

# 6

## Jini Technology Basics

**J**ini network technology is the new basis for networking in the J2ME. It was developed at Sun Microsystems as a way for software and hardware components to be integrated into one network, known as a *Jini technology-enabled network* (or *Jini network*).

### 6.1 Jini Network Terminology

A Jini network brings together many different types of devices with Java technology as the common base. In the past, electronic devices and computer programs had no easy way of networking together. Now, with the use of Jini technology, devices, such as cell phones, pagers, PDAs, and TV set-top boxes, can speak a common language.

Jini technology has four basic parts, making it simple to use:

- *Plug-and-Work model*: The power of Jini network technology to enable devices to connect to a network and interact with services without human administration
- *Lookup service*: Where Jini services announce their availability
- *Discovery protocol*: A process used to find the required lookup services in a local network
- *Proxy object*: Part of a Jini service that runs on the client

With these four simple parts, one can understand how an entire Jini federation operates. With the exception of the Plug-and-Work model, the parts are self-explanatory. The next section describes the Plug-and-Work model in detail.

## 6.2 The Plug-and-Work Model

The concept of a Plug-and-Work network is simple: A device should be able to connect to a network of services without human intervention, which is easier said than done in most cases. However, Jini network technology enables devices to break the ties to a human administrator. Instead, it enables a device to discover for itself what is on a network.

The Plug-and-Work model is based on two concepts: code downloading and remote interfaces:

- *Code downloading:* The ability of data and functionality to be automatically downloaded to a device so that it does not require updating by a human. Also, code downloading extends Java technology class type checking, enabling subclassing to extend functionality far into the future and into many subtypes that may not be designed or developed yet.
- *Remote interfaces:* Java technology interfaces negate the need for strict dependency on a service implementation. Instead, the well-known interface describes the dynamically downloaded code, which enables a device to dynamically learn how to use a new service.

With code downloading and remote interfaces, a device can go about its business of utilizing a service without being tightly coupled to an implementation or relying on a human to program it correctly to access the service.

When a device can download code on its own, there is no need to upgrade it manually with the latest software off the Web. Each time a device uses a service, it gets the service's most up-to-date implementation, containing all the bug fixes and enhancements available up to that moment. The data and behavior are downloaded together, enabling a late binding to occur from the device to the service at the last minute. This enables a device to evolve as the services it uses are improved over time, and it enables the device to get the latest updated service.

In the case of a stock quote service, the device needs to know only the interface to the service to access the stock price. If the Yahoo! stock quote service, for example, is temporarily disconnected, the device can automatically discover a new stock quote service with the same interface and use it without having to be reprogrammed.

The strengths of the Plug-and-Work model are in the way it handles network failures and its ability to switch over to any service that uses a well-known interface on the network.

### 6.3 Object Protocol

With Jini network technology, one deals with an object exchange protocol; this means that software objects are passed back and forth between entities. With older protocols, only data is passed back and forth. In all other communication protocols, data is used as the flow of information. The objects that transfer with Jini technology contain the encapsulation of data and methods to interact with the object's data, which is a more faithful implementation of the object-oriented distributed computing model. With objects being passed back and forth, encapsulated data and methods are conveniently grouped together no matter where they show up on the network.

When a computer system on one side of the world gets a proxy object from a Jini lookup service on the other side, all the data and method interfaces that the system needs to interact with that object have been completely transferred to it.

A true distributed computing environment cannot exist without being able to easily transport objects on a network. With Jini technology, for the first time, all the objects of a programming language like Java are distributed to any system in the Jini federation.

### 6.4 Dynamic Network Edges

The dynamic nature of Jini network technology is a great way for devices on the "edge" of a network to handle change. The *edge* of a network is where the end-user exists and is the point farthest away from where services originate.

An example of an end-user on the edge of a network is a cell phone user. He or she is the endpoint of the services that the cell phone provider makes available. Being so far away from the service provider means that the end-user on the edge of a network is isolated from provider-assisted intervention if something goes wrong. In addition, the edge of a network is far away in network distance from updates to the cell phone. A cell phone user is usually resistant to having anything manually done to the device itself because a cell phone should be a self-contained unit and should be able to maintain the connection to the service provider in a robust manner without hassle.

