Table of Contents

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Chapter 1. Why Networked Java?

In the last 10 years, network programming has stopped being the province of a few specialists and become a core part of every developer's toolbox. Today, more programs are network aware than aren't. Besides classic applications like email, web browsers, and Telnet clients, most major applications have some level of networking built in. For example:

- Text editors like BBEdit save and open files directly from FTP servers.
- IDEs like Eclipse and IntelliJ IDEA communicate with CVS repositories.
- Word processors like Microsoft Word open files from URLs.
- Antivirus programs like Norton AntiVirus check for new virus definitions by connecting to the vendor's web site every time the computer is started.
- Music players like Winamp and iTunes upload CD track lengths to CDDB and download the corresponding track titles.
- Gamers playing Quake gleefully frag each other in real time.
- Supermarket cash registers running IBM SurePOS ACE communicate with their store's server in real time with each transaction. The server uploads its daily receipts to the chain's central computers each night.
- Schedule applications like Microsoft Outlook automatically synchronize calendars with other employees in the company.

In the future, the advent of web services and the semantic web is going to entwine the network ever more deeply in all kinds of applications. All of this will take place over the Internet and all of it can be written in Java.

Java was the first programming language designed from the ground up with networking in mind. Java was originally designed for proprietary cable television networks rather than the Internet, but it's always had the network foremost in mind. One of the first two real Java applications was a web browser. As the global Internet continues to grow, Java is uniquely suited to build the next generation of network applications. Java provides solutions to a number of problems—platform independence and security being the most important—that are crucial to Internet applications, yet difficult to address in other languages.

One of the biggest secrets about Java is that it makes writing network programs easy. In fact, it is far easier to write network programs in Java than in almost any other language. This book shows you dozens of complete programs that take advantage of the Internet. Some are simple textbook examples, while others are completely functional applications. One thing you'll notice in the fully functional applications is just how little code is devoted to networking. Even in network intensive programs like web servers and clients, almost all the code handles data manipulation or the user interface. The part of the program that deals with the network is almost always the shortest and simplest.

In brief, it is easy for Java applications to send and receive data across the Internet. It is also possible for applets to communicate across the Internet, though they are limited by security restrictions. In this chapter, you'll learn about a few of the network-centric applications that have been written in Java. In later chapters, you'll develop the tools you need to write your own network programs.

1.1. What Can a Network Program Do?

Networking adds a lot of power to simple programs. With networks, a single program can retrieve information stored in millions of computers located anywhere in the world. A single program can communicate with tens of millions of people. A single program can harness the power of many computers to work on one problem.

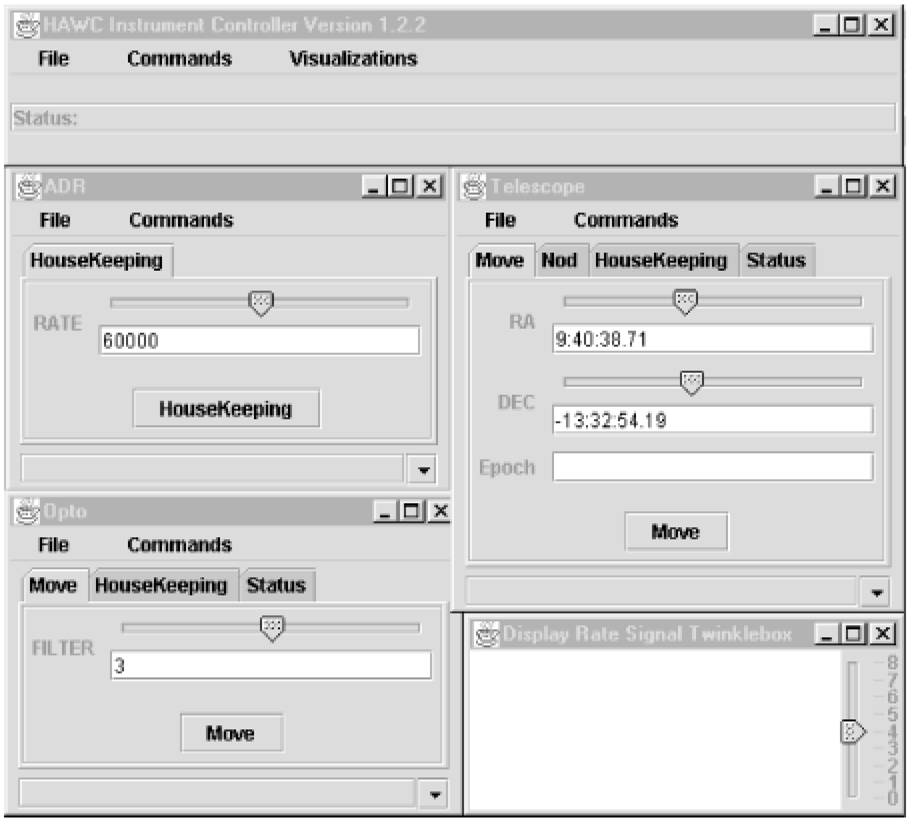
Network applications generally take one of several forms. The distinction you hear about most is between clients and servers. In the simplest case, clients retrieve data from a server and display it. More complex clients filter and reorganize data, repeatedly retrieve changing data, send data to other people and computers, and interact with peers in real time for chat, multiplayer games, or collaboration. Servers respond to requests for data. Simple servers merely look up some file and return it to the client, but more complex servers often do a lot of processing on the data before answering an involved question. Peer-to-peer applications such as Gnutella connect many computers, each of which acts as both a client and a server. And that's only the beginning. Let's look more closely at the possibilities that open up when you add networking to your programs.

