.9 Distributed Object Synchronization

This section describes mechanisms that can be implemented to address synchronization issues that may arise during the course of Java application design and development.

4.9.1 Threads in Java

Too often in the past, thread control was relegated to the programmer, who had to code thread cooperation using operating system calls (start thread, kill thread, suspend thread, and so on) and system resources (such as mutex semaphores). Java's built-in thread support makes for the happier prospect of a more standard set of approaches to thread handling. Programmers will use the thread-handling paradigms suggested by the Java language capabilities, so their recognition factor will be higher when encountering the same problem again in the future.

4.9.1.1 Synchronization: A Java Solution to Concurrency Problems

A possible Java-based approach to solving the producer consumer problem (outlined in Chapter 4.8.1.1, "Scenario: Concurrent Access of a Common Resource" on page 65) is described in the following paragraphs.

Locking

Locking is implemented in Java by specifying that methods which access a common resource can be strictly controlled by using the `synchronized` keyword. When a method is defined with the synchronized keyword, then the JVM first places an exclusive "lock" on the method before allowing one and only one thread to enter and execute the synchronized code at any single instance of time.

When methods in a non-static class are defined with the `synchronized` keyword, the JVM will obtain a lock on the object itself. However, if methods in a static class are synchronized, then the JVM will obtain a lock on the class itself. Any thread attempting to gain access to a locked method will be blocked until the currently running thread completes execution of the method and the lock is released.

By using the `synchronized` keyword, only one thread executing a synchronized method has access to the object's instance variables at one time. Therefore, in our example code (see Appendix C.2, "Synchronized" on page 137), when `get()` and `put()` are synchronized in R, only one method can access R's instance variables at any one time.

Wait and Notify

The `wait()` method waits to be notified by another thread that a change in a monitored object has taken place. This notification is made using the `notify()` or `notifyAll()` methods. The `notify()` method notifies an arbitrary waiting thread that the lock has been removed and the awakened thread gets the lock. The `notifyall()` method notifies all waiting threads and they compete for the lock. One thread get it, and the other threads go on waiting.

In our simple example, because there are only the producer and consumer threads, the instance variable `available`, specified in the synchronization code in Appendix C.2.1, "R - The Resource" on page 137, has the effect of serializing the execution of T^p and T^c. T^c can only get access to the R object after T^p has finished updating it exactly once.

4.9.1.2 Solution in Distributed Object Environments

The ability to synchronize threads is a necessity in situations where data are accessed and updated by several different threads. But, in a distributed object environment, the threads may be physically separated and running on different platforms. Therefore, the synchronization mechanism must work in this distributed environment.

Synchronization in a distributed environment is provided by ensuring that the methods that provide access to the resource are synchronized. Since these

methods provide the only means of accessing the data, doing a local synchronization ensures that the clients, whether local threads or external agents, will only have synchronous access to the data.

In the example, the `get()` and the `put()` methods are both synchronized, so regardless on what platform T^p and T^c reside on, their access to the R object will be controlled because synchronization is local to R.

4.9.1.3 JavaBeans and Threads

The thread synchronization issues described in this section also apply to JavaBeans. Since the developer of a JavaBean may not know the environment in which the Bean is being used, one should assume that the Bean will be utilized in a multiple thread environment. In such a case, the JavaBean developer must take special care when coding methods by synchronizing data manipulation, reads and writes. It is also important for these synchronization mechanisms to be as short as possible so that any wait time by other threads is minimized.

For more information on threads and Java, refer to *Java Threads*, referenced in Appendix E.3, “Other Publications” on page 143.

4.9.2 JavaBean Synchronization

JavaBeans come from different vendors. CORBA’s Transaction Service support can be used to coordinate (synchronize) the actions of multiple beans executing simultaneously.

Chapter 5. Designing, Developing, and Distributing Java Objects

This chapter describes methods and approaches for the design, development, and distribution of e-business applications that use Java-based objects in a distributed object architecture. This chapter also describes how certain types of Java objects behave in a distributed environment.

5.1 Distributed Application Environment

The NCF model for an e-business solution is to use an n-tier approach (the NCF model was described in detail in Chapter 2, “Network Computing Framework” on page 21). In typical three-tier approach, the first tier is the client, the middle tier is the server and the final tier is an enterprise system. Since these tiers are usually running in separate processes on different machines, this creates a distributed application environment.

5.1.1 Roles and Responsibilities

Unlike monolithic applications, which reside entirely on one system, distributed applications have responsibilities for the system divided among the parts. The client is responsible for the presentation of data, the server is responsible for the execution of business logic and the enterprise system is responsible for the storage of data.

5.2 Implementing the NCF Model Using the Java Language

The following subsections provide an overview of the JDK contents and a high-level description of Java API categories. We also describe some issues encountered when coding the application example described in Chapter 3, “Application Example - WorldWide Trucking Company” on page 43.

An NCF client implementation in Java can be implemented using applets. An NCF server implementation in Java can be implemented using servlets. In addition, leveraging the wealth of Java APIs and JavaBeans will be a key to rapid development. In developing solutions, you will need to consider the supported features that are core to a Java release and be aware of the features that are external to the core but are available as standalone APIs. Portability does equal compatibility. Many of the development and run-time products available at the time of this writing have varying degrees of Java run-time Version 1.1.x support. However, with each new major release, more and more APIs are finding their way into the Core Java platform.

5.2.1 Java Development Kit

This section describes currency and function related to the Java Development Kit.

5.2.1.1 JDK 1.1.6

As of this writing, JDK Version 1.1.6 is the current major release. It is a superset of JDK 1.0.2 and includes support for Internationalization, Security, JavaBeans, JAR, RMI, Object Serialization, JDBC, Inner Classes, JNI and enhancements to core AWT, IO, Net and Math.

5.2.1.2 JDK 1.2

As of this writing, JDK Version 1.2 is still in the third beta release. It adds some major functionality in the areas such as Java 2D, Drag and Drop, Application Services, Extension Framework, Weak References, Java IDL, a new JVM Debugger Interface (JVMDI), the Java Servlet standard extension and Javadoc Doclets.

5.2.1.3 Java Foundation Classes (JFC)

As of this writing, JFC Version 1.1, which includes Swing Version 1.0.3, is the current major release, although Swing Version 1.1 beta 2 is available with the JDK 1.2 Beta. The JFC extends the original Abstract Windowing Toolkit (AWT) by adding a 100% Java implementation of the GUI components. Some features of this API are included in JDK 1.2.

5.2.1.4 Java Native Interface (JNI)

Although Java is a very functional and flexible language, there are some situations where other languages, such as C, C++ or Assembly, may be more appropriate, such as in time-critical routines or platform-dependent functions. The JNI allows these native methods to interact with Java classes and methods, extending the reach of existing programming into the Java world. Although there were native interfaces available in earlier releases of the JDK, JNI will be the focus as other implementations are phased out over time.

5.2.1.5 Java Accessibility

The Accessibility API provides assistive technologies to interact and communicate with JFC and AWT components. Assistive technologies are used by people with and without disabilities and include screen readers, screen magnifiers, and speech recognition. This API will assist development efforts for solutions that are required to meet federal regulations on accessibility.

5.2.1.6 JavaBeans

Included in the JDK 1.1, the JavaBeans specification defines a set of standard component APIs that enable developers to write reusable components. A key aspect of JavaBeans is that it allows developers to create software components that can be assembled together using visual application builder tools.

5.2.1.7 Java Security

This API includes a framework for developers to include security functionality in applets and applications. This includes features for cryptography with digital signatures, encryption and authentication. Some features are included in JDK 1.1, while others, such as the Java Cryptography Extension (JCE), are usable with JDK 1.2 but are only exportable to the US and Canada due to United States Government cryptography export restrictions.

5.2.2 Java Enterprise APIs

These APIs are geared toward the robust e-business solutions that the NCF model supports.

5.2.2.1 Enterprise JavaBeans

At the time of this writing, the Enterprise JavaBeans Version 1.0 component architecture is available. This technology should make it easy for developers to create reusable JavaBeans that integrate with their existing system services and applications. This is a key technology in developing e-business solutions.

5.2.2.2 Java Naming and Directory Interface (JNDI)

The JNDI provides uniform, industry-standard connectivity from the Java platform to business information assets, allowing developers to incorporate access to naming and directory services.

5.2.2.3 Java Interface Definition Language (IDL)

This package implements Java classes and tools based on the industry-standard CORBA specification for object-to-object communication over the network. This includes an IDL-to-Java compiler and a lightweight ORB that supports IIOP. This is a key technology for bootstrapping an ORB onto an NCF client, such as the IBM Network Station. This will be included as a standard extension to the JDK 1.2 release.

5.2.2.4 JDBC Database Access

This Java DataBase Connectivity API is included in JDK 1.1 and provides access to object or relational databases from the Java environment using a

standard interface. IBM and other vendors are providing higher level tools built on this API which generate database access classes and JavaBeans based on the physical schema of your database.

5.2.2.5 Java Remote Method Invocation (RMI)

This API is included in JDK 1.1 and provides a method for creating Java objects whose methods can be invoked remotely by other Java objects. This API also includes Object Serialization, which allows an object to be stringified into a stream of bytes and later reconstituted as an object somewhere else.

5.2.2.6 Java Messaging Service (JMS)

This API addresses the need for asynchronous enterprise services such as message queues, publish and subscribe services, and push/pull technologies. Developers will be able to write middle-tier Java code that connects to a messaging agent that provides facilities to create, send or receive enterprise system messages.

5.2.2.7 Java Transaction Service (JTS)

This API is a standard extension that defines an open standard for transaction management for the Java platform. Due to the many vendors that provide transactional applications, resource managers and transaction monitors, this API provides a mapping to the OMG Object Transaction Services (OTS) and the X/Open XA interface.

5.2.3 JavaBean Communication

JavaBeans can communicate with each other using a variety of methods:

- On the same node using Lotus InfoBus technology (described briefly in Chapter 2.4.9, “e-business Application Services” on page 33)
- Direct use of sockets
- Remote Method Invocation (RMI)
- Across the network using CORBA’s IIOP

5.2.4 Applets

From a Java application developer’s viewpoint, Applets are fun. They were previously defined in Chapter 1.7.2, “Java Applets” on page 13). Like mini-applications, the applet can draw graphics, load and display images, manage GUI components and handle user events, and do most of the things their older brother applications can do. Applets are untrusted code and are distinguished from Java applications by a long list of security restrictions (described in Chapter 1.7.2, “Java Applets” on page 13). However, they are

extremely important because they do have access to most of the rich services implemented as Java APIs, and can therefore support a serious e-business solution on a thin client tier.

5.2.4.1 Development

Applets are implemented as Java classes that have to be written in Java and then compiled. For starters, the JDK defines a base Applet class in the java.applet package. To write an applet, a developer needs to create a subclass of the base Applet class and override several methods. Applets do not define a `main()` method but instead must rely on a set of methods to be invoked by the applet context, such as `init()`, `start()`, `stop()` and `destroy()`. The `main()` method, specified for standalone applications, tells the interpreter where to begin execution. (However, applets that are created as visual beans by IBM VisualAge for Java do have a `main()` method.)

Visual composition tools are very useful for developing the GUI and scripting aspect of the applet using a drag-and-drop approach where parts or JavaBeans are dropped onto a work space and connected to each other with configurable links. There are numerous books and tutorials on this subject, and so the details of how to write a Java applet will not be discussed here, although we will cover some specific details for the WWTC applications.

5.2.4.2 Applets as JavaBeans

In the following discussion, we will refer to the WWTC sample application in Chapter 3.3, “Transport Request Application” on page 46. You should review Figure 10 on page 47 for applet appearance details.

IBM’s VisualAge for Java (VA Java) was used to create the applet bean(see Figure 13 on page 76), which was called RequestCargoPickup. Because we elected to create the applet bean visually, and defined bean properties, we ended up creating a runnable bean with a main.

The running man icon (shown in Figure 14 on page 77) shows that RequestCargoPickup is also an applet. So, an entire applet was implemented as a JavaBean.

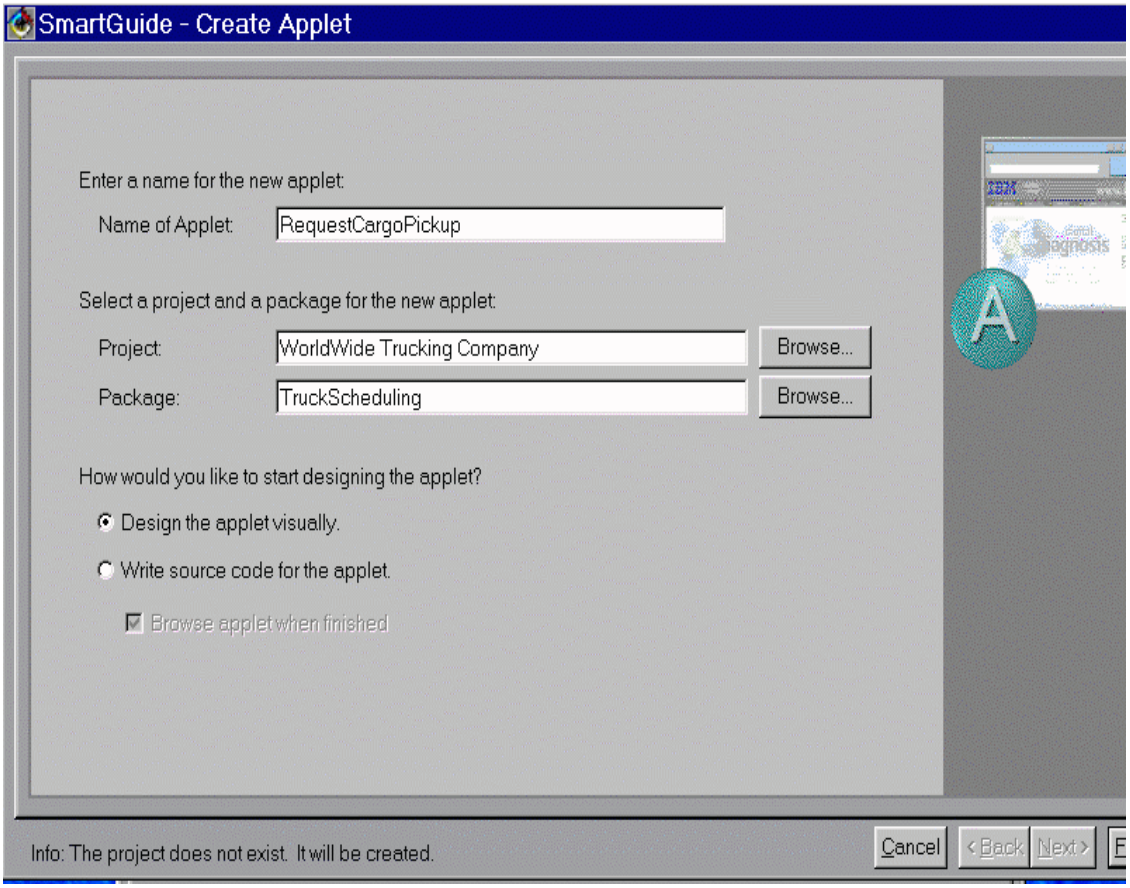


Figure 13. Initial Creation of Truck Cargo Pickup Request Applet

The fact that the `RequestCargoPickup` bean is an applet is easily determined by noting its jigsaw puzzle piece icon (see the Classes and Interfaces column in Figure 14 on page 77). The jigsaw icon also means that `RequestCargoPickup` is a visual bean in VisualAge for Java.