The infrastructure of Jini network technology enables a cell phone to be deployed once without requiring constant service provider intervention to update

software or to fix bugs. Jini network technology enables code to be downloaded (for example, through Remote Method Invocation—RMI) without interfering with client code already on the phone.

When older legacy devices are mixed with new devices (as on many cell phone networks today), and robust interaction is required between these devices, Jini network technology really shines. It is not necessary to know all the possible edges of the network nor to intervene or administer the client code of those edges. It is necessary only to be able to download proxy code to the edges, so less knowledge is embedded in the edges, enabling the creation of a network that does not break down and can scale expeditiously at the edges.

Dynamic network edges are the reality of a quickly evolving environment such as the world of cell phone users. When a cell phone or a PDA that links to back-end wireless services needs to know what the service can do and can access that service by downloading it, much complexity is hidden from the device. This is the main advantage of Jini network technology.

## 6.5 Leases

Another aspect of Jini network technology is service leases. When one creates a Jini service, one must register it and request a lease from the lookup service. The lease returned from the lookup service identifies the amount of time left before the registry entry expires for that particular service, which prevents a Jini service from becoming stale as an invalid entry in the lookup service registry. An invalid registry entry can occur when the service goes down or when the network connection is lost.

Leased Jini services enable a robust network in a Jini federation. Failure of services on the network does not cause the entire network to fail. Leasing is part of the general programming model of Jini technology. One can also use leases to manage other resources on the network.

## 6.6 Attributes

The robustness of the Jini federation is also enhanced by the lookup service's ability to use matching for service lookup, which adds flexibility to the system, in addition to its inherent reliability.

Each Jini proxy object has one or more attributes. These attributes are type/value pairs used to distinguish one Jini service from another. For example, one

can have a Jini service attribute with a value that describes the location of the service it represents, such as “second floor, building #1.”

The Jini client can set a matching pattern with the Jini lookup service of a set of attributes or of service types with the values specifically defined or intentionally left blank (the latter means that the client does not care what the value is). The Jini lookup service finds the matching services according to that set of attributes and/or service types and returns them to the client. This is a powerful way for Jini clients to match on generic services (for example, a fax service) or on very specific services (for example, a fax service on a certain floor of a specific building).

## 6.7 Groups

A Jini lookup group is a way to categorize Jini services. If a network has many lookup services, one may want to select a particular one. Doing this enables specific lookup services to be limited to only a small number of specialized services.

For example, consider a service for a television in a house. One may want to register it with the entertainment group on a certain lookup service of a home network. Another lookup service that is assigned to the heating and ventilation group can ignore the service requests from the entertainment group, and the entertainment group can ignore the announcements coming from the heating and ventilation lookup service. Groups facilitate an organized service structure. Enhanced organization provides efficiency in terms of client interactions with the specific services.

## 6.8 Lookup and Discovery

Figure 6.1 shows the basic parts of a Jini federation as it first begins to connect. Initially, a service must be created with a public API—for example, a pager acting as a client and connecting to a created StockQuote service. Once the public API is established for the StockQuote service, an implementation can be made based on its specification. For the example in this chapter, the StockQuote service is implemented as wireless. In Figure 6.1, one sees a Jini service proxy object (`WirelessStockQuoteProxy`) being registered with the Jini lookup service through the `JoinManager()` constructor method. The lookup service keeps the `WirelessStockQuoteProxy` object in its registry for later download by a Jini client. The `WirelessStockQuoteService` is the implementation of the `WirelessStockQuoteProxy` interface.