1.1.1. Retrieve Data

At the most basic level, a network client retrieves data from a server. It can format the data for display to a user, store it in a local database, combine it with other data sources both local and remote, analyze it, or all of the above. Network clients written in Java can speak standard protocols like HTTP, FTP, or SMTP to communicate with existing servers written in a variety of languages. However, there are many clients for these protocols already and writing another one isn't so exciting. More importantly, programs can speak custom protocols designed for specific purposes, such as the one used to remotely control the High Resolution

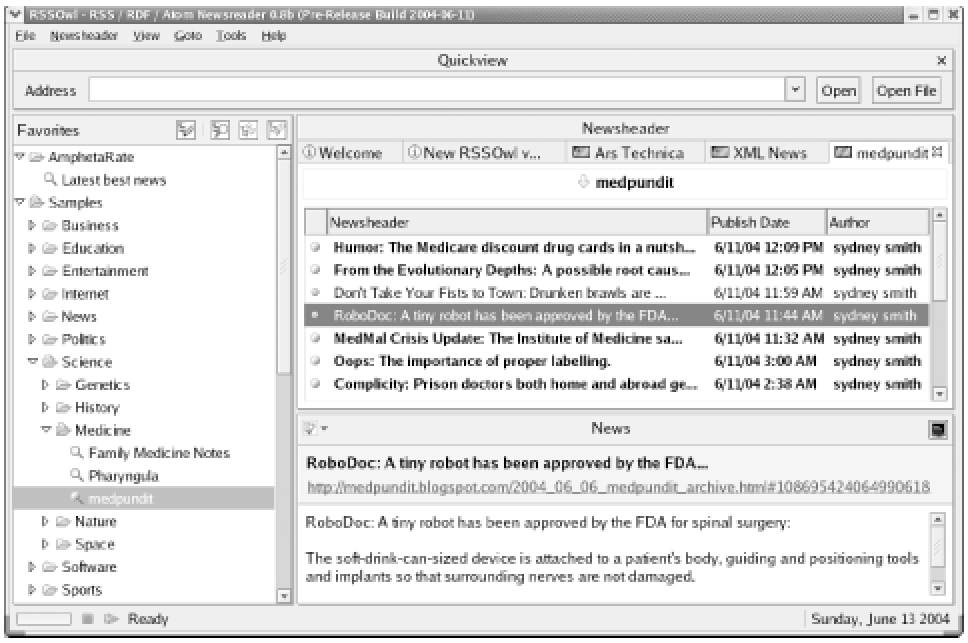
Airborne Wideband Camera (HAWC) on the Stratospheric Observatory for Infrared Astronomy (SOFIA). Figure 1-1 shows an early prototype of the HAWC controller.

Figure 1-1. The HAWC controller prototype



Also interesting is the use of existing protocols like HTTP to retrieve data that will be manipulated in new and unique ways. A custom network client written in Java can extract and display the exact piece of information the user wants. For example, an indexing program might extract only the actual text of a page while filtering out the HTML tags and navigation links. Of course, not every file downloaded from a web server has to be loaded into a browser window, or even has to be HTML. Custom network clients can process any data format the server sends, whether it's tab-separated text, a special purpose binary format for data acquired from scientific instruments, XML, or something else. Nor is a custom client limited to one server or document at a time. For instance, a summary program can combine data from multiple sites and pages. For example, RSS clients like RSSOwl, shown in Figure 1-2, combine news feeds in several different formats from many different sources and allow the user to browse the combined group. Finally, a Java program can use the full power of a modern graphical user interface to show this data to the user in a way that makes sense for the data: a grid, a document, a graph, or something else. And unlike a web browser, this program can continuously update the data in real time.

Figure 1-2. The RSSOwl newsreader is written in Java using the SWT API



medpundit

Newsheader	Publish Date	Author
Humor: The Medicare discount drug cards in a nutsh...	6/11/04 12:09 PM	sydney smith
From the Evolutionary Depths: A possible root caus...	6/11/04 12:05 PM	sydney smith
Don't Take Your Fists to Town: Drunken brawls are ...	6/11/04 11:59 AM	sydney smith
RoboDoc: A tiny robot has been approved by the FDA...	6/11/04 11:44 AM	sydney smith
MedMal Crisis Update: The Institute of Medicine sa...	6/11/04 11:32 AM	sydney smith
Oops: The importance of proper labelling.	6/11/04 3:00 AM	sydney smith
Complicity: Prison doctors both home and abroad ge...	6/11/04 2:38 AM	sydney smith

RoboDoc: A tiny robot has been approved by the FDA...

http://medpundit.blogspot.com/2004_06_06_medpundit_archive.html#108695424064990618

RoboDoc: A tiny robot has been approved by the FDA for spinal surgery:

The soft-drink-can-sized device is attached to a patient's body, guiding and positioning tools and implants so that surrounding nerves are not damaged.

Of course, not everything transmitted over HTTP is meant for humans. Web services allow machines to communicate with each other by exchanging XML documents over HTTP for purposes ranging from inventory management to stock trading to airline reservations. This can be completely automated with no human intervention, but it does require custom logic written in some programming language.

Java network clients are flexible because Java is a fully general programming language. Java programs see network connections as streams of data that can be interpreted and responded to in any way necessary. Web browsers see only certain kinds of data streams and can interpret them only in certain ways. If a browser sees a data stream that it's not familiar with (for example, a response to an SQL query), its behavior is unpredictable. Web sites can use server-side programs written in Java or other languages to provide some of these capabilities, but they're still limited to HTML for the user interface.

Writing Java programs that talk to Internet servers is easy. Java's core library includes classes for communicating with Internet hosts using the TCP and UDP protocols of the TCP/IP family. You just tell Java what IP address and port you want, and Java handles the low-level details. Java does not support NetWare IPX, Windows NetBEUI, AppleTalk, or other non-IP-based network protocols, but in the first decade of the new millennium, this is a non-issue. TCP/IP has become the *lingua franca* of networked applications and has effectively replaced pretty much all other general-purpose network protocols. A slightly more serious issue is that Java does not provide direct access to the IP layer below TCP and UDP, so it can't be used to write programs like ping or traceroute. However, these are fairly uncommon needs. Java certainly fills well over 90% of most network programmers' needs.

Once a program has connected to a server, the local program must understand the protocol the remote server speaks and properly interpret the data the server sends back. In almost all cases, packaging data to send to a server and unpacking the data received is harder than simply making the connection. Java includes classes that help your programs communicate with certain types of servers, most notably web servers. It also includes classes to process some kinds of data, such as text, GIF images, and JPEG images. However, not all servers are web servers, and not all data is text, GIF, or JPEG. As a result, Java lets you write protocol handlers to communicate with different kinds of servers and content handlers that understand and display different kinds of data. A web browser can automatically download and install the software needed by a web site it visits using Java WebStart and the Java Network Launching Protocol (JNLP). These applications can run under the control of a security manager that prevents them from doing anything potentially harmful without user permission.

1.1.2. Send Data

Web browsers are optimized for retrieving data: they send only limited amounts of data back to the server, mostly through forms. Java programs have no such limitations. Once a connection between two machines is established, Java programs can send data across the connection just as easily as they can receive from it. This opens up many possibilities.

1.1.2.1. File storage

Applets often need to save data between runs—for example, to store the level a player has reached in a game. Untrusted applets aren't allowed to write files on local disks, but they can store data on a cooperating server. The applet just opens a network connection to the host it came from and sends the data to it. The host may accept the data through HTTP POST, FTP, SOAP, or a custom server or servlet.