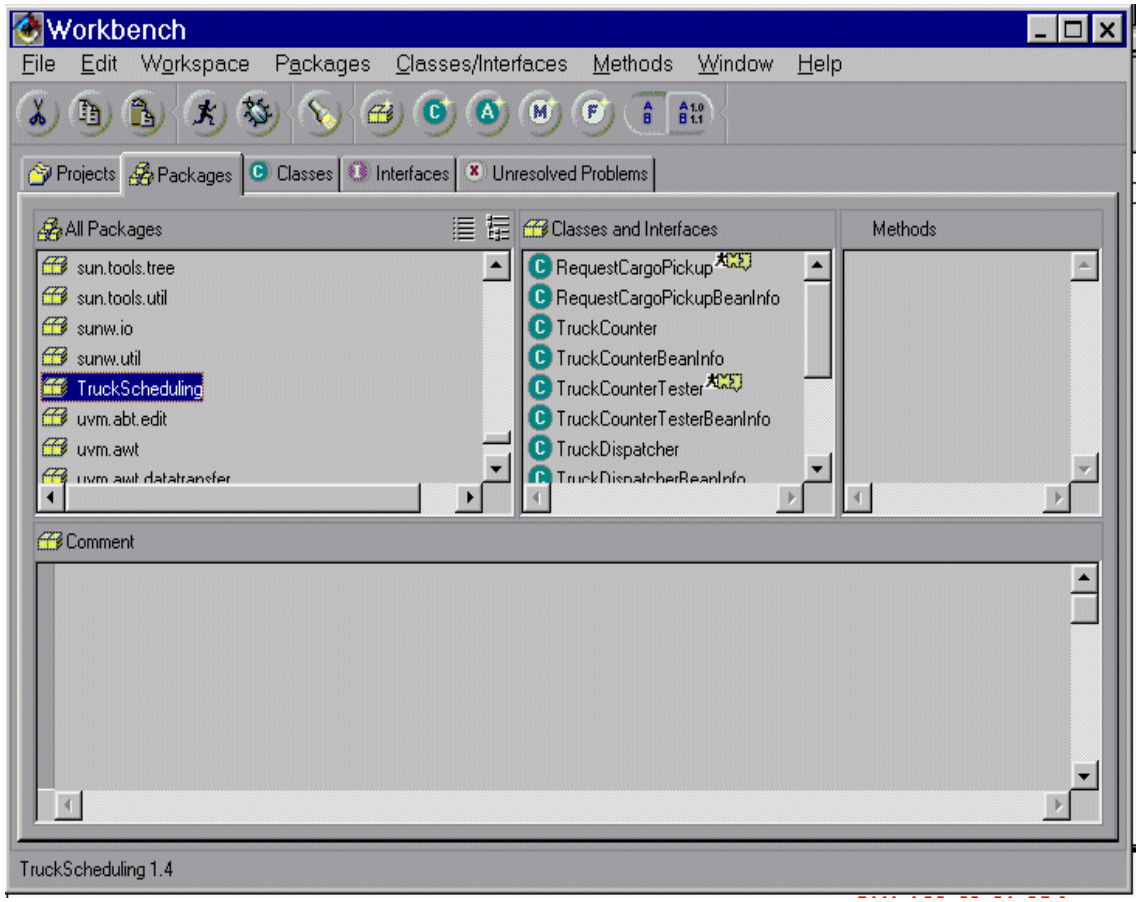


Figure 14. TruckScheduling Package's Beans and Classes

Note the `RequestCargoPickupBeanInfo` class in Figure 14. VA Java generated that class for us automatically the first time we created a bean property feature (`NumberOfTrucksRequested`) using the BeanInfo tab while working in the Visual Composition Editor on the `RequestCargoPickup` bean (see Figure 15 on page 78). VisualAge automatically creates a BeanInfo class for your bean when you add the first new feature.

Note the `B` for Bound indicator on the `NumberOfTrucksRequested` property. Bound properties send value changes on connections. In our example (see Figure 16 on page 80), the property is used as input to a method executed when the user presses the **OK** pushbutton. In the Visual Composition Editor, the bound property connection appears a a purple wavy line.

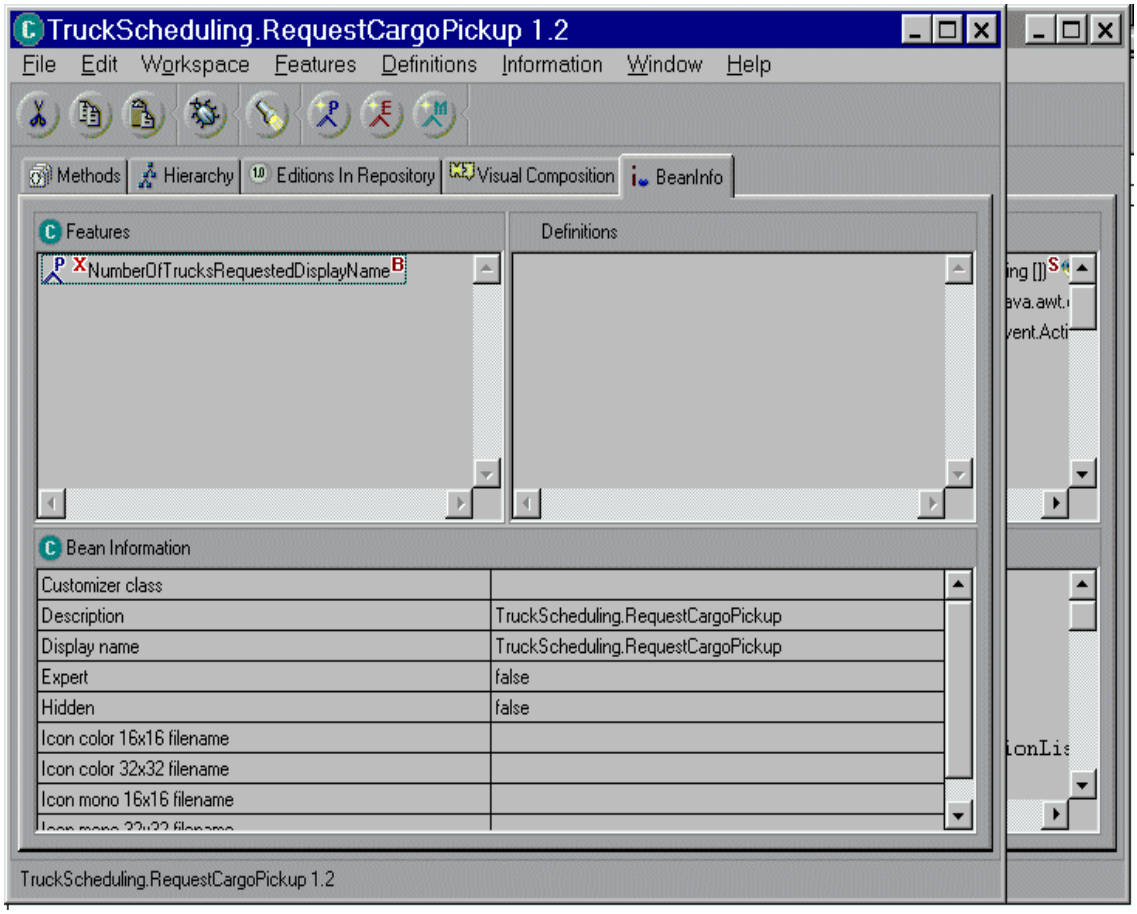


Figure 15. Result of Automatic Bean Creation

Now, examine the appearance of the applet itself when open in VA Java's Visual Composition Editor. Note that there is a non-visual bean (class name is `TruckDispatcher`) called TruckScheduler1 that appears on the far right-hand of the screen. (The bean identifier may not be clear because of image clipping.) The various data fields (Requested Pickup Date and Time, Pickup Address, and so on) are visual beans supplied by VA Java. When working with VA Java in the Visual Composition Editor, everything you drag and drop is a bean.

The `TruckDispatcher` class exists to function as the interface to WWTC's remote truck dispatching server application known as `WorldWideTruckingDispatcher`. The relationship between our applet and the

remote server will be discussed in more detail in Chapter 5.2.5.2, “Servlets as JavaBeans” on page 88.

Note also that we used VA Java’s capabilities to add our own category (a filed folder) of beans in the left column of the bean palette. You may notice in Figure 16 on page 80 that choosing our custom bean folder caused the folder name (Tim’s Beans) to appear in the VA Java message area in the lower left corner of the Composition Editor window. In retrospect, it would have been more appropriate to name the folder as WWTC’s Beans.

5.2.4.3 Considerations for Applet Programming

The main problem found when running this applet on the IBM Browser and the Navio Browser on the Network Station 1000 was that the canvas and bean appearances differed from that displayed when the applet was run on an IBM PC 350 desktop system. (Figure 10 on page 47 and Figure 11 on page 48 are screen shots of the applet executing on a PC.) On our NC, visual problems encountered included:

- Overlap of data fields onto label fields
- Different font size and type from what was developed in the VA Java environment
- Clipping at the right hand edge of the right-most beans

For the bean clipping problem, the right edge of all the push buttons was not visible. Although the browser used to display the applet on the NC allowed the user to widen the window of the applet, that did not restore the clipped portion of the image, and it did not relieve the overlapped field problem.

The only solution that yielded any improvement was to space out the fields much wider when building the original screen in VA Java. However, this action did not completely alleviate the problem.

Note: This clipping problem is not specific to NC’s; visual programming problems like this can appear on any platform, including Windows and UNIX.

This situation illustrates the design principle to always test your applet fully on your target Java environment, NC or otherwise. Since the target JVM is responsible for controlling the screen display, an applet should be tested on each type of hardware and JVM implementation likely to be found in the user’s environment.

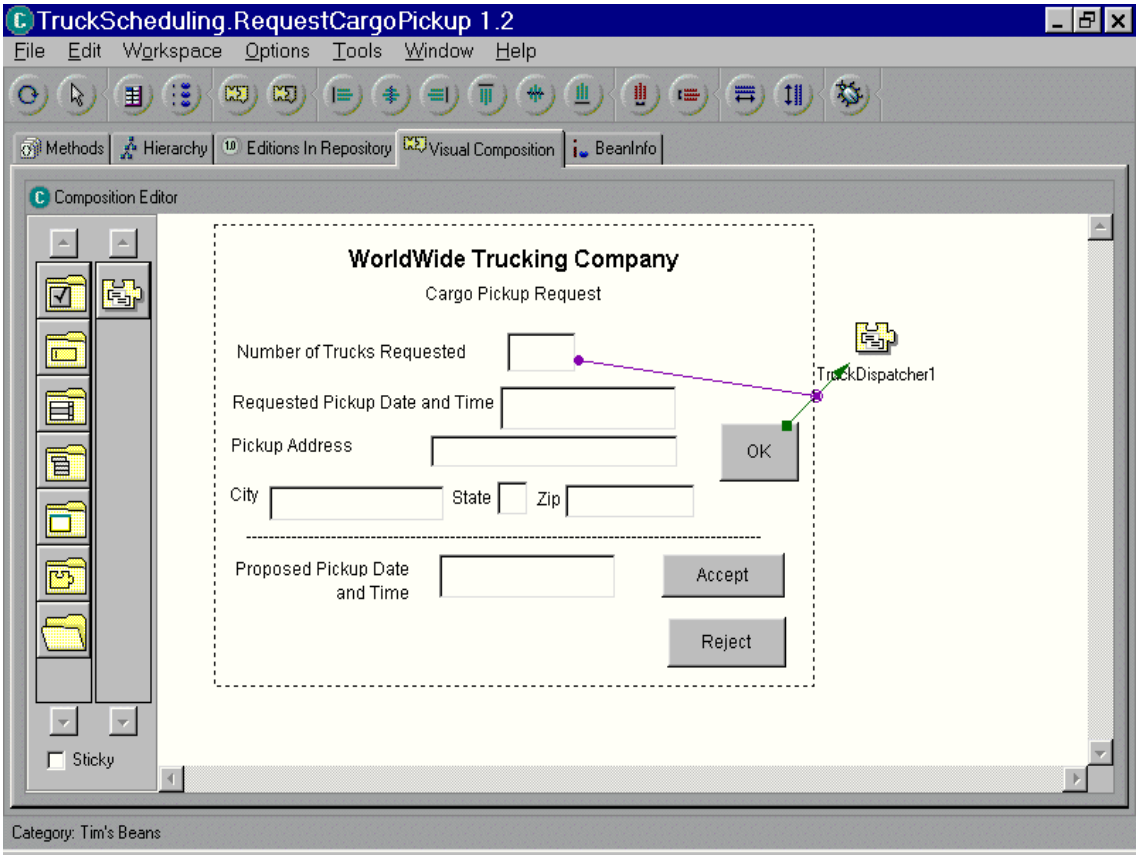


Figure 16. TruckSchedulingRequest Applet's Internal Visual Components

When the **OK** button is pressed by the user, the parameters that have been filled in are sent to the TruckDispatcher bean, which is then executed. In our particular case, the number of requested trucks is sent as a parameter to the `TruckDispatcher.RequestingCargoPickup` method, which is then executed.

To summarize, the steps required to create an applet and associated beans are:

1. Create a project and package to hold the applet and bean(s).
2. Create the applet, defining it as a visual, runnable JavaBean.
3. Layout the visual appearance of the applet, including dropping visual and non-visual beans on the canvas.
4. Link the beans together to perform the desired actions.

Note that nothing about the initial applet and initial bean creation process was altered because of the use of an NC as the applet's eventual run-time target. As mentioned earlier, some visual layout changes were made later to deal with runtime display problems.

5.2.4.4 Sample Applet Code

JAR File Available

The JAR file for the customer client-side applet is available for external download. Go to the external IBM Redbooks site, `http://www.redbooks.ibm.com` and click on **Additional Materials** in the left-hand frame, underneath the Products category. When the list of publications appears, scroll down and select SG245111. You will see the WWTCJAR file in a ZIP format.

- RequestCargoPickup Class

  This is the main applet and visual bean that was created using VA Java's SmartGuide (also known as the Quick Start window). The main applet declaration is shown in Figure 17 on page 81. (Many repetitive declarations involving multiple labels, buttons, and text fields were deleted from the excerpt for clarity.)

  Note the declaration of an instance of the `TruckDispatcher` non-visual bean as a private attribute of the class.

  VA Java generated all of the code for this class. No programmer changes were necessary.

```
/**
 * This applet was generated by a SmartGuide.
 *
 */
public class RequestCargoPickup extends Applet implements
java.awt.event.ActionListener {
private Button ivjButton1 = null;
private Label ivjLabel1 = null;
private TextField ivjTextField1 = null;
private TruckDispatcher ivjTruckDispatcher1 = null;
}
```

Figure 17. Sample Applet Code

- TruckDispatcher Class

  This class is the main interface to the remote Truck Dispatcher server. It was created as a non-visual bean.

  VA Java generated a code stub for this class when we initially created it using SmartGuide. We made it into a non-visual bean when we defined the requestingCargoPickup method for the bean (see Figure 18 on page 83).

  For simplicity, we didn't define any button and data field activity back on our applet that were related to pickup dates and so forth. For illustration purposes, we wanted to keep Figure 16 on page 80 simple and uncluttered. So, we defined those values ourselves with some simple code lines (see the try block in Figure 19 on page 84).

  Note that Figure 19 doesn't show the method's code related to the remote implementation of the dispatcher. That is covered in Chapter 5.3.1.5, "JavaBeans and RMI" on page 98.

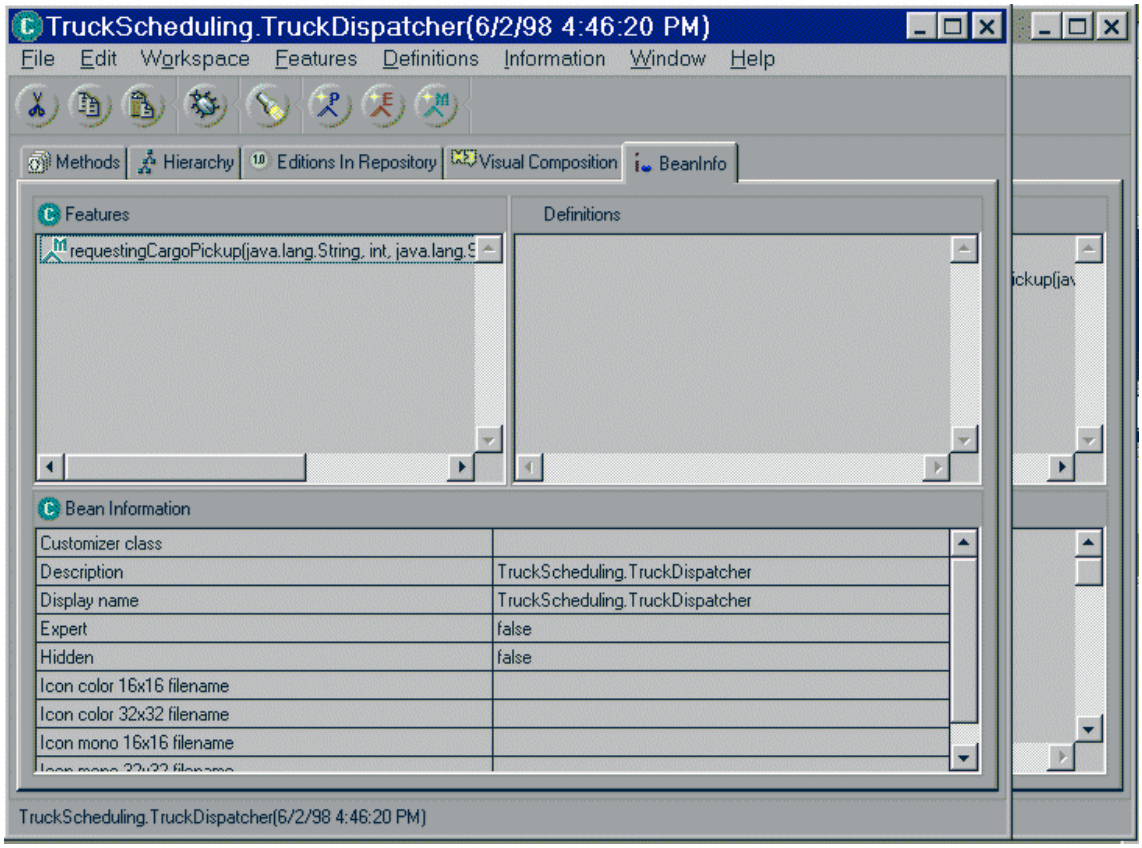


Figure 18. Non-Visual TruckDispatcher Bean Method Definition

```
public boolean requestingCargoPickup(String customerName, int
truckQuantity, String pickupAddress, java.util.Date pickupDateTime) {
  boolean approvalStatus = false;
try {
  customerName = "Austin ITSO";
  pickupAddress = "11400 Burnet Rd., Austin, TX  78759-3493";
  // Get the current time.
  pickupDateTime  = new Date();


return approvalStatus;
}
```

Figure 19. Simple TruckDispatcher Code

That's it, just two important classes. `RequestCargoPickup` is the main applet and bean, and it uses TruckDispatcher non-visual bean. From some viewpoints, TruckDispatcher provides a server function, because it calculates the approval status for the customer pickup request. We'll develop and extend this bean into a true server later in this document.