1.1.2.2. Massively parallel computing

There've always been problems that are too big for one computer to solve in a reasonable period of a time. Sometimes the answer to such a problem is buying a faster computer. However, once you reach the top of the line of off-the-shelf systems you can pick up at CompUSA, price begins to increase a lot faster than performance. For instance, one of the fastest personal computers you can buy at the time of this writing, an Apple PowerMac with two 2.5GHz processors, will set you back about $3,000 and provide speeds in the ballpark of a few gigaflops per second. If you need something a thousand times that fast, you can buy a Cray X1 supercomputer, which will cost you several tens of million dollars, or you can buy a thousand or so PowerMacs for only a few million dollars—roughly an order of magnitude less. The numbers change as the years go by. Doubtless you can buy a faster computer for less money today, but the general rule holds steady. Past a certain point, price goes up faster than performance.

At least since the advent of the cheap microcomputer a quarter of a century ago, programmers have been splitting problems across multiple, cheap systems rather than paying a lot more for the supercomputer of the day. This can be done informally by running little pieces of the problem on multiple systems and combining the output manually, or more formally in a system like Beowulf. There's some overhead involved in synchronizing the data between all the different systems in the grid, so the price still goes up faster than the performance, but not nearly as much faster as it does with a more traditional supercomputer. Indeed, cluster supercomputers normally cost about 10 times less than equally fast non-cluster supercomputers. That's why clusters are rapidly displacing the old style supercomputers. As of June 2004, just under 60% of the world's top 500 publicly acknowledged supercomputers were built from clusters of small, off-the-shelf PCs, including the world's third-fastest. There are probably a few more computers worthy of inclusion in the list hidden inside various government agencies with black budgets, but there's no reason to believe the general breakdown of architectures is different enough to skew the basic shape of the results.

When it comes to grid computing, Java is uniquely suited to the world of massively parallel clusters of small, off-the-shelf machines. Since Java is cross-platform, distributed programs can run on any available machine, rather than just all the Windows boxes, all the Solaris boxes, or all the PowerMacs. Since Java applets are secure, individual users can safely offer the use of their spare CPU cycles to scientific projects that require massively parallel machines. When part of the calculation is complete, the program makes a network connection to the originating host and adds its results to the collected data.

There are numerous ongoing efforts in this area. Among them is David Bucciarelli's work on JCGrid (http://jcgrid.sourceforge.net/), an open source virtual filesystem and grid-computing framework that enables projects to be divided among multiple worker machines. Clients submit computation requests to the server, which doles them out to the worker systems. What's unique about JCGrid compared to systems like Beowulf implemented in C is that the workers don't have to trust the server or the client. Java's security manager and byte code verifier can ensure the uploaded computation tasks don't do anything besides compute. This enables grids to be established that allow anyone to borrow the CPU cycles they need. These grids can be campus-wide, company-wide, or even worldwide on the public Internet. There is a lot of unused computing power wasting electricity for no reason at any given time of day on the world's desktops. Java networking enables researchers and other users to take advantage of this power even more cheaply than they could build a cluster of inexpensive machines.

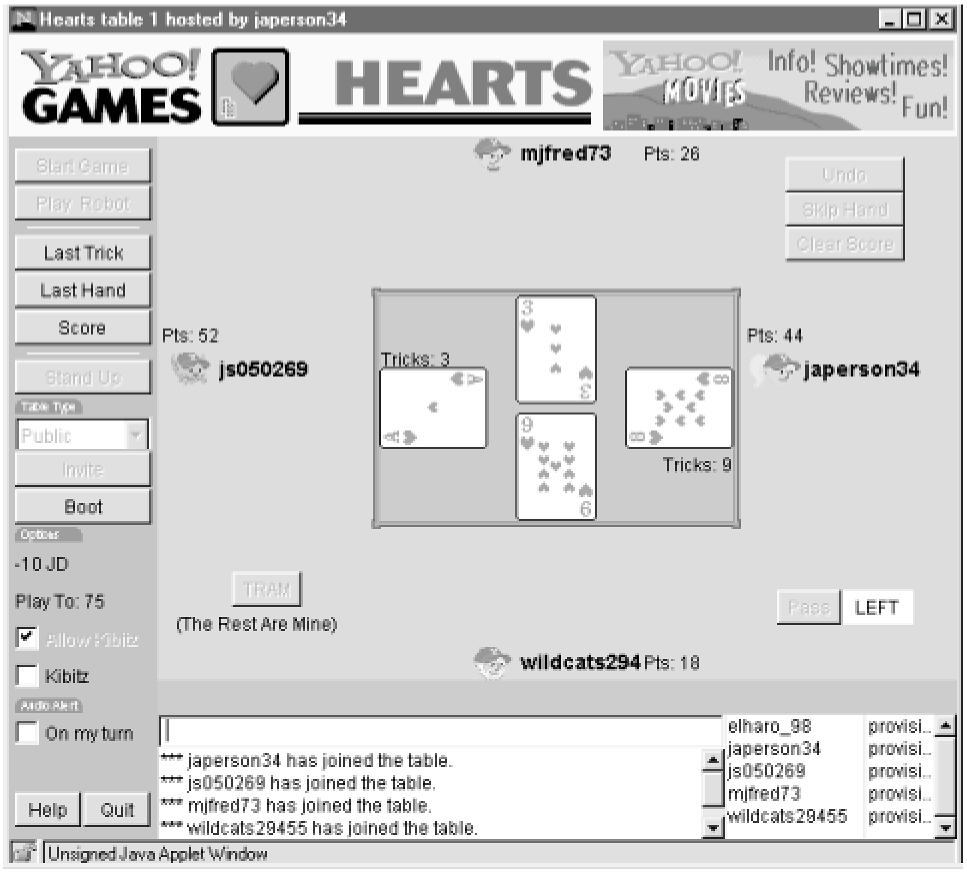
1.1.3. Peer-to-Peer Interaction

The above examples all follow a client/server model. However, Java applications can also talk to each other across the Internet, opening up many new possibilities for group applications. Java applets can also talk to each other, though for security reasons they have to do it via an intermediary proxy program running on the server they were downloaded from. (Again, Java makes writing this proxy program relatively easy.)

1.1.3.1. Games

Combine the easy ability to include networking in your programs with Java's powerful graphics and you have the recipe for truly awesome multiplayer games. Some that have already been written include Backgammon, Battleship, Othello, Go, Mahjongg, Pong, Charades, Bridge, and even strip poker. Figure 1-3 shows a four-player game of Hearts in progress on Yahoo. Network sockets send the plays back to the central Yahoo server, which copies them out to all the participants.