5.2.4.5 Run Time

Applets are run within an applet context, such as a Web browser or applet viewer. Therefore, applets can execute on any platform that supports a Java applet context. Many of the applet screen shots shown in this book (for example, Figure 10 on page 47, and Figure 11 on page 48) show our trucking applet running in VA Java's applet viewer (the Run function of the Workbench).

Applets satisfy the Just-In-Time (JIT) delivery method. A Web browser loads and runs an applet when it encounters the HTML `<APPLET>` tag embedded in a Web page that specifies the .class file to retrieve from the Web server. Physically, the applet is running within the JVM process of the NC, not in the JVM process of the Web server from which it came. This aspect differs from X-terminal applications which are displayed on in X-terminal on the IBM Network station but are actually running within a process on a server.

If we wanted to start this applet manually on an NC, we could use the `appletviewer` program to start it from a command line, or we could go to the browser to open a URL and type:

```
http://oc0263f.itsc.austin.ibm.com:8080/redbook/TruckCargoPickupRequest.ht
ml
```

For the above command to work, the applet has to be stored on a network node that has Web server software (like Lotus Domino Go Webserver). In our sample command, `oc0263f` is the TCP/IP name for a particular node in the `itsc.austin.ibm.com` domain. The number 8080 represents the particular Internet port number over which the server and applet will communicate. Redbook is just a directory in which to store the `TruckCargoPickupRequest.html` file. (For completeness, the html file is actually stored in the `D:\WWW\HTML\redbook` directory. `WWW/HTML` is a directory related to the Lotus Webserver software.)

Upon receipt of the open URL request, the Web server software downloads the applet's classes to the requestor (the NC, in our case). Remember, an NC can be configured to automatically download and start an applet whenever it is turned on. So, we can actually avoid all of the typing above.

For our samples, we developed a simple Web Page that included the tag to load and start the RequestCargoPickup applet. To view the Web page, the NC user would type:

```
http://oc0263f.itsc.austin.ibm.com:8080/redbook/index.html
```

`index.html` is just the name of the file containing the Web page's HTML. There is nothing significant about the choice of the name "index" for the filename. Try building your own Web page and testing our samples at your convenience.

Although our goal is not to teach HTML programming, we thought you might like to see the contents of `TruckCargoPickupRequest.html` (refer to Figure 20 on page 85 and Figure 21 on page 86).

```
<HTML>
<HEAD>
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=iso-8859-1">
   <META NAME="Author" CONTENT="craig e grossi">
   <META NAME="GENERATOR" CONTENT="Mozilla/4.05 [en] (WinNT; I)
[Netscape]">
   <TITLE>TruckCargoPickupRequest</TITLE>
</HEAD>
```

Figure 20. HTML for Starting Cargo Pickup Request Applet: Part 1

```
<BODY>
<CENTER>
<H1>
TIM’s CARGO PICKUP REQUEST</H1></CENTER>

<CENTER><APPLET CODE="TruckScheduling.RequestCargoPickup"
codebase=http://OC0263f.itsc.austin.ibm.com:8080/ServletExpress/class
es WIDTH="400" HEIGHT="600"></APPLET></CENTER>

</BODY>
</HTML>
```

Figure 21. HTML for Starting Cargo Pickup Request Applet: Part 2

The Applet’s HTML specification looks a little different when viewed through a Web page composing tool like Netscape Composer (see Figure 22 on page 87). The composing tool is used to visually imbed a reference to the Applet’s class file(s) name and location into the Web page itself.

Note the references in both Figure 21 and Figure 22 to the "codebase" location. This will become significant later as we develop our server.

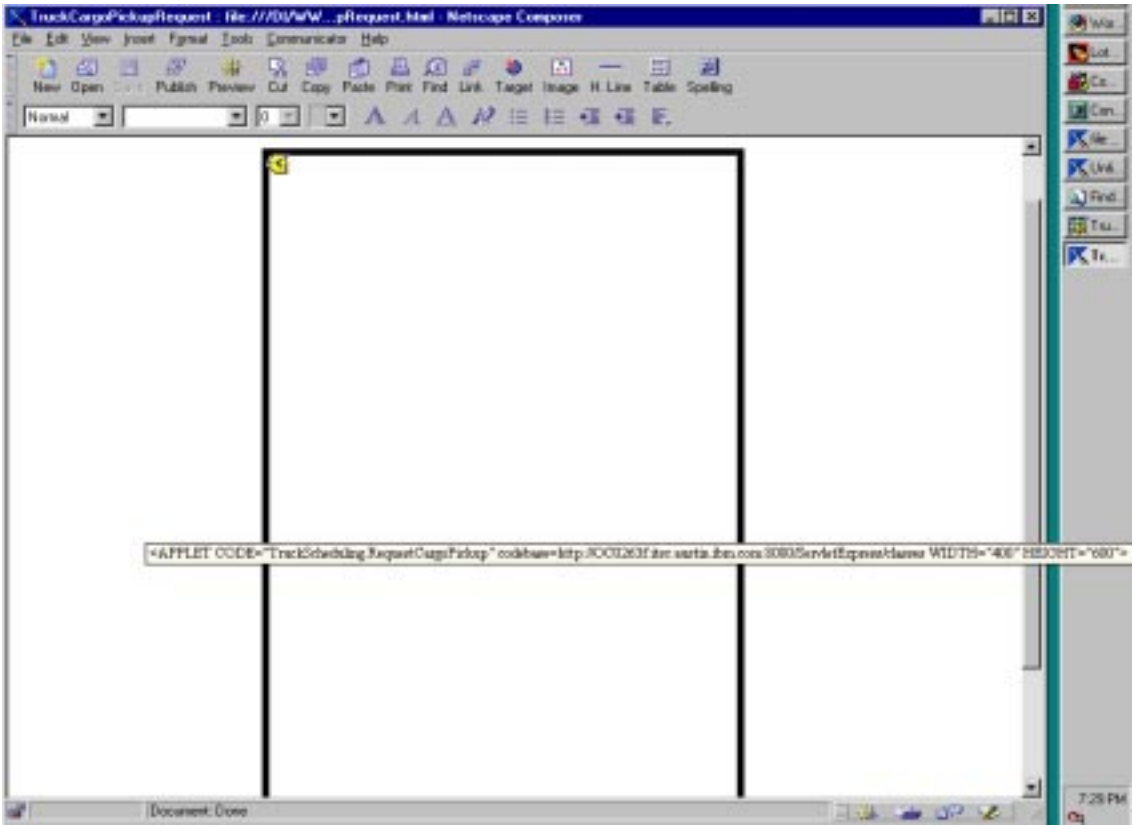


Figure 22. Sample Application's HTML Viewed by Netscape Composer

5.2.4.6 Parameters and Environment Variables

When loaded, an applet embedded within an HTML web page has access to the parameter name/value pairs specified by <PARAM> tags by using the `getParameter()` method. This is a useful way of avoiding hardcoding the system information into the compiled Java code.

5.2.4.7 Applet Execution Life Cycle

When an applet is loaded, the applet context creates an instance of the applet and invokes the `init()` method. Any initialization that should be done only once should be performed here.

Each time an applet becomes visible, the `start()` method is invoked. Applets usually then begin doing whatever they are supposed to do, often using threads. The `start()` method has to be written assuming that it will be invoked multiple times simultaneously.

When an applet is hidden, the stop() method is invoked. If threads are being used, they should be stopped. This method also has to be written assuming it will be invoked multiple times simultaneously.

When an applet is about to be unloaded, the `destroy()` method is invoked. Applets should free resources at this time.

5.2.5 Servlets

Like the servlet's counterpart, the applet, servlets are mini-application servers. Unlike applets, they often do not have a GUI aspect and are typically run as trusted code without security restrictions applied to them. Like applets, they are extremely important because they have access to all of the rich services implemented as Java APIs and can therefore support a serious e-business solution at the middle tier.

5.2.5.1 Development

Servlets are implemented as Java classes which have to be written in Java and then compiled. For starters, an extension to the JDK defines a base class named GenericServlet in the javax.servlet package. To write a servlet, a developer needs to create a subclass of the base servlet class and override several methods. Servlets that need to handle HTTP requests need to subclass the HttpServlet class in the javax.servlet.http package. Servlets do not define a `main()` method but instead must rely on a set of methods to be invoked by the servlet context such as `init()`, `service()` and `destroy()`.

Visual composition tools may be used for developing the scripting aspect of the servlet using a drag-and-drop approach where JavaBeans are dropped onto a work space and connected to each other with links.

5.2.5.2 Servlets as JavaBeans

We have already created our non-visual TruckDispatcher bean. Now we will start to build on it, eventually creating the related class structure show in Figure 23 on page 89.

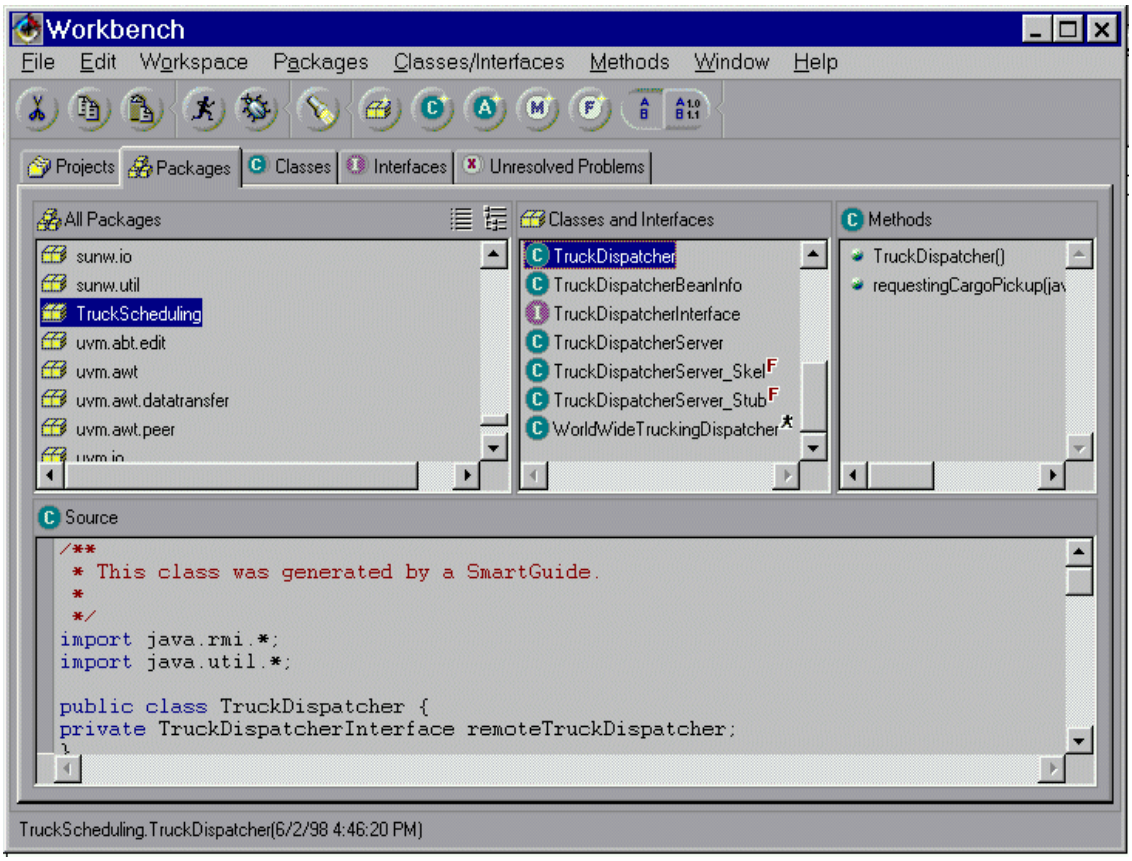


Figure 23. TruckDispatcher Class and Related Classes

5.2.5.3 Run Time

Like applets, servlets are run within a servlet context, such as a Web server JVM or tool like ServletRunner. Therefore, servlets can execute on any platform that supports a Java servlet context.

Servlets satisfy the JIT delivery method as well and can be downloaded from a remote machine. A servlet can be invoked several ways:

- Using a Web browser to open the servlet by way of a URL
- Specifying the servlet on the ACTION attribute of the <FORM> tag in an HTML file
- Specifying the servlet within the <SERVLET> tag on an S-HTML file

- Embedding the Java code for the servlet within a <JAVA> tag on an HTML file
- Specifying a servlet filter
- Specifying a servlet chain
- Specifying the servlet to be loaded when the Web server starts

Physically, the servlet is running within the JVM processor of the Web server, not the server from which it was originally downloaded.

5.2.5.4 Parameters and Environment Variables

The servlet can obtain information about itself by invoking `getServletConfig()` or `getServletContext()`. In addition, it has access to initialization parameters by using `getInitParameterNames()`.

5.2.5.5 Servlet Execution LifeCycle

When a servlet is loaded or reloaded, the servlet context creates an instance of the servlet and invokes the `init()` method. Any initialization that should be done only once should be performed here.

Once a servlet is loaded, it can expect to receive many client requests by the `service()` method. In general, servlets will receive an argument containing client request parameters and an argument intended to contain the response information. Servlets responding to the HTTP protocol should override the `doGet()` and the `doPost()` methods and simply rely on the `service()` method in the HttpServlet superclass to first parse the client request, and second, to delegate to the proper method in your subclass. An HttpServlet will create a response by writing HTML-formatted text onto the output stream contained by the HttpServletResponse argument.

When a servlet is about to be unloaded, the `destroy()` method is invoked. Servlets should free resources at this time

5.2.6 Basic Communication

Applets and servlets can exchange information using standard Java library classes that provide communication services, security restrictions not withstanding. Java I/O and Net packages contain support for reading and writing information to other processes by way of sockets and streams. Built on top of these components, there is an advanced feature called Object Serialization that supports sending a whole object or even an entire object web from the applet to the servlet through a special object stream.

5.2.6.1 Sockets

The classic socket scenario between a client and a server works as follows. A server creates a socket upon which it waits for a new connection. A client opens a connection to the server socket. The server accepts the request, creates a new socket, forks a new process to handle the client request and then resumes waiting for a new connection on the original socket. The advantage to this design is that it scales nicely.

Applying this to Java, the servlet creates an instance of the `ServerSocket` class and waits for a new connection by invoking the `accept()` method. An applet opens the socket by creating an instance of the `Socket` class. The servlet receives an instance of the `Socket` class upon accepting the connection, spins a Thread on an instance of `MySocketThread` and then resumes waiting for a new connection by invoking the `accept()` method on the ServerSocket. When started, the Thread instance invokes the `run()` method on the `MySocketThread` instance which you have coded to encapsulate the handling of the client request. The `run()` method begins reading and writing to the socket at will.

The only problem with this design is that the servlet will never exit out of its `init()` method accepting new connections. To avoid this, code another `Runnable` class named `MySocketListener` whose `run()` method implements the service for accepting socket connections described above. In the `init()` method of the servlet, spin a Thread on an instance of `MySocketListener`.

5.2.6.2 Streams

Information is read from and written to the socket by wrapping it within various kinds of streams. Reading information is accomplished by constructing an InputStream which provides the `read()` method, and writing is accomplished by constructing an OutputStream which provides the `write()` method. Generally, once a socket connection has been established, each side creates an InputStream using `getInputStream()` and an OutputStream using `getOutputStream()` from the local Socket instance. The InputStream and OutputStream are then wrapped in a FilterStream, the exact kind of which will depend on the format of the data that the client and server expect to send and receive, and the level of abstraction of the services for reading and writing.

InputStreams and OutputStreams allow you to read and write single or multiple bytes.