Figure 1-3. A networked game of Hearts using a Java applet from http://games.yahoo.com/



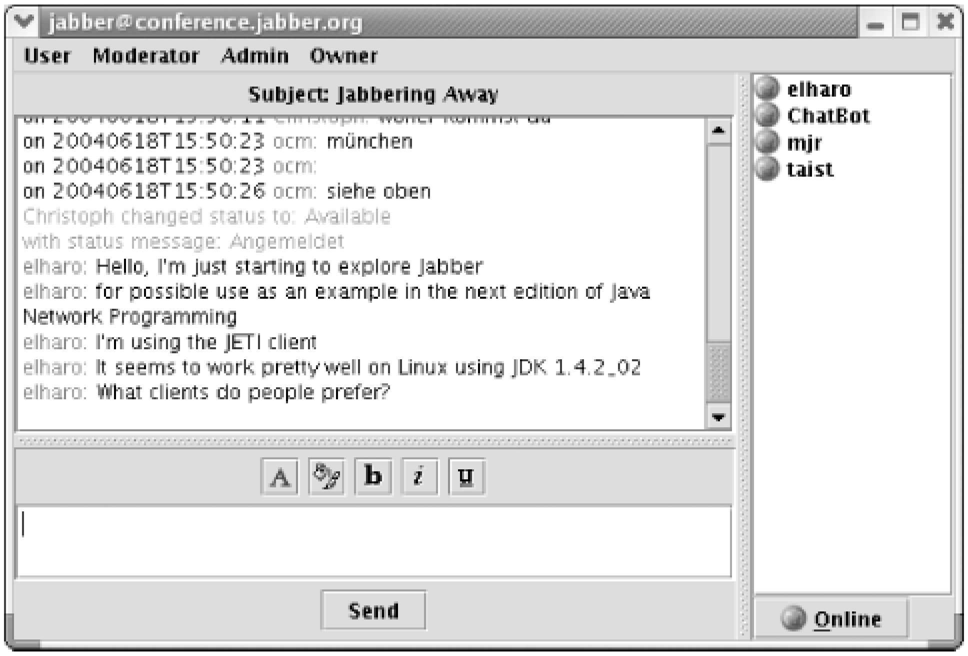
```
*** japerson34 has joined the table.
*** js050269 has joined the table.
*** mjfred73 has joined the table.
*** wildcats29455 has joined the table.
```

elharo_98	provisi...
japerson34	provisi...
js050269	provisi...
mjfred73	provisi...
wildcats29455	provisi...

1.1.3.2. Chat

Real-time chat probably isn't one of the first generation network applications (those would be file transfer, email, and remote login), but it certainly showed up by the second generation. Internet Relay Chat (IRC) is the original Internet chat protocol, and the cause of much caffeine consumption and many late nights in the dorm rooms of highly connected campuses in the 90s. More recently, the focus has shifted from public chat rooms to private instant messaging systems that connect users who already know each other. Network-wise, however, there isn't a huge amount of difference between the two. Perhaps the biggest innovation is the buddy list that allows you to know who among your friends and colleagues is online and ready to chat. Instant messaging systems include AOL Instant Messenger (AIM), Yahoo! Messenger, and Jabber. It isn't hard to find Java clients for any of these. Text typed on one desktop can be echoed to other clients around the world. Figure 1-4 shows the JETI client participating in a Jabber chat room.

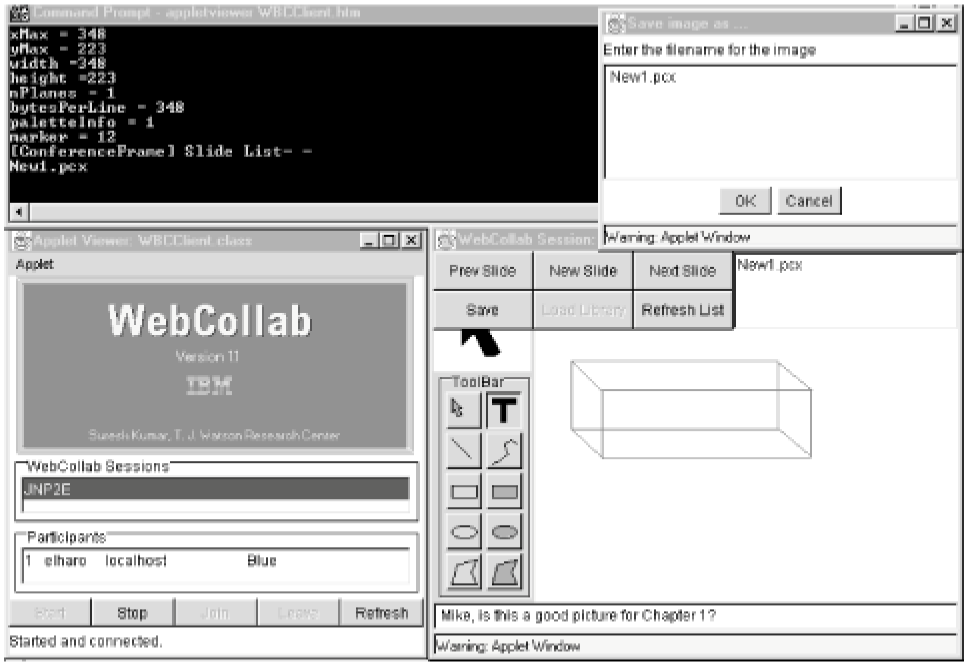
Figure 1-4. Networked text chat using Jabber



```
on 20040618T15:50:23 ocm: münchen
on 20040618T15:50:23 ocm:
on 20040618T15:50:26 ocm: siehe oben
Christoph changed status to: Available
with status message: Angemeldet
elharo: Hello, I'm just starting to explore Jabber
elharo: for possible use as an example in the next edition of Java
Network Programming
elharo: I'm using the JETI client
elharo: It seems to work pretty well on Linux using JDK 1.4.2_02
elharo: What clients do people prefer?
```

Java programs aren't limited to sending text. They can send graphics and other data formats, too. Adding a canvas with basic drawing ability to the chat program allows a whiteboard to be shared between multiple locations. A number of programmers have developed whiteboard software that allows users in diverse locations to draw on their computers. For the most part, the user interfaces of these programs look like any simple drawing program with a canvas area and a variety of pencil, text, eraser, paintbrush, and other tools. However, when networking is added, many different people can collaborate on the same drawing at the same time. The final drawing may not be as polished or as artistic as the Warhol/Basquiat collaborations, but it doesn't require the participants to all be in the same New York loft, either. Figure 1-5 shows several windows in a session of the IBM WebCollab program. WebCollab allows users in diverse locations to display and annotate slides during teleconferences. One participant runs the central WebCollab server that all the peers connect to, while conferees participate using a Java applet loaded into their web browsers.

Figure 1-5. WebCollab



```
xMax = 348
yMax = 223
width =348
height =223
nPlanes = 1
bytesPerLine = 348
paletteInfo = 1
marker = 12
[ConferenceFrame] Slide List- -
New1.pcx
```

Enter the filename for the image: New1.pcx

Warning: Applet Window

WebCollab Sessions: JNP2E

Participants: 1 elharo localhost Blue

Started and connected.

New1.pcx

Mike, is this a good picture for Chapter 1?

Warning: Applet Window

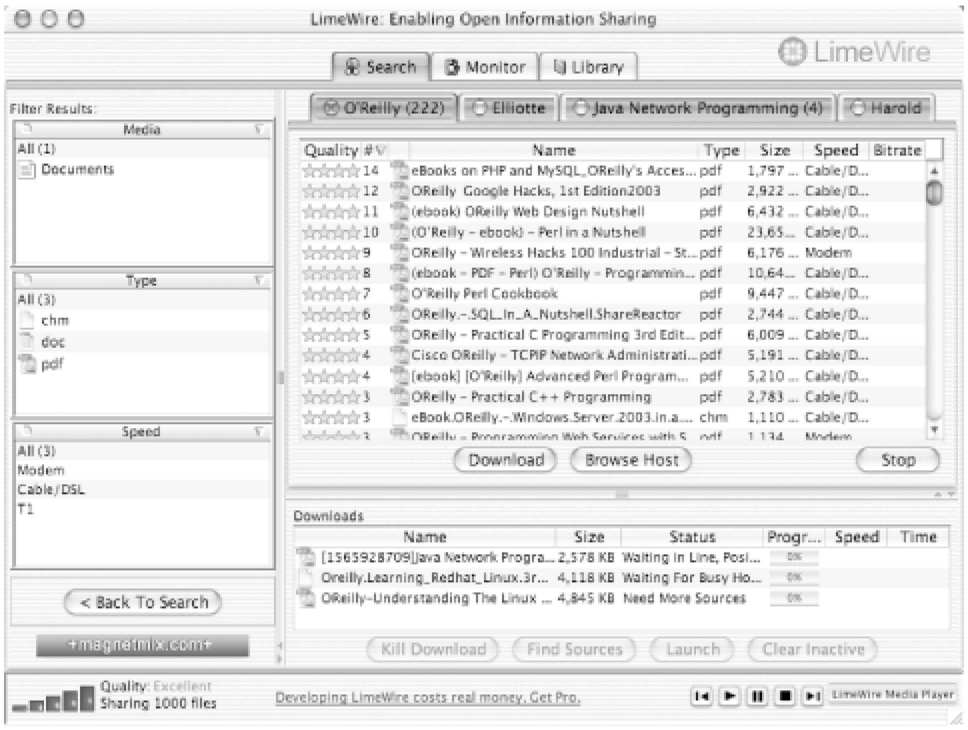
Peer-to-peer networked Java programs allow multiple people to collaborate on a document at one time. Imagine a Java word processor that two people, perhaps in different countries, can both pull up and edit simultaneously. More recently, the Java Media Framework 2.0 has added voice to the media that Java can transmit across the network, making collaboration even more convenient. For example, two astronomers could work on a paper while one's in New Mexico and the other in Moscow. The Russian could say, "I think you dropped the superscript in Equation 3.9," and then type the corrected equation so that it appears on both displays simultaneously. Then the astronomer in New Mexico might say, "I see, but doesn't that mean we have to revise Figure 3.2 like this?" and use a drawing tool to make the change immediately. This sort of interaction isn't particularly hard to implement in Java (a word processor with a decent user-interface for equations is probably the hardest part of the problem), but it does need to be built into the word processor from the start. It cannot be retrofitted onto a word processor that did not have networking in mind when it was designed.

1.1.3.3. File sharing

File transfer is one of the three earliest and most useful network applications (the other two being email and remote login). Traditionally, file transfer required a constantly available server at a stable address. Early Internet protocols such as FTP were designed under the assumption that sites were available 24/7 with stable addresses. This made sense when the Internet was composed mostly of multiuser Unix boxes and other big servers, but it began to fail when people started connecting desktop PCs to the network. These systems were generally only available while a single user was sitting in front of them. Furthermore, they often had slow dialup connections that weren't always connected, and hostnames and IP addresses that changed every time the computer was rebooted or reconnected. Sometimes they were hidden behind firewalls and proxy servers that did not let the outside world initiate connections to these systems at all. While clients could connect from anywhere and send files to anywhere, they couldn't easily talk to each other. In essence, the Internet was divided into two classes of users: high-bandwidth, stable, well-connected server sites, and low-bandwidth, sporadically connected client sites. Clients could talk to each other only through an intermediate server site.

In the last few years, this classist system has begun to break down. High-bandwidth connections through cable modems and DSL lines mean that even a $298 Wal-Mart PC may have bandwidth that would have been the envy of a major university 15 years ago. More importantly, first Napster and now Gnutella, Kazaa, Freenet, and BitTorrent have developed file transfer protocols that throw out the old assumptions of constant, reliable connectivity. These protocols allow sporadically connected clients with unstable IP addresses hidden behind firewalls to query each other and transfer files among themselves. Many of the best clients for these networks are written in Java. For instance, the LimeWire Gnutella client shown in Figure 1-6 is an open source pure Java application that uses a Swing GUI and standard Java networking classes.