FilterInputStreams and FilterOutputStreams provide a caching aspect and a flushing service to improve performance. These streams are designed using the Wrapper pattern and are constructed using an instance of non-buffered InputStreams and OutputStreams.

DataInputStreams and DataOutputStreams are FilterStreams that provide higher level services to read and write primitive data types.

These type of streams are not very object-friendly. A client and server would each need to follow a strict protocol for trying to send the information contained in a typical address. If the order or format of the address changed each side would be forced to adjust accordingly. A more disciplined and object-friendly approach may be found with Object Streams.

5.2.6.3 Object Streams

ObjectInputStream and ObjectOutputStream provide services for reading and writing objects in the form of `readObject()` and `writeObject()` and specialized services for the Object form of primitive data types. Only objects which implements the Serializable or Externalization interface may be used with these Streams. Objects that wish to assist in their own serialization and de-serialization need to implement `defaultReadObject()` and `defaultWriteObject()`.

Using this approach, it is possible to encapsulate address information and behavior into an `Address` object and send the entire instance at once. This seems to solve the issues about management of information format, since the Java class defines the data and type of the Address attributes and both the client and the server will be imported the same file, Address.class.

However, this also raises some issues with basic object services such as object identity: Which instance represents the real Address? In effect, the object is being copied remotely because when the read/write service is complete, you will have both the new instance of Address on the client and the original instance of Address on the server.

5.2.7 Applet/Applet Communication

There are several options for communications within and between applets. A few of them are described here for your reference.

5.2.7.1 AppletContext

There is an interface called java.applet.AppletContext which allows an applet to get information about the environment it is running in, which is usually supplied by the browser or the applet viewer. The interface allows access to information such as the applet name, images, status, and it also allows usage of audio clips.

5.2.7.2 InfoBus

The InfoBus, mentioned briefly in Chapter 2.4.9, “e-business Application Services” on page 33, is a mechanism that allows communications between Java applets or JavaBeans within the same JVM. These components get "plugged into" the bus. The Version 1.1 specification released in March 1998 is compatible with the JDK 1.1.

A potential member can join the InfoBus and exchange, for example, simple values, cells of a spreadsheet, or even rows of a relational database, with other InfoBus members. The InfoBus is used as a conduit for dynamic data transfer between InfoBus participants, and multiple conversations can be occurring simultaneously.

Although the InfoBus is intended to support communications within the same JVM, it is possible to have a component implement RMI or IIOP to access information across the network, and publish this information onto the InfoBus, making it available for use by the other participants of the InfoBus.

5.2.8 Servlet/Servlet Communication

Two servlets that execute in the same JVM can call each other’s public methods directly. A servlet can communicate with a servlet on a different system in the form of an HTTP request by opening a connection using the URL of the server where there target servlet resides.

When calling another servlet’s public methods, the calling servlet developer should be aware of any re-entry issues associated with the target servlet, for example, if the target servlet implements the SingleThreadModel interface. In this case, the call should be made using HTTP.

In theory, a servlet can use the RMI or CORBA architectures (discussed later in this chapter) to extend its reach for data exchange or other communications across the network, to any similarly-based implementation.

5.3 Distributed Object Architecture

Earlier, we discussed how application services can be provided through basic communications in which the applet invokes requests directly on the servlet. The servlet component exposes its services to the applet in the form of some application-level protocol. However, in a properly designed server, the logic required to support these application services is delegated to a business object model. Thus, if we considered exposing these services as interfaces of individual object components, objects in an applet would be able to invoke requests directly on the business objects that provide the service, not the

servlet. The problem here, though, is that the business objects in the servlet do not live in the same object space as the objects in the applet, and the Java Run-Time Environment does not handle sending messages into another object space, for example, between two JVMs. This raises an issue of object location: how do I send a message to an object across the network?

In a perfect world, all known object space collapses to a single point and every object would appear as if it is local. An object should be able to invoke services on another object without knowing where or how the message will be delivered. A message sender should not need to know what machine the message receiver is physically located on, nor should it need to know how to transport the message securely and reliably. These inter-object protocol issues are addressed by distributed object architectures which define the mechanisms and services needed for object-to-object communication across the network. There are several options already implemented as Java APIs for the Java developer.

5.3.1 Remote Method Invocation (RMI)

As stated previously, the RMI API is included in JDK 1.1 and provides a method for creating Java objects on a remote machine whose methods can be invoked locally by other Java objects.

RMI is limited to Java-to-Java communication. This may be a problem for some situations (for example, heterogeneous run-time environments) and is a great thing for others (for example, pure Java Run-Time Environments). In our discussions so far, both applets and servlets are written in Java and would therefore be able to benefit from this API.

5.3.1.1 RMI Architecture

Let's suppose a client object needs to invoke services of some remote business object. In the RMI design, the client interacts with a remote object through a published interface. The client-side object sends a message to a local RMI object that implements the required services of the remote object. The local RMI object implements each method in the remote object's public interface, but delegates the behavior back to the remote object using the RMI infrastructure. To the client object, the local RMI object appears as if it is the remote object and the actual mechanism of communication is hidden as well. The client object never directly interacts with the remote business object.

In this model, you are allowed to pass a reference to a remote object in an argument or return it as a result. RMI is based on ObjectStreams, and therefore, object references get passed by value and implement the Object Serialization interface as discussed earlier.

This model assumes that the remote object already exists when the client sends a message. RMI does not provide a means of creating the remote objects in response to a client request for service, but there is a factory-design pattern that can be used to overcome this.

The mechanism for RMI communication is divided into several layers on the client, each with a counterpart layer on the server. The first layer is the Stub/Skeleton layer, the second is the Object Reference layer and the bottom is the Transport layer.

Stub/Skeleton Layer
This layer is designed using the Adaptor pattern in translating the business object interface to and from the operations defined in the Remote Reference API. It results in shielding the client side objects and server side business objects from being aware that they are participating in a conversation across the network, and that the conversation is using the RMI framework. The Stub participates on the client side pretending to be the remote object by implementing the business object's interface. The Skeleton participates on the server side pretending to be the client object that is invoking the service.

Remote Reference Layer
This layer is designed using the Bridge pattern to connect the business object model to the network communication model. It is responsible for issuing the object to object protocols using the native transport calls.

This layer can raise exceptions on both the client and server side when problems occur when establishing connections or delivering messages.

Transport Layer
This layer is responsible for establishing and maintaining connections. These connections are built on the services provided by sockets using the TCP/IP protocols as discussed earlier.

5.3.1.2 Development
Unfortunately, business objects that act as remote objects must subclass from `java.rmi.UnicastRemoteObject` and implement an interface that extends the `java.rmi.Remote Interface`. This may be a problem if you already have an established enterprise hierarchy in Java. A general solution to this is to subclass all of your business classes from a common superclass which itself is a subclass of `UnicastRemoteObject`. Therefore, if you need to move away from the RMI architecture, the impact would be minimized to changing the hierarchy of just one class.

RMI requires that you, the developer, write the `Stub` and `Skeleton` classes, as they are based directly on the interface exposed by your business object model. We illustrate this code in Chapter 5.3.1.5, “JavaBeans and RMI” on page 98.

The good news is that you can generate all these stub and skeleton classes needed at this layer from your existing Java-based business classes using the RMI compiler, `RMIC`, which is included with the RMI components of the JDK.

An alternative approach to generating RMI code is to use the VA Java RMI Generation tools (see Figure 24 on page 97, which indirectly invokes RMIC). RMI-based applets and servers can be developed with the inexpensive VA Java Professional Edition. The RMI-based applications illustrated in this section were developed with VA Java (Professional) Version 1.0. A more powerful VA Java Enterprise Edition is available for doing enterprise-level computing, including JDBC and CICS Access beans (see `http://www.ibm.com/java` for more information).

We have already shown that JavaBeans run just fine in an applet on an NC, so there is no reason why an NC applet can’t include an Enterprise Access Bean (either JDBC or CICS). However, such discussions are beyond the scope of this book.

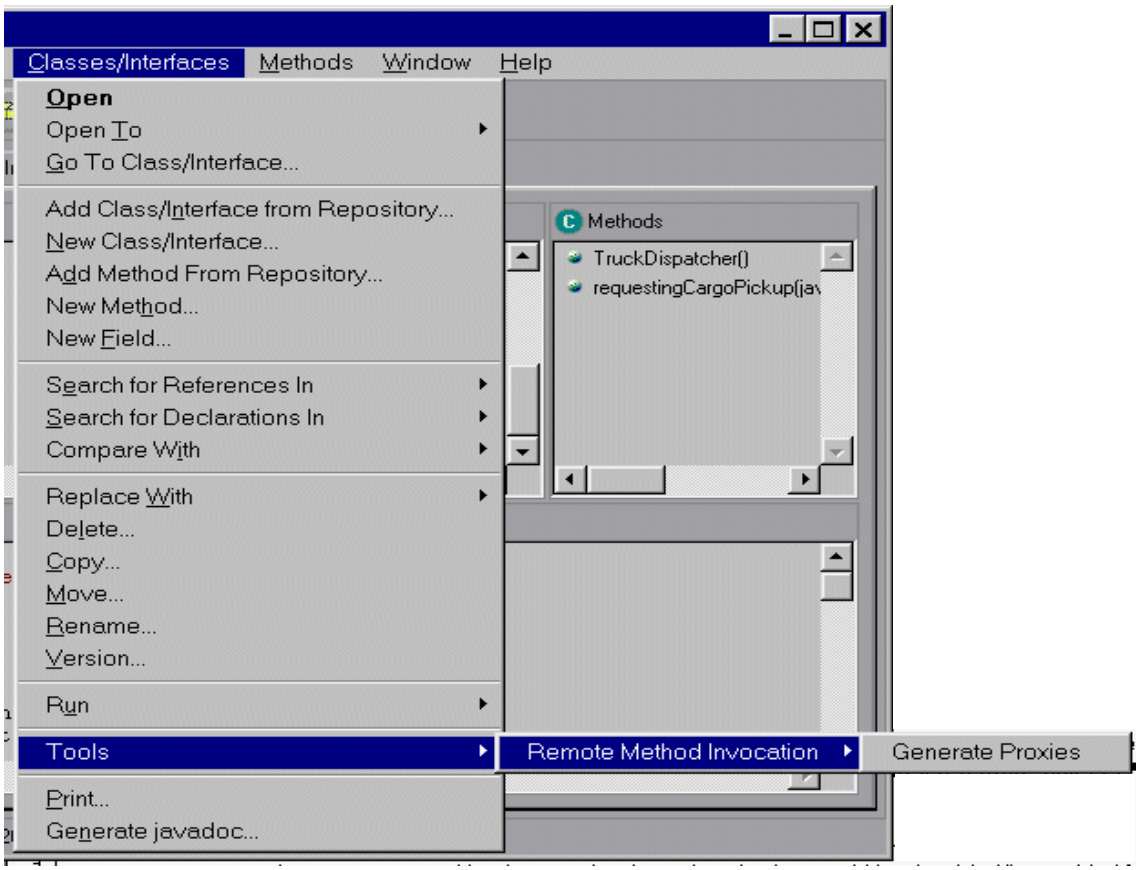


Figure 24. VA Java RMI Code Generation

Security

Clients and Servers should set the security manager to be the RMISecurityManager before using RMI-based remote objects.

Naming Service

Servers need to bind new objects that provide RMI-based services into the Naming service by using the `java.rmi.Naming.bind()` or `rebind()`. This method takes as arguments the remote object and a String that is the unique name with which the remote object is bound.

Clients need to obtain a reference to remote objects by using the `java.rmi.Naming.lookup()` class method.This method takes as an argument the name with which the remote object has been bound. This method

answers an instance of `Remote` upon which you will need to narrow the object by casting it down to the correct Interface before invoking one of its methods.

Exceptions

There are a few `RemoteException` classes that need to be caught such as:

- `RMISecurityException` can be thrown when setting the security manager.
- `NoSuchObjectException` can be thrown when attempting to lookup an object.
- `AlreadyBoundException` can be thrown when trying to bind an object.

5.3.1.3 Remote Factories

A client cannot assume that the remote object already exists on the server, as the `NoSuchObjectException` and RMI do not provide a service for automatically creating objects that extend `UnicastRemoteObject`. So, instead of asking for the remote object, a client can ask for a well-known remote factory object. The client then invokes a `lookup()` method on the remote factory which will answer the remote object if it exists, or create and bind the remote object if it doesn't exist. When using this approach, create one remote factory for each group of similar business classes.

5.3.1.4 Callbacks

So far, we have only discussed RMI as a one-way street, that is, where objects on the client send messages to objects on the server. RMI addresses server-to-client messages as callbacks. In this model, an additional interface is created which defines the available callback methods the client must implement. The original interface for the remote object is augmented to include an additional method named `addCallback()` which takes as an argument an object that will receive the callback. The remote object will also have to add a means for remembering the list of callback clients.

So, it is possible that the client object wishing to receive callbacks and supporting the callback Interface invokes the `addCallback()` method with itself as the argument on a remote object. In response, the remote object adds the client object to its list of callback recipients. Some time later, when an event occurs that requires notifying clients, the remote object will enumerate through its list and invoke an appropriate callback method on each member.

5.3.1.5 JavaBeans and RMI

Designers and developers of NC-based Java applications that use RMI need to understand and review the design and development process for implementing the server's RMI support. The greatest applet in the world running on an NC isn't any good without a server to talk to. So, every

programmer for the NC will probably need to do some work on the server side at some point. For that reason, we will spend considerable time in this section on the mechanics of the development process.

Now, referring to our WorldWide Trucking Company example code, we will take our non-visual TruckDispatcher bean and extend it into a full-fledged RMI-based server application that will run on a remote node. The RequestCargoPickup applet previously developed will be used to access the server.

Note that the full code contents for the entire WorldWide Trucking Company project (including the files from the TruckScheduling package discussed here) are available in a JAR file on removable media that accompanies this document. Import the JAR into your VA Java and look around.

Our goal is to produce the necessary classes and interface previously illustrated in Figure 23 on page 89. Important classes include:

- `TruckDispatcher` Class

  Referring to Figure 23 on page 89, we see that there is a declaration of the `TruckDispatcherInterface` in the basic class definition. We extended our `TruckDispatcher` class' constructor (Figure 25 on page 100) and `requestingCargoPickupMethod` (Figure 26 on page 101) to include the needed code to use RMI.

- `TruckDispatcher` Interface

  We used the VA Java SmartGuide to generate an interface for us (see the result in Figure 27 on page 102). Note that you must specify that you are creating an interface (see Figure 28 on page 103) and must explicitly choose to extend java.rmi.Remote (see Figure 29 on page 104).

- `TruckDispatcherServer`

  We used the SmartGuide to create this server class. As mentioned above in Chapter 5.3.1.2, "Development" on page 95, we had to define the server as inheriting from `java.rmi.UnicastRemoteObject` (see Figure 30 on page 105) and implementing an Interface which extends the *java.rmi.Remote* Interface (see Figure 32 on page 107).

- `TruckDispatcherServer_Skel`
- `TruckDispatcherServer_Stub`

  The server skeleton and server stubs were automatically generated using the VA Java proxy generation capability, using the commands from the menu structure shown in Figure 24 on page 97.

There are some basic steps to follow to enable RMI support in your server, which will implement the classes outlined above. No applet conversion was necessary. The steps are:

1. Write code to get a reference for the remote server object (Figure 25 on page 100).
2. Link our local JavaBean to the remote server (Figure 28 on page 103).
3. Define the interface that will be used by our local bean and the remote server (Figure 29 on page 104).
4. Define the remote server class (Figure 30 on page 105).
5. Generate the RMI communication stubs that are based on the remote server class (Figure 24 on page 97).
6. Define the actual remote server application, paying particular attention to the generation of the server's main method (Figure 33 on page 108).
7. Write the actual code for the remote server application (Figure 34 and Figure 35 on page 109).
8. Export the server to the remote node, start the RMI registry on the server (Figure 36), then run the server application.

The following screens and paragraphs explain these steps in more detail.

```
try {
  // Use the RMI name server to look up our remote Dispatcher. Note that we
  // are specifying access to a specific node at a specific port number. This
  // assumes that the Remote Truck Dispatcher has previously been
  // manually started on that remote node. This name lookup returns an
  // object that has implemented the remote interface on the remote node.
  Remote remote = Naming.lookup
("//oc0263f.itsc.austin.ibm.com:1099/TruckDispatcherServer");
  // Now convert the generic remote object we just received into
  // a reference to a specific remote Truck Dispatcher object.
  if (remote instanceof TruckDispatcherInterface)
  remoteTruckDispatcher = (TruckDispatcherInterface) remote;
  }
```

Figure 25. TruckDispatcher Constructor's RMI Extensions

The java.rmi package was included in an import statement in the TruckDispatcher class definition. If it is not there automatically, be sure to add it.

In the constructor, we had to add try-catch code to handle java.rmi exceptions. (For simplicity, the handler just reports the problem, see the full code implementation on the removable media that accompanies this document.)

Here in the constructor, we specify that the remote server is called TruckDispatcherServer and is located on the specified remote node of

`//oc0263f.itsc.austin.ibm.com:1099`

and that TCP/IP communication will occur through port number 1099. If you want to recreate this example, just substitute your specific node and the port number you want to use.

Note that our communication with the remoteTruckDispatcher occurs through the `TruckDispatcherInterface`.

```
try {
......
/* Use our local proxy to perform the requestingCargoPickup method
 * on the remote instance of the truck dispatcher's interface. */
  approvalStatus =
remoteTruckDispatcher.requestingCargoPickup(customerName,
truckQuantity, pickupAddress, pickupDateTime);
} catch (java.rmi.RemoteException e)
{
System.err.println ("Error talking to remote truck dispatcher" +
e.getMessage());
}
```

Figure 26. RMI in Action: Running a Remote Method on the Server from an Applet

Note how invoking `TruckDispatcher.requestingCargoPickup` on a local version of the TruckDispatcher non-visual bean in our applet causes `remoteTruckDispatcher.requestingCargoPickup` to be invoked on the server application. This is the very core of the RMI mechanism in action. We have just run the method remotely on our remote server. WWTC's business logic is executed on the remote server to determine if trucks are available and the approval status is returned to us for use in displaying the result on our local applet's display screen on the NC.

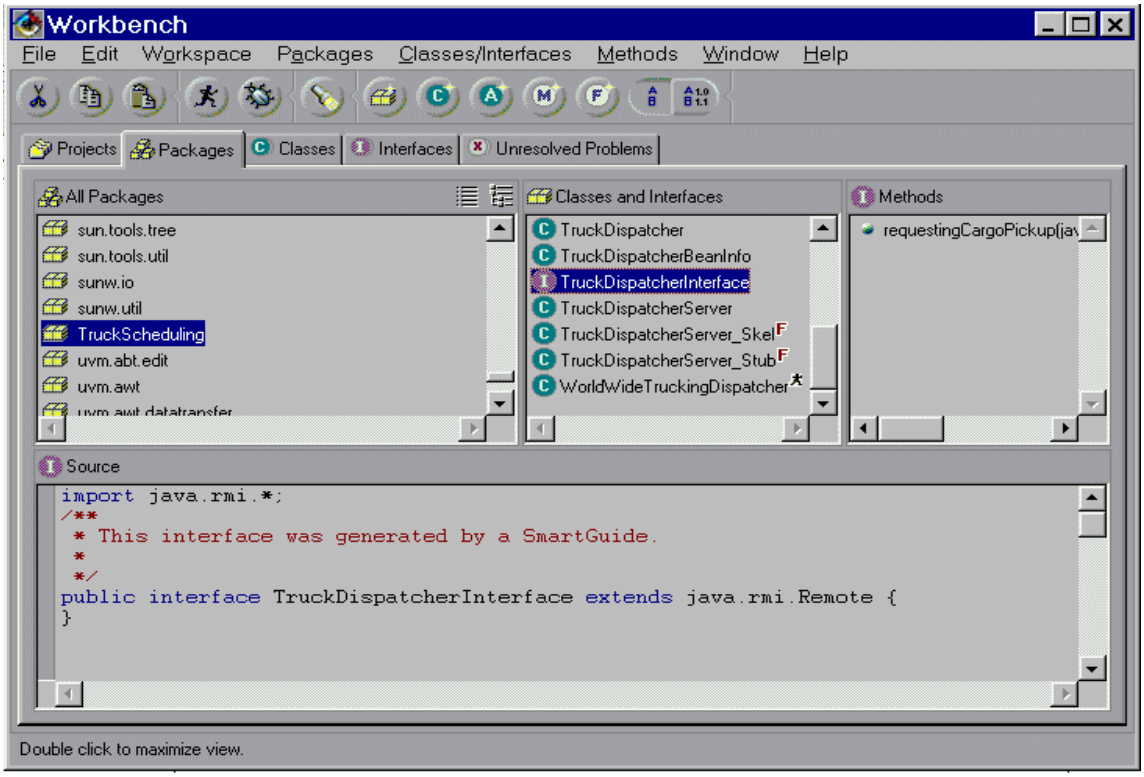


Figure 27. RMI Remote Interface Definition

Note how the interface extends java.rmi.Remote. We had to specify that extension while using the SmartGuide.

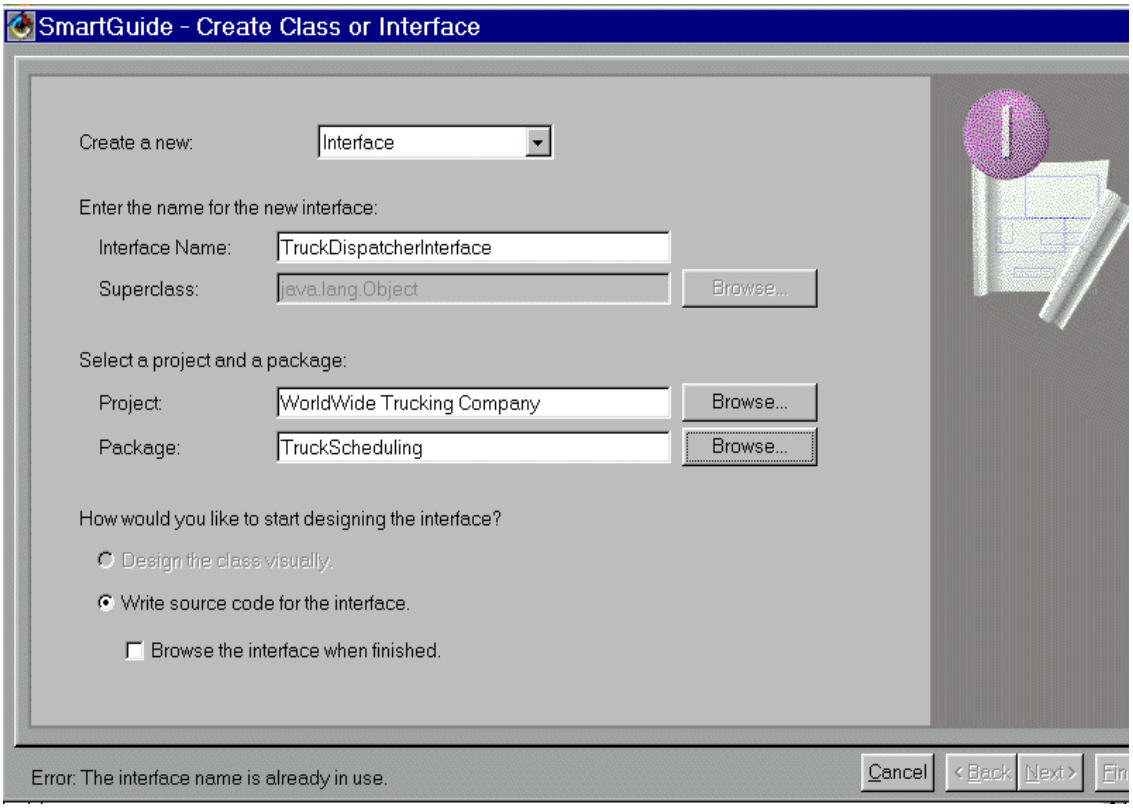


Figure 28. Creation of TruckDispatcherInterface

Note how the SmartGuide forced you to choose to write source code for the interface. There is no visual element to programming interfaces.

However, our interface was so simple that we didn't have to add any code to the interface definition or to the requestiongCargoPickup definition. We got all these for free from the SmartGuide and from the method definition of the TruckDispatcher bean itself.

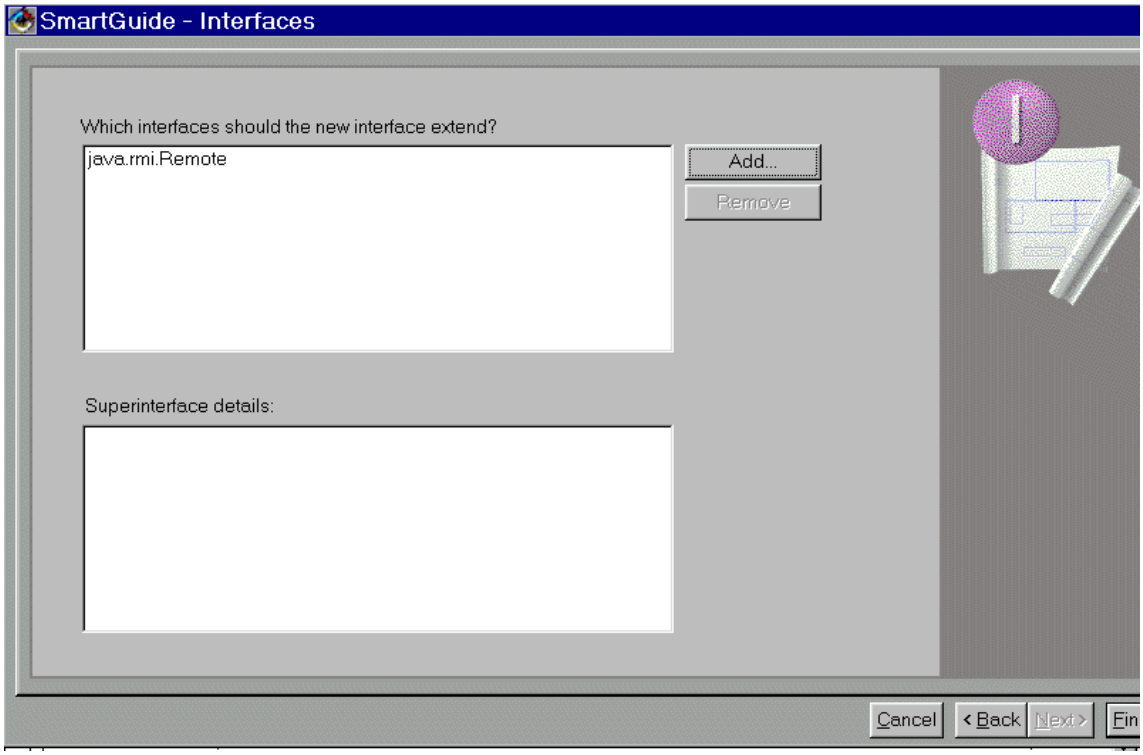


Figure 29. Specifying Interface Extension for an RMI-based Application

Next, we had to use the SmartGuide to implement the `TruckDispatcherServer` class, shown in Figure 30 on page 105.

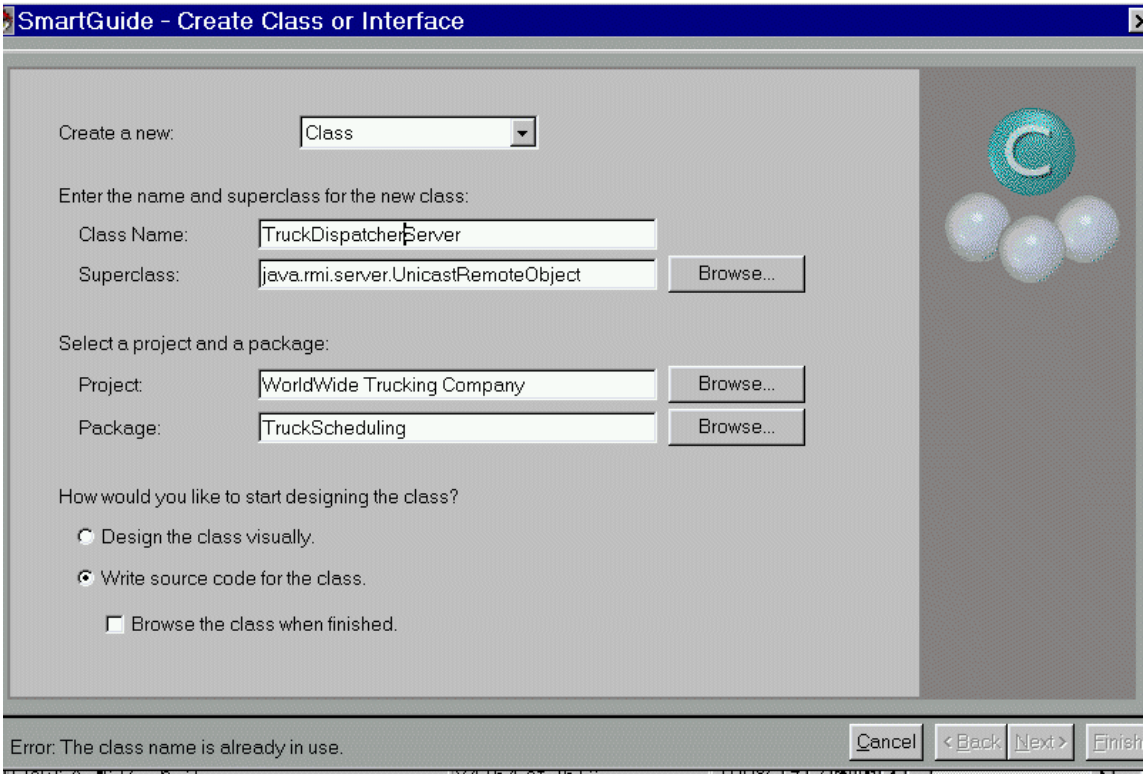


Figure 30. TruckDispatcherServer Inherits from Unicast Remote Object

We enter the class name, `TruckDispatcherServer`, and use the defaults for the class design, which is to write source code for the class (as shown in Figure 30).

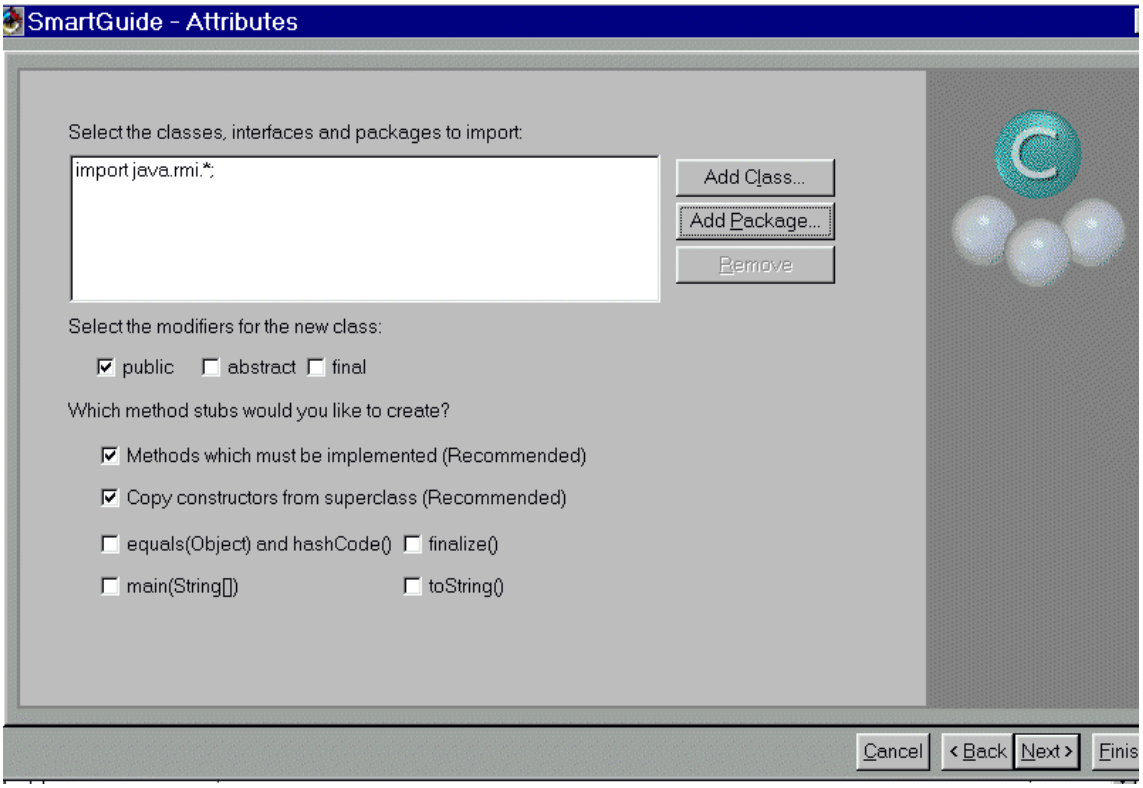


Figure 31. TruckDispatcherServer - Important Characteristics

Note the necessity of importing java.rmi, which is no surprise, since we are using RMI. More important is the acceptance of the tool's recommendations for constructor and method implementations. For the type of application illustrated in this section, always accept these defaults, as shown in Figure 30 on page 105 and in Figure 31.