Figure 1-6. LimeWire

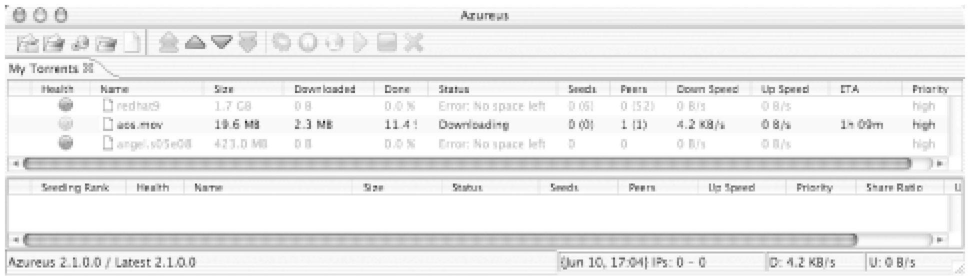


Quality	#	Name	Type	Size	Speed	Bitrate
☆☆☆☆☆	14	eBooks on PHP and MySQL_OReilly's Acces...	pdf	1,797 ...	Cable/D...	
☆☆☆☆☆	12	OReilly Google Hacks, 1st Edition2003	pdf	2,922 ...	Cable/D...	
☆☆☆☆☆	11	(ebook) OReilly Web Design Nutshell	pdf	6,432 ...	Cable/D...	
☆☆☆☆☆	10	(O'Reilly - ebook) - Perl in a Nutshell	pdf	23,65...	Cable/D...	
☆☆☆☆☆	9	OReilly - Wireless Hacks 100 Industrial - St...	pdf	6,176 ...	Modem	
☆☆☆☆☆	8	(ebook - PDF - Perl) O'Reilly - Programmin...	pdf	10,64...	Cable/D...	
☆☆☆☆☆	7	O'Reilly Perl Cookbook	pdf	9,447 ...	Cable/D...	
☆☆☆☆☆	6	OReilly.-.SQL_In_A_Nutshell.ShareReactor	pdf	2,744 ...	Cable/D...	
☆☆☆☆☆	5	OReilly - Practical C Programming 3rd Edit...	pdf	6,009 ...	Cable/D...	
☆☆☆☆☆	4	Cisco OReilly - TCPIP Network Administrati...	pdf	5,191 ...	Cable/D...	
☆☆☆☆☆	4	[ebook] [O'Reilly] Advanced Perl Program...	pdf	5,210 ...	Cable/D...	
☆☆☆☆☆	3	OReilly - Practical C++ Programming	pdf	2,783 ...	Cable/D...	
☆☆☆☆☆	3	eBook.OReilly.-.Windows.Server.2003.in.a...	chm	1,110 ...	Cable/D...	
☆☆☆☆☆	3	OReilly - Programming Web Services with S	pdf	1,134	Modem	

Name	Size	Status	Progr...	Speed	Time
[1565928709]Java Network Progra...	2,578 KB	Waiting In Line, Posi...	0%		
Oreilly.Learning_Redhat_Linux.3r...	4,118 KB	Waiting For Busy Ho...	0%		
OReilly-Understanding The Linux ...	4,845 KB	Need More Sources	0%		

The Gnutella protocol LimeWire supports is primarily used for trading music, pornography, and other copyright violations. The more recent BitTorrent protocol is designed for larger files such as Linux distro CD images. BitTorrent is designed to serve files that can be referenced from known keys provided by traditional sources like web sites, rather than allowing users to search for what's currently available. Another unique feature of BitTorrent is that downloaders begin sharing a file while they're still downloading it. This means hosting a large and popular file, such as the latest Fedora core release, doesn't immediately overwhelm a connection because only the first couple of users will get it directly from the original site. Most downloaders will grab much of the file from previous downloaders. Finally, BitTorrent throttles download bandwidth to match upload bandwidth, so leeching is discouraged. One of the best BitTorrent clients, Azureus (http://azureus.sourceforge.net/), shown in Figure 1-7, is written in pure Java.

Figure 1-7. Azureus



Health	Name	Size	Downloaded	Done	Status	Seeds	Peers	Down Speed	Up Speed	ETA	Priority
●	redhat9	1.7 GB	0 B	0.0 %	Error: No space left	0 (6)	0 (52)	0 B/s	0 B/s		high
●	aos.mov	19.6 MB	2.3 MB	11.4 %	Downloading	0 (0)	1 (1)	4.2 KB/s	0 B/s	1h 09m	high
●	angel.s05e08	423.0 MB	0 B	0.0 %	Error: No space left	0	0	0 B/s	0 B/s		high

Seeding Rank	Health	Name	Size	Status	Seeds	Peers	Up Speed	Priority	Share Ratio

Azureus 2.1.0.0 / Latest 2.1.0.0 [Jun 10, 17:04] IPs: 0 - 0 D: 4.2 KB/s U: 0 B/s

The free sharing of information between individuals without controllable server intermediaries terrifies all sorts of groups that would like to control the information whether for profit (the RIAA and the MPAA) or politics (various governments). Many of these have attempted to use legal and/or technological means to block peer-to-peer networks. The techies have responded with network protocols that are designed to be censorship-resistant through encryption and other means. One of the most serious is Ian Clarke's Freenet (http://freenet.sourceforge.net/). In this network protocol, encrypted files are divided up and duplicated on different computers that do not even know which files they're sharing. Furthermore, file transfers are routed through several intermediate hosts. These precautions make it extremely difficult for cybervigilantes, lawyers, and the police to find out who is sharing which files and shut them down. Once again, the primary Freenet implementation is written in Java, and most research and development has been done with Java as the language of choice.

1.1.4. Servers

Java applications can listen for network connections and respond to them, so it's possible to implement servers in Java. Both Sun and the W3C have written web servers in Java designed to be as fully functional and fast as servers written in C, such as the Apache HTTP server and Microsoft's Internet Information Server. Many other kinds of servers have been written in Java as well, including IRC servers, NFS servers, file servers, print servers, email servers, directory servers, domain name servers, FTP servers, TFTP servers, and more. In fact, pretty much any standard TCP or UDP server you can think of has probably been ported to Java.

More interestingly you can write custom servers that fill your specific needs. For example, you might write a server that stores state for your game applet and has exactly the functionality needed to let players save and restore their games, and no more. Or, since applets can normally only communicate with the host from which they were downloaded, a custom server could mediate between two or more applets that need to communicate for a networked game. Such a server could be very simple, perhaps just echoing what one applet sent to all other connected applets. WebCollab uses a custom server written in Java to collect annotations, notes, and slides from participants in the teleconference and distribute them to all other participants. It also stores the notes on the central server. It uses a combination of the normal HTTP and FTP protocols as well as its custom WebCollab protocol.

Along with classical servers that listen for and accept socket connections, Java provides several higher-level abstractions for client-server communication. Remote method invocation allows objects located on a server to have their methods called by clients. Servers that support the Java Servlet API can load extensions written in Java called *servlets* that give them new capabilities. The easiest way to build a multiplayer game server might be to write a servlet rather than an entire server.