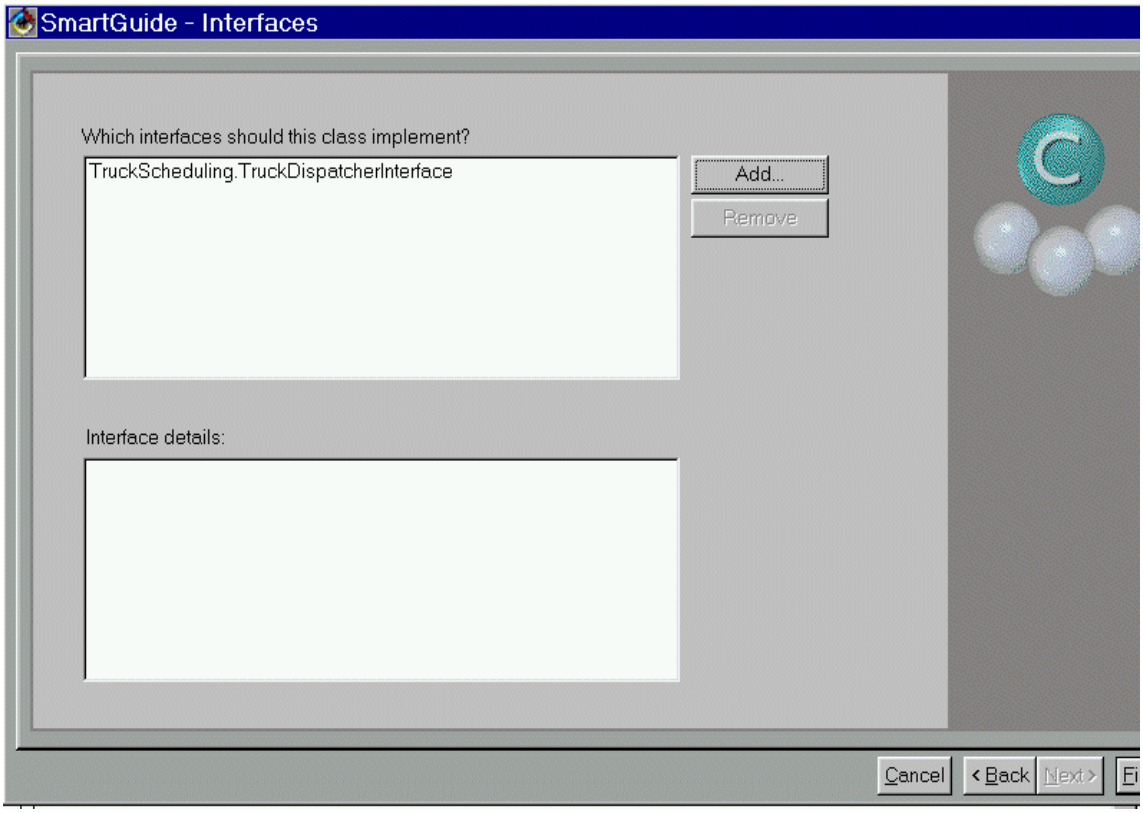


Figure 32. Interface Implementation

The TruckDispatcherServer class must explicitly implement the desired interface, so that you can use the VA Java's automatic RMI proxy generation tool. Define the communicating methods (in our example, `requestingCargoPickup`) identically for the original class, the interface class, and the server class.

Next, we used the SmartGuide to define an actual applet that will become the main server application that runs on WWTC's remote server node. This applet is the WorldWideTruckingDispatcher. We need a main method, since we will run this as a standalone server application on the server node. Define this during the class creation via SmartGuide, as shown in Figure 33 on page 108.

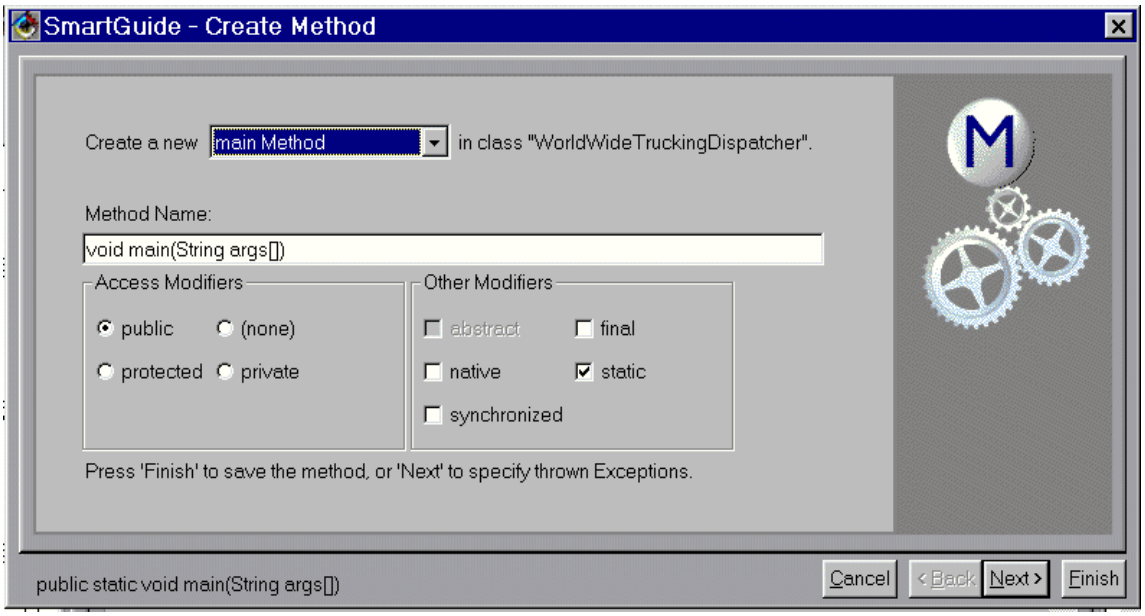


Figure 33. Main Method Creation for the Remote Server Application

Modify the characteristics illustrated in Figure 33 above to match your actual server application needs. The defaults shown were sufficient for our example application.

The main method code for the remote server is shown in two parts (Figure 34 and Figure 35). Note that the appearance of the main method code has been modified slightly for text formatting purposes, and non-essential code boilerplate has been removed for clarity.

```
/**main entrypoint - starts the application */
public static void main(java.lang.String[] args) {
  // Finally, create the real truck dispatcher that runs on the main
  // server box at WorldWide Trucking's headquarters.
try {
TruckDispatcherServer theRealTruckDispatcher = new
TruckDispatcherServer();
// Remember that on the client side we did a naming lookup.
// On this side we have to bind the dispatcher server to the well-
// known name that the client was looking for. Code is machine-specific.
Naming.rebind
("//oc0263f.itsc.austin.ibm.com:1099/TruckDispatcherServer",
theRealTruckDispatcher);
```

Figure 34. Remote Server's Main - Part 1

In Figure 34, notice the use of `rebind`. This means if the server is restarted, the name definition will automatically be redefined for this new instance of the remote server. If `rebind` was not used, an exception would be raised for the redefinition attempt.

```
System.out.println("Bound Truck dispatcher Server\n");

// Keep the dispatcher alive forever, giving time for the
// client to attach.
Thread.currentThread().join();
} catch (Exception e) {System.out.println(e); }
return;
}
```

Figure 35. Remote Server's Main - Part 2

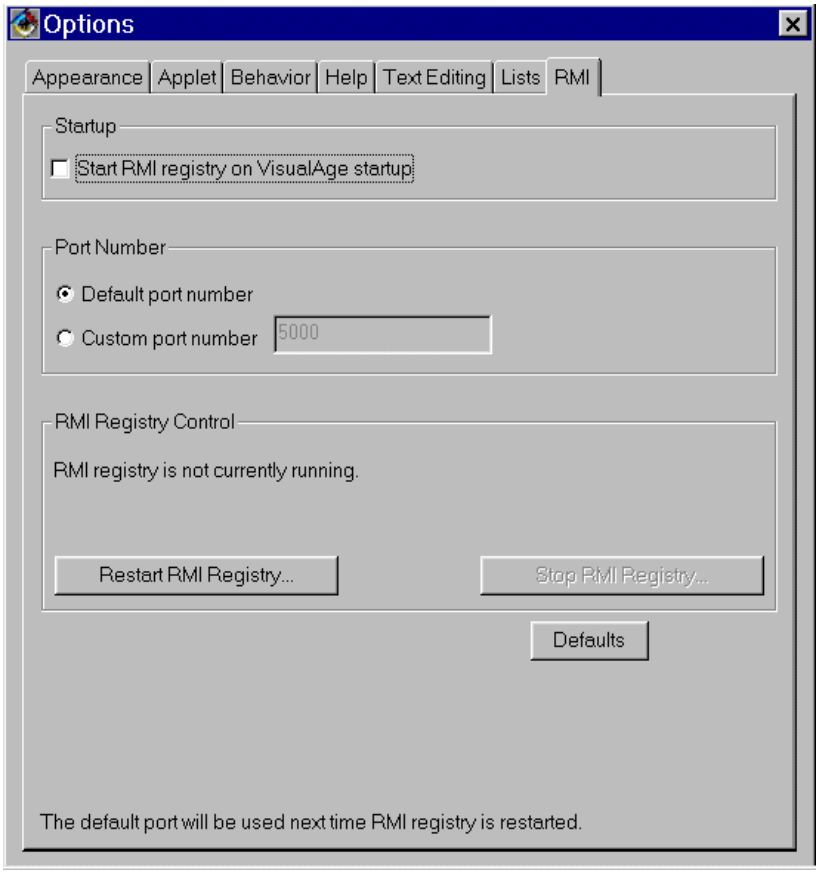


Figure 36. RMI Registry Startup

The menu choice to access RMI registry start panel is Workspace->Options, then select the **RMI** tab to display the startup panel (see Figure 36).

5.3.1.6 Run Time

Before the server and client components are started, you should start the RMI Registry on the server. This is done differently on each platform. (VA Java for Windows NT version is shown in Figure 36.)

The registry listens to a specific port and can serve the Naming requests of both the client and the server. This is important for the NC, which doesn't run its own version of RMI Registry.

Of course, since our example in this RMI programming section (Chapter 5.3.1.5, “JavaBeans and RMI” on page 98) is a server and not a servlet, you must manually start the server application on the server node.

Dynamic Stub Loading
Since references to remote objects can be passed as parameters, stub classes need to be downloaded when a class is not found locally. When using applets, the AppletClassLoader and AppletSecurityManager will control theses services at run time.

Garbage Collection
RMI uses a reference-counting garbage collecting scheme that attempts to track all outstanding references to a remote object. Each time a client obtains a reference to the object, the object’s reference count is increased, and each time a client loses the reference, the object’s reference count is decreased. Eventually, the count should reach zero, and RMI places the object on the weak reference list on the server which allows it to be garbage-collected.

Client remote references are considered leased and may expire, causing the client to be taken off the remote reference list.

Object Services
Persistent object references will be available in JDK 1.2, which will allow clients to re-establish a reference to a remote object after a server failure, which may have resulted in a crash.

Remote object activation will also be available in JDK 1.2, which will allow clients to assume that remote objects already exist on a server.

5.3.2 Common Object Request Broker Architecture (CORBA)

Like RMI, CORBA is based on a proxy design where the client object invokes methods on a stub, which in turn uses an ORB infrastructure to transport the message to the remote object. Although it appears to the client that the proxy is the remote object, the client object never interacts with the remote object directly. Because a client side ORB and a server side ORB can be implemented in any language, they communicate using an open, standard Internet Inter-Orb Protocol (IIOP).

Unlike RMI, the CORBA object bus is a specification that takes a language-neutral approach to inter-object communication across a network. One of its major goals is to allow objects implemented in different languages to communicate using an Interface Definition Language (IDL) that defines the exposed services of each. Objects, services and frameworks that make up

the CORBA model are each defined using IDL, which provides a benefits of having a system meta-model available in development and run time.

It also attempts to address many of the issues we have already raised about distributed objects by defining a long list of universal object services for managing objects in the real world, such as those services needed by a robust e-business solution. For more information about CORBA, refer to *Understanding CORBA*, by Otte, Patrick and Roy, Prentice Hall, 1996, ISBN 0-13-459884-9.

5.3.2.1 Interface Definition Language

When the client and server communicate, they need to agree on what kind of information will be shared and exchanged. The Interface Definition Language helps accomplish this. The IDL defines the interfaces, data types and objects to be used between client and server. The IDL enables some level of programming language independence, including support for C, C++, Smalltalk., COBOL, Ada, and Java.

Module

The module keyword is used to avoid confusion between interface names specified in your IDL and names in other programming languages or systems on the network. It helps provide a scope or context for the interfaces defined in the IDL.

IDL-to-Java Compiler

The IDL-to-Java tool was not part the JDK installation utilized in this redbook. However, it is available from the following URL at JavaSoft:
`http://developer.java.sun.com/developer/earlyAccess/jdk12/idltojava.html`

5.3.2.2 Architecture

As of this writing, the current version is CORBA 2.2. The OMG Guide defines four areas that make up the object architecture. See Figure 37 on page 113 for a graphical representation of this architecture.

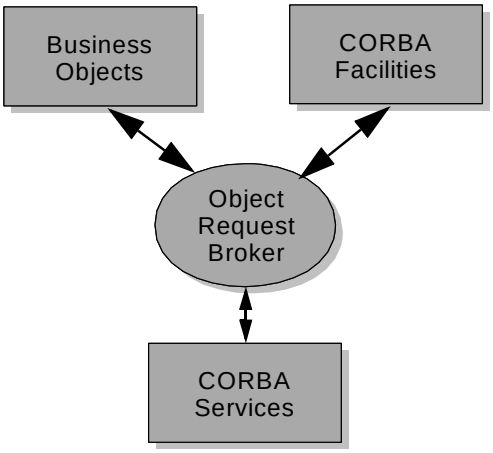


Figure 37. Basic CORBA Architecture

ORB

The Object Request Broker (ORB) is the main mechanism for inter-object communication. The ORB handles the issues of knowing where an object is and exactly how to send it a message. It distinguishes itself from other middleware by providing for dynamic method invocations at run time. As part of the language-neutral, platform independence, Windows 95 objects written in Java can send messages to AIX objects written in Smalltalk. ORBs allow for self-describing systems, as they must implement an Interface Repository describing the services provided.

The components of an ORB:

- Client-side IDL Stub
- Dynamic Invocation interface
- Interface Repository
- Server-side IDL Skeleton
- Dynamic Skeleton Interface
- Object Adaptor
- Implementation Repository

CORBA Services

So far, the OMG has published 15 IDL-defined services for managing objects. These system-level services can be applied to your application to provide varying levels of robustness. Since they are defined as services and are

available dynamically, component providers do not need to design for these issues. However, whether these services are supported is up to the ORB implementation.

- LifeCycle Service - Object are born, live a happy life, then die. This service helps manage creating, copying, moving and deleting objects
- Persistent Service - Objects are around as long as the process that created them is up and running. If you want objects to exist beyond this point, you will need to manage persistence by placing them in some form of data store like a database.
- Naming Service - Trying to find an object on the network means being able to identify the object by name. This is managed by a naming service.
- Event Service - Objects want to be notified of a change in an another remote object's state. This is managed by the event service, which defines an event channel for broadcasts.
- Concurrency Control - Typical business solutions need to address many client requests simultaneously. Managing locks on sensitive resources, such as threads and transactions, is handled by the concurrency control service.
- Transaction Service - This service provides a robust two-phase commit for transactions at the object level.
- Relationship Service - Components that need to be related dynamically will rely on this service for managing referential integrity.
- Externalization Service - At the edge of the CORBA universe, information enters and exits as data streams by way of the Externalization service.
- Query Service - Perhaps you don't have a particular object in mind, but rather a set of objects that match a certain criteria. These requests are provided by the query service in the form of an Object Query Language (OQL).
- Licensing Service - Usage-based systems need to track the activity of objects at different levels of granularity. This service supports tracking hits from objects to web sites.
- Properties Service - This services allow you to dynamically associate name/value pairs with an object.
- Time Service - If objects wore watches, they would need to use this service to manage a common knowledge of time. Also, this service provides a trigger for time-based actions.
- Security Service - This service provides the framework for keeping your business objects secure.

- Trader Service - As we enter into a global, commercial object space, there may be a choice of which object to use. This service provides a framework for advertising services and competing for work at the object level.
- Collection Service - This service deals with handle common forms of object collections.

CORBA Facilities

The CORBA group is also attempting to define a set of IDL-defined application services that provide both vertical and horizontal frameworks for direct use by business objects. These target the areas of data interchange, workflow, firewalls and internationalization.

Business Objects

At the top of the CORBA architecture is the layer of IDL-defined business objects that you create to model your business. They represent the core entities of your business and industry. To encourage re-use, these should be designed without knowledge of the application context in which they will participate.

5.3.2.3 Development

There are many Java implementations of the CORBA specification which claim CORBA 2.0 compliance. However, each provides varying degrees of support for the object services level.