1.1.5. Searching the Web

Java programs can wander through the Web, looking for crucial information. Search programs that run on a single client system are called *spiders*. A spider downloads a page at a particular URL, extracts the URLs from the links on that page, downloads the pages referred to by the URLs, and repeats the process for each page it downloads. Generally, a spider does something with each page it sees, from indexing it in a database to performing linguistic analysis to hunting for specific information. This is more or less what services like Google do to build their indices. Building your own spider to search the Internet is a bad idea because Google and similar services have already done the work, and a few million private spiders would soon bring the Net to its knees. However, this doesn't mean you shouldn't write spiders to index your own local Intranet. In a company that uses the Web to store and access internal information, a local index service might be very useful. You can use Java to build a program that indexes all your local servers and interacts with another server program (or acts as its own server) to let users query the index.

The purposes of *agents* are similar to those of spiders (researching a stock, soliciting quotations for a purchase, bidding on similar items at multiple auctions, finding the lowest price for a CD, finding all links to a site, and so on), but whereas spiders run a single host system to which they download pages from remote sites, agents actually move themselves from host to host and execute their code on each system they move to. When they find what they're looking for, they return to the originating system with the information, possibly even a completed contract for goods or services. People have been talking about mobile agents for years, but until now, practical agent technology has been rather boring. It hasn't come close to achieving the possibilities envisioned in various science fiction novels. The primary reason for this is that agents have been restricted to running on a single system—and that's neither useful nor exciting. In fact, through 2003, the only successful agents have been hostile code such as the Morris Internet worm of 1989 and the numerous Microsoft Outlook vectored worms.

These cases demonstrate one reason developers haven't been willing to let agents go beyond a single host: they can be destructive. For instance, after breaking in to a system, the Morris worm proceeded to overload the system, rendering it useless. Letting agents run on a system introduces the possibility that hostile or buggy agents may damage that system, and that's a risk most network managers haven't been willing to take. Java mitigates the security problem by providing a controlled environment for the execution of agents that ensure that, unlike worms, the agents won't do anything nasty. This kind of control makes it safe for systems to open their doors to agents.

The second problem with agents has been portability. Agents aren't very interesting if they can only run on one kind of computer. It's sort of like having a credit card for Nieman-Marcus: a little bit useful and has a certain snob appeal, but it won't help as much as a Visa card if you want to buy something at Sears. Java provides a platform-independent environment in which agents can run; the agent doesn't care if it's visiting a Sun workstation, a Macintosh, a Linux box, or a Windows PC.

An indexing program could be implemented in Java as a mobile agent: instead of downloading pages from servers to the client and building the index there, the agent could travel to each server and build the index locally, sending much less data across the network. Another kind of agent could move through a local network to inventory hardware, check software versions, update software, perform backups, and take care of other necessary tasks. A massively parallel computer could be implemented as a system that assigns small pieces of a problem to individual agents, which then search out idle machines on the network to carry out parts of the computation. The same security features that allow clients to run untrusted programs downloaded from a server lets servers run untrusted programs uploaded from a client.

1.1.6. Electronic Commerce

Shopping sites have proven to be one of the few real ways to make money from consumers on the Web. Although many sites accept credit cards through HTML forms, this method is inconvenient and costly for small payments of a couple of dollars or less. Nobody wants to fill out a form with their name, address, billing address, credit card number, and expiration date every day just to pay $0.50 to read today's *Daily Planet*. A few sites, notably Amazon and Apple's iTunes Music Store, have implemented one-click systems that allow customers to reuse previously entered data. However, this only really helps sites that users shop at regularly. It doesn't work so well for sites that typically only receive a visit or two per customer per year.

But imagine how easy it would be to implement this kind of transaction in Java. The user clicks on a link to some information. The server downloads a small applet that pops up a dialog box saying, "Access to the information at *http://www.greedy.com/* costs $0.50. Do you wish to pay this?" The user can then click buttons that say "Yes" or "No". If the user clicks the No button, then they don't get into the site. Now let's imagine what happens if the user clicks Yes.

The applet contains a small amount of information: the price, the URL, and the seller. If the client agrees to the transaction, then the applet adds the buyer's data to the transaction, perhaps a name and an account number, and signs the order with the buyer's private key. The applet next sends the data back to the server over the network. The server grants the user access to the requested information using the standard HTTP security model. Then it signs the transaction with its private key and forwards the order to a central clearinghouse. Sellers can offer money-back guarantees or delayed purchase plans (No money down! Pay nothing until July!) by agreeing not to forward the transaction to the clearinghouse until a certain amount of time has elapsed.

The clearinghouse verifies each transaction with the buyer and seller's public keys and enters the transaction in its database. The clearinghouse can use credit cards, checks, or electronic fund transfers to move money from the buyer to the seller. Most likely, the clearinghouse won't move the money until the accumulated total for a buyer or seller reaches a certain minimum threshold, keeping the transaction costs low.

Every part of this process can be written in Java. An applet requests the user's permission. The Java Cryptography Extension authenticates and encrypts the transaction. The data moves from the client to the seller using sockets, URLs, servlets, and/or remote method invocation (RMI). These can also be used for the host to talk to the central clearinghouse. The web server itself can be written in Java, as can the database and billing systems at the central clearinghouse, or JDBC can be used to talk to a traditional database like Informix or Oracle.

The hard part of this is setting up a clearinghouse, and getting users and sites to subscribe. The major credit card companies have a head start, although none of them yet use the scheme described here. In an ideal world, the buyer and the seller should be able to use different banks or clearinghouses. However, this is a social problem, not a technological one, and it is solvable. You can deposit a check from any American bank at any other American bank where you have an account. The two parties to a transaction do not need to bank in the same place.

1.1.7. Ubiquitous Computing

Networked devices don't have to be tied to particular physical locations, subnets, or IP addresses. Jini is a framework that sits on top of Java, easily and instantly connecting all sorts of devices to a network. For example, when a group of coworkers gather for a meeting, they generally bring a random assortment of personal digital assistants, laptops, cell phones, pagers, and other electronic devices with them. The conference room where they meet may have one or two PCs, perhaps a Mac, a digital projector, a printer, a coffee machine, a speaker phone, an Ethernet router, and assorted other useful tools. If these devices include a Java virtual machine and Jini, they form an impromptu network as soon as they're turned on and plugged in. (With wireless connections, they may not even need to be plugged in.) Devices can join or leave the local network at any time without explicit reconfiguration. They can use one of the cell phones, the speaker phone, or the router to connect to hosts outside the room.

Participants can easily share files and trade data. Their computers and other devices can be configured to recognize and trust each other regardless of where in the network one happens to be at any given time. Trust can be restricted; for example, all company employees' laptops in the room are trusted, but those of outside vendors at the meeting aren't. Some devices, such as the printer and the digital projector, may be configured to trust anyone in the room to use their services, but not allow more than one person to use them at once. Most importantly of all, the coffee machine may not trust anyone; but it can notice that it's running out of coffee and email the supply room that it needs to be restocked.