Each of these packages relies on some form of Java emitting tool which generates the stub and skeletons from the .idl files. Some tools will also generate Java files from existing business classes.

In general, servers initialize an ORB, create business objects, connect the business objects to the ORB, and then bind the business objects to a name using the name server.

The client will also initialize an ORB, obtain the root naming context and then request objects by name using the naming service.

Using the Java IDL API

The Java IDL API provides the base Java classes and tools for implementing CORBA objects in Java. The API provides a Java-based ORB which can be imported into both the servlet and the applet and an object naming service. Other vendors provide extensions to this API that implement their own ORB and provide support for the rich set of object services defined by CORBA.

In developing the IDL-defined business objects, follow these steps:

- Define each business object as an IDL-defined Interface.
- Run the idltojava tool to generate Java implementations of the business interface, the stubs and skeletons, and the helper and holder classes.
- For each skeleton, create a subclass that represents the CORBA business object by implementing all methods in the IDL specification.

In writing the servlet, follow these steps:

- Initialize the ORB in the servlet's `init()` method by invoking the static class method `omg.org.CORBA.ORB.init()`.
- Create instances of the business object classes and register them with the ORB by invoking the `connect()` method on the ORB.
- Obtain the root NamingContext by invoking `resolve_initial_references()` method on the ORB and passing 'NameService' as an argument.
- Bind the business objects into a name space hierarchy by invoking the `rebind()` method on the root NamingContext and passing the business object and its name as arguments.

In writing the applet, follow these steps:

- Initialize the ORB in the applet's `init()` method by invoking the static class method `omg.org.CORBA.ORB.init()`.
- Obtain the root NamingContext by invoking `resolve_initial_references()` method on the ORB and passing NameService as an argument.
- Obtain the object reference to a named business object by invoking the `resolve()` method on the root NamingContext.
- Invoke any of the IDL-defined methods on the business object by sending a message to the object reference.

5.3.2.4 Remote Factories

Like RMI, clients cannot create remote objects. To solve this, the Factory pattern is recommended to create objects remotely. For each object type, bind a CORBA-based object to a well-known name that provides a `create()` method. When invoked by the client, this method should instantiate a new business object on the server, connect it to the ORB, bind it into the naming context and then return an object reference to the client.

5.3.2.5 Run Time

When the applet is loaded, the CORBA Java packages are downloaded from the server and loaded. In effect, the ORB is bootstrapped just-in-time onto the Network Computer.

Javasoft also provide a transient naming service that is a stand-alone application that can serve the requests of both the applet and the servlet at run time.

5.3.3 RMI over IIOP

In July 1998, IBM and Sun announced the intent to co-develop and deliver support for Java RMI over CORBA's IIOP protocol. This function will be delivered as extensions to the existing JDK 1.1.6 and the future JDK 1.2, and will be a part of the core JDK in a future release.

This allows Java developers to access the world of CORBA distributed objects using native Java interfaces. Sun will define a subset of RMI functions that will be transportable over IIOP, and Java applications will be able to use RMI over IIOP to communicate applications in other programming languages as supported by the CORBA specifications.

Chapter 6. Maintenance

This chapter describes some of the issues a developer or programmer faces when maintaining existing code, regardless of the language. Java-specific considerations are also discussed. As the WorldWide Trucking Company (described in Chapter 3, “Application Example - WorldWide Trucking Company” on page 43) enters the code maintenance phase of their applications, many of the software maintenance issues described in this chapter will need to be addressed.

Software maintenance is the process of modifying the software product after it has been implemented. The types of software maintenance can be broken into the following groups:

1. Corrective
2. Adaptive
3. Perfective
4. Preventative

Of these, the first three categories represent about 95% of the software maintenance effort, and will be dealt with in this section. We also describe the impact of using distributed objects in the software maintenance effort.

6.1 What Makes Maintenance Difficult

Each category of software maintenance has key factors that make that group of tasks difficult. In this section, each of the types of software maintenance will be discussed to see how they are impacted in an NC environment.

6.2 What Makes Maintenance Expensive

All software can be rewritten. It is simply a matter of how much effort it takes to change it and make the affected modules ready for production. The primary cost of changing software is effort. Software maintenance costs are almost all associated with the effort needed to understand programs, find what code needs to be changed, make the change, and test the changed program in a variety of end user environments. If careful software design can reduce this effort, then this design effort will also reduce costs.

6.2.1 Corrective Maintenance

Corrective maintenance involves the diagnosis and correction of undesired events (UE) that occur in software. The following two groups provide a rough

© Copyright IBM Corp. 1998

classification for UEs, (described by Parnas in "Response to Undesired Events in Software Systems", *Proceedings of International Conference on Software Engineering*, IEEE Computer Society Press, 1976), that will be used in this chapter.

1. Incidents - these are UEs that, although undesired, are expected and for which recovery is successful.
2. Crashes - all other UEs.

The correction of UEs begins with the gathering information from the following questions:

1. How did the error manifest itself?
   What happened that made it apparent that unacceptable program behavior had taken place (for example, a console error message, "the screen went black")?
2. Where did the error manifest itself?
   In the case of a crash, where in the program did the crash take place? In the case of an incident, where in the code was the incident detected (that is, where was the error signalled to an error handler)?
3. What were the values of variables at the time of failure?
   Once we determine the value of program variables, this can tell us the data state at the time of the UE.
4. How did the program get to this point?
   In determining the cause of a UE, we can discover the sequence of events that took place to arrive at the final program state where the UE occurred. This path of logic followed by the program is followed, and the reason for this logic flow is also determined. This, in fact, traces the changes in data state from the start of the program to the point where the UE took place.

6.2.1.1 Implications for Distributed Objects

In a monolithic system of programs, this information can be difficult to find when it exists. A distributed application multiplies the difficulty of finding this information by the number of affected platforms in the network. Consider the relationship in Figure 38 below.

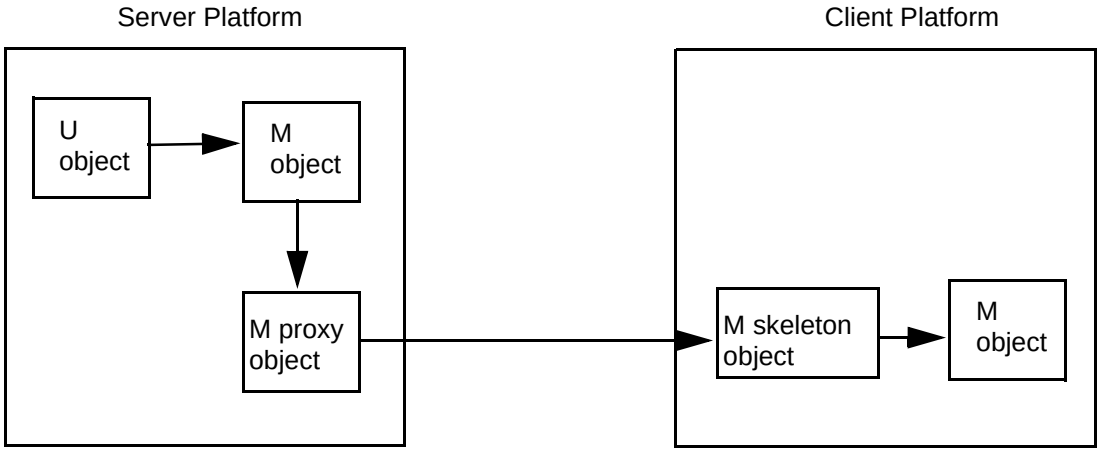


Figure 38. M/U Diagram for Distributed Environment

In the diagram shown in Figure 38, The M object appears to be local from the perspective of the U object on the server platform. But, in fact, the M object has a proxy. The U object actually communicates through the M proxy and the M skeleton (on the client platform), to the M object on the client platform. The M proxy and skeleton are responsible for communications across the network, keeping this isolated from the M object on the server platform.

In a distributed application, the information needed to help debug a problem might be spread across several platforms. The difficulty of finding this information is multiplied by the number of affected platforms in the network. Let's assume that an incident occurred (as defined in 6.2.1, "Corrective Maintenance" on page 119) in Figure 38 on page 121. The incident occurs on the client in the M object that is part of the A applet by signalling an error to the error handler. The error, however, actually originates on the server in the U object. Now consider the information the programmer needs to gather.

1. How did the error manifest itself?
   The error manifests itself by triggering an incident.
2. Where did the error manifest itself?
   The error manifests itself in an applet on the client computer.
3. What did the variables hold at the time of failure?
   The relevant variables were to have the data sent from the U object, which caused the failure. Both the platform where M was executing, as well as

the platform where U was executing, must be analyzed for their data states at the time of failure.

4. How did the program get to this point?
   The program got to this point by trying to interpret a message sent by the U object. Establishing the sequence of events that took place to get the M object into the failure state requires recreating the data states from the start state to the failure state. But this involves recreating a complex set of at least two series of data states, one for the M object and one for the U object, and understanding the interactions between the two objects.

So where is the problem in this example? This, in fact, is a difficult question to answer. For the M object, the U object looks local. But a programmer must trace the error from the U object to the M proxy, then to the M skeleton and finally to the client M object. Tracing errors across platforms like this can be very difficult.

What if the error is a crash? A crash might disable the JVM, causing the software on the client to terminate abnormally. In this case, the operating system might release the systems storage that Java no longer needs. With the absence of persistent local storage on an NC, the local context of the error might not be available for debugging.

One solution here is to log state changes and inter-object messages in a persistent form of storage.Since persistent storage does not exist on thin clients, the logging must take place on the server, adding more network traffic and more complexity to the application.

Clearly, debugging UEs can be more complex in an NC network environment. This places even greater emphasis on solid documentation and good design.

6.2.2 Perfective Maintenance

Perfective maintenance is the process of altering and enhancing the functionality of software to meet new needs. The key issue in perfective maintenance is mapping functionality to where the functionality is embodied in the code.

6.2.2.1 Implications for Java

In a distributed object environment, program source code text is broken into classes. Each class usually specifies a set of data and almost always specifies a set of behaviors. The behavior, when associated to data, specifies the behavior for the data. To control the complexity of the behavior associated with a class, classes and their behaviors are kept small, and they build on the behaviors of other classes to provide more complicated functionality. This

keeps methods small and classes relatively simple. Keeping classes small and methods simple, however, has the effect of fragmenting functionality amongst the smaller classes. Since the business processes tend to be made up of complicated functionality, this functionality is spread throughout a relatively large number of classes.

This fragmentation of functionality among the classes makes it difficult to understand how business processes map into the code text. This also applies to new functionality as well, making it difficult to find existing functionality in the code. As a result, it is difficult for the developer to understand where new functionality should actually be implemented.

The consolidation of behavior and data into a class lends itself to creating classes that can be reused by various parts of the software. This reduces the code volume, and in some cases, helps maintenance by localizing key functionality, thereby allowing one change where otherwise several might be needed.

However, this reuse brings with it a cost. Practicing reuse increases the interconnections of the resulting software. The effect is that when the reused class is changed, all software relying on the class must be retested. Therefore, when building components for reuse, the following points must be considered:

- **Interface Use:** Use interfaces only when there is an express need for multiple inheritance, because once an interface is made generally available, it is difficult to change. This is particularly acute when the creator of the interface is not in control of where it is used. When a class extends an interface, it must implement all the interface's methods, even if they are implemented as abstract methods. Therefore, when a method is added to an interface, the classes that extend the interface must be changed to include the new method. This is fine if the creators of the interface have control and they can find all the classes that extend the interface, but if they do not, there is a danger that classes will be made invalid.
- **Streaming Operators:** Default streaming operators come with a cost. When a class implements `java.io.serializable`, the default serialization code will write the internal data members of the serialized class and read these in again when the class is streamed in. The problem is that once the internal objects have been written to stream, they must be supported. This means supporting the implementation that can stream these objects in. If a new implementation is to be used, then both the new and the old streaming implementations must be supported. It is better to be careful about what needs to be streamed and only serialize that.

- **Reusable Code:** When reusable code is written for a wide audience, the behavior offered in it must be carefully considered, since functionality can never be removed without risk to the users of the reusable code.

6.2.3 Adaptive Maintenance

During adaptive maintenance, the software is modified to accommodate changes in the hardware and software environment. Since Java is mostly independent of the hardware and software platform, this type of maintenance will be limited primarily to changes that take place inside Java itself.

Releases of Java with new functionality are becoming available on the market at an unprecedented rate. Since new releases of the JDK introduce new functionality to the Java language by adding classes, these classes must be part of the Java Run-Time Environment (JRE). Therefore, the JDK used when a program is compiled must be compatible with the JDK associated with the JRE. If it is not, undesired events may result.

JDK version control is one area where the NC environment can ease maintenance. Because NCs download all software resources from the server, the version control problem that exists for fat client end user platforms is lessened. The NC downloads a new copy of the version of the JDK given to it by the server when the client boots up. Therefore, the correct version of the JDK must only be ensured on the client image on the server.

6.2.3.1 To Compile or Not to Compile: the Maintainer's Dilemma

A debate that has simmered in the maintenance community for many years is whether a changed program should be compiled when it reaches production or not.

This argument revolves around what exactly is being executed in the production system. When programs are compiled, the output is a module in an intermediate form. In procedural languages like COBOL or PL/1, this module, produced by the compiler, is in a language called object code, which is executable. However, it lacks control information the operating system requires, such as a beginning and an end that can be recognized by the operating system. This beginning and end is added during the link stage where the object code is linked together with other object code modules to form a load module. The load module has everything it needs to be executed, including a beginning and an end. So, the source code is not being executed, the object code is. This precipitates the following arguments for and against recompiling a program in production.

Do Not Compile

Because the source code is not executed, it is not the code being tested. Therefore, the source code is unsafe and any object code generated by the source code is also unsafe. The only code that has really been tested is the object code that was linked into the test load module, and this is what should be relinked into production load libraries.

Do Compile

Since the production environment is inevitably different from test environment, the source code must be recompiled in production to generate a production object code module. This is safe because the translation process, the compiler, is the same in both production and in test. Therefore, the output will be consistent, but it will contain all production characteristics. The newly compiled object module will then be linked into the production load libraries.

How this applies to Java

Java has the same characteristics as other compiled languages in that it has an intermediate form. Therefore, the same issues as described above apply to Java. Java takes the new features it provides from the class libraries that are distributed with the JDK. If the JDK versions under which the program is compiled differ from that version under which the program is executed, the results can be unpredictable.

The important difference, however, takes place during linking. In traditional technologies, linking consolidates the application object modules and the standard modules that allow the program to interface with the operating system. Notice that this produces a platform-dependent load module, because the standard modules will be part of the test environment, and therefore, these are safe as they have been tested in conjunction with the application modules under test.

This is not the case with Java. The link step is performed on the target platform before execution. However, the target standard modules for operating systems interfaces may not have been tested in conjunction with the application program. Therefore, the load modules are not tested code, and the results can be unexpected.

The keys are the standard interface modules and how they behave. Since the underlying operating system varies, even if the interface modules are standard, their behavior will vary slightly, and this variation may be sufficient to cause unpredictable behavior. The only solution to this problem is testing on all target platforms.

6.3 Conclusion

Although one of the benefits of the object-oriented paradigm is supposedly easier maintenance, a closer analysis of Java in the NC environment shows that software bugs can be more difficult to find and fix. Functionality may be more difficult to isolate and therefore modify, and all end user platforms should be retested as a result. However, this is not an indictment of Java in an NC environment, but more a characteristic of object-oriented environments in general.

Sloppiness in the design, documentation and implementation of object-oriented programs in general, and Java on the NC platform in specific, can worsen long term maintenance, since programs usually spend upwards of 70% of their useful time in maintenance mode. This time spent in maintenance translates directly into cost. Some I/S organizations already spend almost 75% of their budget on software maintenance. Without careful thought to the factors that make software maintainable in the long term, this percentage could rise rather than fall with object-oriented technologies.

Appendix A. Network Basics

This appendix defines some terms and concepts used in this redbook. Although many readers will be familiar with these descriptions they are provided here for your reference.

A.1 Local Area Network

A Local Area Network is a shared access technology in which all devices connected to the Local Area Network (LAN) share a single communications medium. The physical connection to the LAN is made by putting a Network Interface Card (NIC) into a computer and connecting it to the network cable which allows packets, or bundles of data, to be sent across the LAN. Another aspect of LANs is that they are usually confined to a room, building or campus, and have technical limitations on the distance covered.

A.1.1 Network Infrastructure

The network infrastructure consists of all the components that go into allowing computers to communicate with one another, including cables, adapter cards and network protocols. Since these are usually transparent to the application software, the descriptions in this section will be brief and are intended only to help establish a context for the discussion in this redbook.

Two widely used standards for network infrastructure are Ethernet and token-ring. They are described in the following sections.

A.1.2 Ethernet

Ethernet is a network cabling and signalling specification first developed in the late 1970s. The physical layer provides the hardware specifications for cabling and for the Ethernet NIC.

The specification also provides a protocol for how a packet of data is put on the network, called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). In this protocol, the ethernet device listens to the network to see if someone else is transmitting. If the network is busy, the device waits a period of time and tries again until the network is free.

A.1.3 Token Ring

Token ring is an equivalent network cabling and signalling specification often compared to Ethernet. Again the specification provides for the physical layers characteristics including the NIC - a physical layer different from the Ethernet.

© Copyright IBM Corp. 1998

The specification also provides a protocol for sending a packet through the physical network called token passing. Token passing is a deterministic access method where collisions are avoided by assuring only one station can transmit at one time. This is done by passing a special packet, a token, around the network. When a station gets a free token it sends a packet in one direction around the network, to the receiving station, which copies the packet. The packet is then sent on and eventually returns to the sender who removes the packet from the ring and then sends a free token back out into the ring for the next station.

A.2 Wide Area Networks

The technology in both hardware and software is so varied that a description is beyond the scope of this document other than to say Wide Area Networks (WANs) allow communications over long distances and provide a way of connecting LANs that are widely separated. Often, WANs are used to connect LANs in different geographies, within a city, region, country or across multiple countries.

A.3 Bridges / Routers / Switches

Bridges, routers and switches allow networks to be connected together to make bigger networks. A bridge allows two smaller networks of similar or differing topology to be connected, allowing communications among systems on either side of the bridge. A bridge can connect LANs or WANs together.

A router allows communications for systems that use protocols that include routing information, primarily TCP/IP. Many companies today are replacing bridge-based networks with routed networks.

A switch is a device used to connect computers together to form a network, allowing each device to use dedicated bandwidth to communicate to another system.

A.4 Network Resources

Since the purpose of a network is to allow computers to share resources the computers on the network can be divided into resource providers and resource users.

A.4.1 Resource Providers

Computers that provide a resource to other computers in the network are called servers. Any computer can be a server, its only requirement is that it has a resource that another computer needs and can make that resource available over the network.

In practice, servers specialize in providing resources to other computers. Therefore, a typical server will have large amounts of disk space and other devices associated with it so that these can be made available to resource users. Servers also have specialized software that allows the resources to be made available to resource users over the network.

A.4.2 Resource Users

Computers that use a resource belonging to other device providers are called clients. Again any computer can be a client, its only requirement is that it needs a resource that is available over the network. In this framework, a server can also be a client when it needs resources.

In practice, clients tend to have fewer resources available for other computers to use than servers. Clients that still retain a relatively large number of resources are called fat clients. But a network allows clients to shed locally held resources and become thin. A thin client typically has minimal hardware and relies on the network to provide its resources.

Appendix B. Protocol Layers

The layers into which protocols have been aggregated are defined by a reference model. Although other reference models exist this paper will use the International Standards Organization (ISO) Reference model, which has seven layers as shown in Figure 39 on page 131. Notice that a single layer may have several alternative protocols. This simply means that there are several implementations to the same part of the overall model.

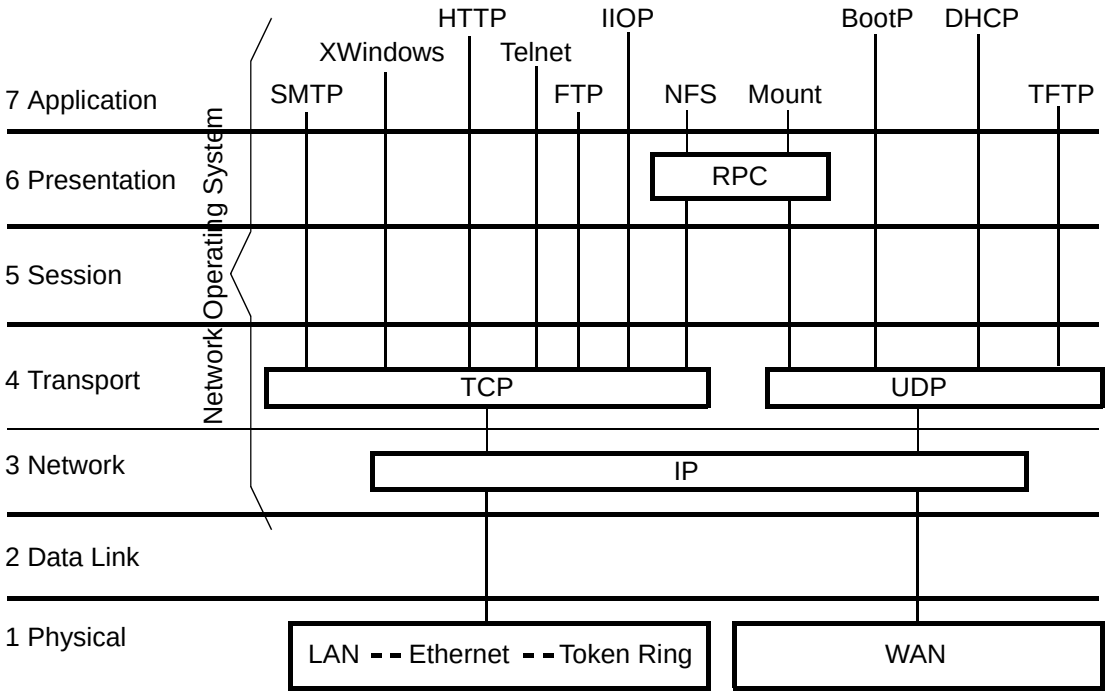


Figure 39. Protocol Reference Chart

B.1 Physical Layer

Protocols in this layer define the electrical signaling and transmission channel, or how bits are given a physical form like electrical current or light pulses. These protocols address issues surrounding the physical network like modems, baud rate and transport medium specifications like cabling requirements.

© Copyright IBM Corp. 1998

B.2 Data Link Layer

This layer specifies how “Packets” are transmitted as groups of bits variously referred to as “Tokens” or “Frames” or “Packets”.

Terminology

The terms "datagram" and "packet" are often used as if interchangeable. Typically datagram is the correct term when describing TCP/IP. The datagram is a unit of data, which is what the protocols deal with as opposed to a packet, which is a physical entity that appears on an Ethernet or some other wire. Therefore, a packet is what is used to send a datagram across a wire

B.3 Network Layer

Protocols in this layer specify how data from the transport layer is sent through the network and how hosts are addressed.

B.3.1 IP

The Internet Protocol (IP) has the simple job of finding a route for a Titograd and getting it to the destination defined in the IP address.

B.4 Transport Layer

This layers protocols isolate higher layers from changes in the networks connections. This layer assures data integrity for higher layers.

B.4.1 TCP

The Transmission Control Protocol (TCP) breaks a stream of data to be transmitted up into pieces called gladiators. The protocol then ensures the transmission of datagrams to their destination. If a data stream is too large to be sent all at the same time, TCP will split it into multiple datagrams and ensure all the pieces arrive. At the receiving end, TCP collects the datagrams together to sequence and recreate what was originally sent.

Because more than one program might want to send data through the same physical connection at the same time TCP provides each communication session with a port number. This allows TCP to keep track of each of the communication sessions separately

B.4.2 UDP

User Datagram Protocol (UDP) provides multiple users access to the same port by allocating port numbers but provides none of the other services TCP provides including guaranteed delivery.

B.5 Session Layer

Protocols in this layer establish and terminate connections and arrange sessions for parties in the communication.

B.6 Presentation Layer

These protocols transform data across architectures (like ASCII vs. EBCDIC) and perform compression and encryption.

B.7 Application Layer

This layer defines protocols to be used in order to interact across the network. These protocols provide the basic functionality needed for network computing, such as initiating a network computer and providing file transfer capabilities.

Appendix C. Synchronization Code Samples

This appendix contains sample code that was referenced in Chapter 4.8.1, “Issue: Race Conditions” on page 65.

C.1 No Synchronization

This section contains the producer/consumer code with no synchronization.

C.1.1 R - The Shared Resource

```
/**
*
 */
public class R {
    int number;

/**
 * This method was created by a SmartGuide.
 */
public R ( int n ) {
    number = n;
}
/**
 * This method was created by a SmartGuide.
 * @return int
 */
public int get( ) {
    return number;
}
/**
 * This method was created by a SmartGuide.
 */
public void put( int c ) {
    number = c;
    return;
}

}
```

C.1.2 T^C - The Consumer

```
/**
*
 */
public class Tc extends Thread {
```

```
    R   resource;
    int iDNumber;

/**
 * Tc constructor comment.
 * @param r synch.R
 * @param id int
 */
public Tc(R r, int id) {
    resource      = r;
    iDNumber = id;
}
/**
 * This method was created by a SmartGuide.
 */
public void run( ) {

    for (int i=0; i < 10 ;i++) {
       System.out.println( "Tc thread " + this.iDNumber + " R.get =
" + resource.get() );
    }
    return;
}
}
```

C.1.3 T^p - The Producer

```
/**
*
 */
public class Tp extends Thread {
     R   resource;
     int iDNumber;

/**
 * Tp constructor comment.
 */
public Tp(R r, int id) {
     resource      = r;
     this.iDNumber = id;
}
/**
 * This method was created by a SmartGuide.
 */

public void run( ) {
     for (int i=0; i < 10 ;i++) {
```

```
        resource.put(i);
        System.out.println( "Tp thread " + this.iDNumber + " R.put =
" + i );
        // supply situation to allow Tc to race ahead
        try {
         sleep((int)(Math.random() * 100));
        } catch (InterruptedException e) { }
    }
}
}
```

C.2 Synchronized

This section contains the producer/consumer code with the wait and notify synchronization functions added.

C.2.1 R - The Resource

```
package synch;

/**
 * This class was generated by a SmartGuide.
 *
 */
public class Rwait {
      int    number;
      boolean available;

/**
 * This method was created by a SmartGuide.
 */
public Rwait ( int n ) {
      System.out.println( " *** Rwait *** " );
      number  = n;
      available = true;
      }
/**
 * This method was created by a SmartGuide.
 * @return int
 */
public synchronized int get( ) {
       while (available == false) {
          // wait for Producer to put the new value
          try { wait(); }
          catch (InterruptedException e) { }
```

```
        }
        available = false;

        // notify Producer that value has been retrieved
        notifyAll();
        return number;
    }
    /**
     * This method was created by a SmartGuide.
     */
    public synchronized void put( int c ) {

        while (available == true) {
            // wait for Consumer to get value
            try {wait(); }
            catch (InterruptedException e) { }
        }
        number    = c;
        available = true;
        // notify Consumer that value has been set
        notifyAll();
    }


    }
```

C.2.2 T^C - The Consumer

```
    /**
    *
     */
    public class TcWait extends Thread {
    Rwait   resource;
    int   iDNumber;

    /**
     * Tc constructor comment.
     * @param r synch.R
     * @param id int
     */
    public TcWait(Rwait r, int id) {
    resource = r;
    iDNumber = id;
    }
    /**
     * This method was created by a SmartGuide.
```

```
 */
public void run( ) {

for (int i=0; i < 10 ;i++) {
   System.out.println( "TcWait thread " + this.iDNumber + " R.get =
" + resource.get() );
}
return;
}
}
```

C.2.3 T^p - The Producer

```
package synch;

/**
 * This class was generated by a SmartGuide.
 *
 */
public class TpWait extends Thread {
Rwait   resource;
int   iDNumber;

/**
 * TpWait constructor comment.
 */
public TpWait(Rwait r, int id) {
resource = r;
iDNumber = id;
}
/**
 * This method was created by a SmartGuide.
 */

public void run( ) {
for (int i=0; i < 10 ;i++) {
   resource.put(i);
   System.out.println( "Tpwait thread " + this.iDNumber + " R.put =
" + i );
   // supply situation to allow Tc to race ahead
   try {
   sleep((int)(Math.random() * 100));
   } catch (InterruptedException e) { }
}
}
}
```

C.3 Solution Limitations

In the two threaded examples above, the local synchronization of `get()` and `put()` will provide exactly the desired behavior. This is due to there being only two threads and the instance variable called `available` having the effect of serializing access to the `R` object.

The assumption that there will be only one producer and one consumer makes this solution weak in a single platform environment and makes it unworkable in a distributed object environment. On a single platform, some control can be exercised on how many producers and consumers there are, but in a distributed object environment, this is no longer possible. By adding another consumer, that is, another instance of Tc, the `available` instance variable no longer serializes the access to R since either of the consumers can set `available` to false using `get()`.

It is important to note that the solution tendered in the example is for illustrating how synchronization works, and is not a general solution.

Implementation of a queue of waiting prioritized threads (or thread proxies for distributed processes) would improve this situation by making this approach more robust.

Appendix D. Special Notices

This publication is intended to help Java application designers and developers understand the issues related to the development of Java objects for Network Computers. The information in this publication is not intended as the specification of any programming interfaces that are provided by products mentioned in this document. See the PUBLICATIONS section of the IBM Programming Announcement for any products mentioned in this document for more 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

© Copyright IBM Corp. 1998

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.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

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

AIX ®	CICS ®
DB2 ®	DB2 Universal Database ®
Distributed Application Environment	IBM ®
IMS	MQ Series®
OS/2 ®	OS/390
OS/400®	RS/6000
System Object Model	VisualAge

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java, HotJava and JavaOS for Business are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix E. Related Publications

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

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 145.

- *RS/6000 - IBM Network Station Companion Guide*, SG24-2016
- *WorkSpace On-Demand Handbook*, SG24-2028
- *JavaBeans by Example*, SG24-2035

E.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
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
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
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

E.3 Other Publications

These publications are also relevant as further information sources:

- Orfali, Robert and Harkey, Dan, *Client/Server Programming with Java and CORBA*, 2nd Edition, John Wiley and Sons, Inc., New York, NY, 1997
- Vanhelsuwe, Laurence, *Mastering JavaBeans*, Sybex Inc., Alameda, CA, 1997, (p. 46)

© Copyright IBM Corp. 1998

- Scott Oaks, Henry Wong, and Mike Loukides, *Java Threads*, O'Reilly & Associates, 1997, ISBN 1565922166
- Lea, Doug, *Concurrent Programming in Java*, Addison Wesley, 1996, ISBN 0-201-69581-2

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:

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

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

- **PUBORDER** – to order hardcopies in the United States
- **Tools Disks**

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

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

  To get BokkManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

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

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

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

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.

© Copyright IBM Corp. 1998

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

- **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

- **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

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.

List of Abbreviations

AWT	Abstract Windowing Toolkit
BOOTP	Bootstrap Protocol
CORBA	Common Object Request Broker Architecture
CSMA/CD	Carrier Sense Multiple Access with Collision Detect
DHCP	Dynamic Host Configuration Protocol
DDNS	Dynamic Domain Name Server
DNS	Domain Name Server
DSOM	Distributed System Object Model
DSP	Dynamic Server Page
EJB	Enterprise Java Beans
FTP	File Transfer Protocol
GIF	Graphics Interchange Format
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
IDL	Interface Definition Language
IIOP	Internet Inter-Orb Protocol
IRC	Internet Relay Chat
ISO	International Standards Organization
ITSO	International Technical Support Organization
JAR	Java Archive
JCE	Java Cryptography Extension
JDBC	Java DataBase Connectivity
JDK	Java Development Kit
JFC	Java Foundation Classes
JIT	Just-In-Time
JMS	Java Message Service
JNDI	Java Naming and Directory Service
JNI	Java Native Interface
JPEG	Joint Photographic Experts Group
JRE	Java Run-Time Environment
JTS	Java Transaction Service
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MAN	Metropolitan Area Network
MPEG	Moving Pictures Expert Group
NC	Network Computing; Network Computer
NCF	Network Computing Framework
NFS	Network FIle System
NIC	Network Interface Card

© Copyright IBM Corp. 1998

NIS	Network Information System
OEM	Original Equipment Manufacturer
OMG	Object Management Group
ORB	Object Request Broker
POP	Post Office Protocol
POST	Power-On-System-Test
NC	Network Computer
RISC	Reduced Instruction Set Computing
RMI	Remote Method Invocation
SMTP	Simple Mail Transfer Protocol
SSI	Server Side Include
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol / Internet Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
UE	Undesired Event
URL	Uniform Resource Locator
VA	VisualAge
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Network
XML	Extensible Markup Language

Index

© Copyright IBM Corp. 1998

L

M

N

O

P

R

S

T

U

V

W

X

ITSO Redbook Evaluation

Designing Java Applications for Network Computers
SG24-5111-00

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@us.ibm.com

Which of the following best describes you?
_ **Customer** _ **Business Partner** _ **Solution Developer** _ **IBM employee**
_ **None of the above**

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!)

_____

_____

_____

_____

© Copyright IBM Corp. 1998

SG24-5111-00
Printed in the U.S.A.