1.1.8. Interactive Television

Before the Web took the world by storm, Java was intended for the cable TV set-top box market. Five years after Java made its public debut, Sun finally got back to its original plans, but this time those plans were even more network-centric. The Java 2 Micro Edition (J2ME) is a stripped-down version of the rather large Java 2 API that's useful for set-top boxes and other devices with restricted memory, CPU power, and user interface, such as Palm Pilots. J2ME does include networking support, though for reasons of size, it uses a completely different set of classes called the Generic Connection Framework rather than the `java.net` classes from the desktop-targeted J2SE.

The Java TV API sits on top of J2ME to add television-specific features like channel changing and audio and video streaming and synchronization. TV stations can send programs down the data stream that allow channel surfers to interact with the shows. An infomercial for spray-on hair could serve a GUI program that lets the viewer pick a color, enter their credit card number, and send the order through the cable modem and over the Internet using their remote control. A news magazine could conduct a viewer poll in real time and report the responses after the commercial break. Ratings could be collected from every household with a cable modem instead of merely the 5,000 Nielsen families.

1.2. Security

Not all network programs need to run code uploaded from remote systems, but those that do (applets, Java WebStart, agent hosts, distributed computers) need strong security protections. A lot of FUD (fear, uncertainty, and doubt) has been spread around about exactly what remotely loaded Java code, applets in particular, can and cannot do. This is not a book about Java security, but I will mention a few things that code loaded from the network is usually prohibited from doing.

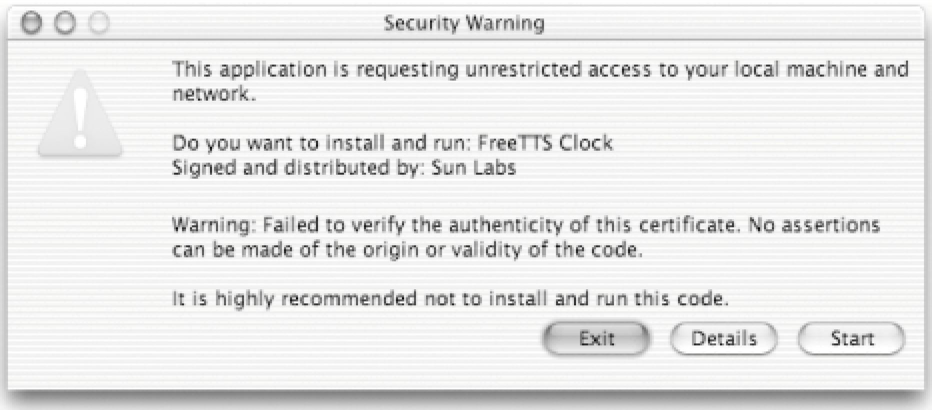
- Remotely loaded code cannot access arbitrary addresses in memory. Unlike the other restrictions in the list, which are enforced by a `SecurityManager`, this restriction is a property of the Java language itself and the byte code verifier.
- Remotely loaded code cannot access the local filesystem. It cannot read from or write to the local filesystem nor can it find out any information about files. Therefore, it cannot find out whether a file exists or what its modification date may be. (Java WebStart applications can actually ask the user for permissions to read or write files on a case-by-case basis.)
- Remotely loaded code cannot print documents. (Java WebStart applications can do this with the user's explicit permission on a case-by-case basis.)
- Remotely loaded code cannot read from or write to the system clipboard. (Java WebStart applications can do this with the user's explicit permission on a case-by-case basis.) It can read from and write to its own clipboard.
- Remotely loaded code cannot launch other programs on the client. In other words, it cannot call `System.exec( )` or `Runtime.exec( )`.
- Remotely loaded code cannot load native libraries or define native method calls.
- Remotely loaded code is not allowed to use `System.getProperty( )` in a way that reveals information about the user or the user's machine, such as a username or home directory. It may use `System.getProperty( )` to find out what version of Java is in use.
- Remotely loaded code may not define any system properties.
- Remotely loaded code may not create or manipulate any `Thread` that is not in the same `ThreadGroup`.
- Remotely loaded code cannot define or use a new instance of `ClassLoader`, `SecurityManager`, `ContentHandlerFactory`, `SocketImplFactory`, or `URLStreamHandlerFactory`. It must use the ones already in place.

Finally, and most importantly for this book:

- Remotely loaded code can only open network connections to the host from which the code itself was downloaded.
- Remotely loaded code cannot listen on ports below 1,024.
- Even if a remotely loaded program can listen on a port, it can only accept incoming connections from the host from which the code itself was downloaded.

These restrictions can be relaxed for digitally signed code. Figure 1-8 shows the dialog a Java WebStart application uses to ask the user for additional preferences.

Figure 1-8. Java WebStart requesting the user allow unlimited access for remotely loaded code



This application is requesting unrestricted access to your local machine and network.

Do you want to install and run: FreeTTS Clock
Signed and distributed by: Sun Labs

Warning: Failed to verify the authenticity of this certificate. No assertions can be made of the origin or validity of the code.

It is highly recommended not to install and run this code.

Even if you sign the application with a verifiable certificate so the warning is a little less blood-curdling, do not expect the user to allow connections to arbitrary hosts. If a program cannot live with these restrictions, you'll need to ask the user to download and install an application, rather than running your program directly from a web site. Java applications are just like any other sort of application: they aren't restricted as to what they can do. If you are writing an application that downloads and executes classes, carefully consider what restrictions should be put in place and design an appropriate security policy to implement those restrictions.

1.3. But Wait! There's More!

Java makes it possible to write many kinds of applications that have been imagined for years, but haven't been practical before. Many of these applications would require too much processing power if they were entirely server-based; Java moves the processing to the client, where it belongs. Other application types require extreme portability and some guarantee that the application can't do anything hostile to its host. While Java's security model has been criticized (and yes, some bugs have been found), it's a quantum leap beyond anything that has been attempted in the past and an absolute necessity for the mobile software we will want to write in the future.

Most of this book describes the fairly low-level APIs needed to write the kinds of programs discussed above. Some of these programs have already been written. Others are still only possibilities. Maybe you'll be the first to write them! This chapter has just scratched the surface of what you can do when you make your Java programs network aware. You're going to come up with ideas others would never think of. For the first time, you're not limited by the capabilities that other companies build into their browsers. You can give your users both the data you want them to see and the code they need to see that data at the same time.