44 JINI TECHNOLOGY BASICS

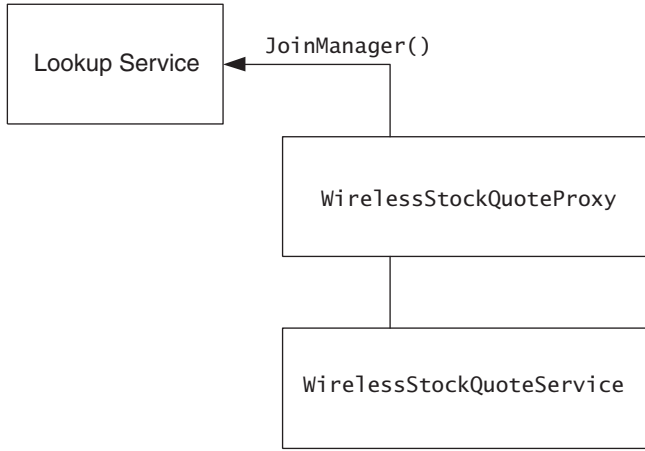


Figure 6.1 Registration of a Jini service proxy object

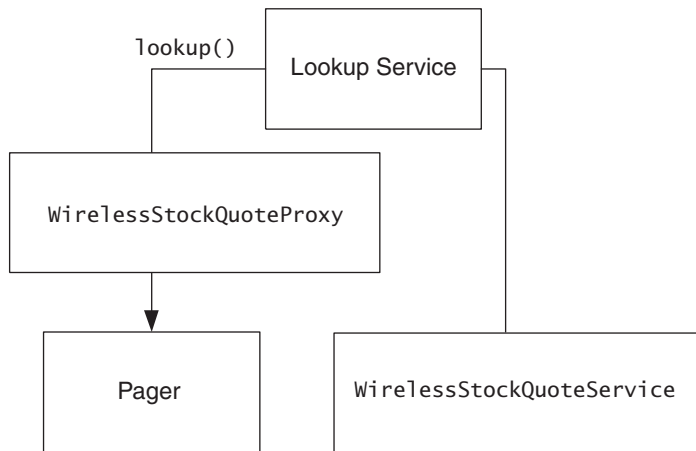
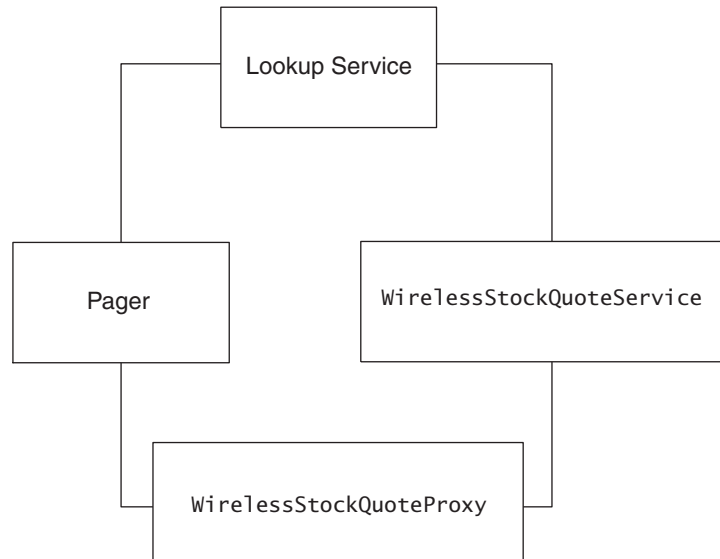


Figure 6.2 Lookup service is discovered, and proxy is downloaded from the lookup service by a Jini client.

For Jini technology, code downloading is crucial because that is how dynamic behavior is formed. With code mobility, there is not just one computer system but a network of computer systems that run a program.

In Figure 6.2, one can see how a Jini client (Pager) first discovers the Jini lookup service and then is able to download the Jini proxy object with a lookup



**Figure 6.3** The Jini client can use the proxy object directly to communicate back to the Jini service.

service `lookup()` method. `WirelessStockQuoteProxy` is downloaded to interact with `WirelessStockQuoteService`. A coding example is given in Section 6.9, along with the details of this process. For now, the basic idea is presented in the figures shown here.

In Figure 6.3, the `Pager` object can use the `WirelessStockQuoteProxy` object to communicate directly to `WirelessStockQuoteService`. The Jini lookup service has finished its job at this point by providing the correct proxy object to the Jini client that requested a connection to the Jini stock quote service. The `Pager` object calls the methods in the `WirelessStockQuoteProxy` object to retrieve the specified stock prices from the `WirelessStockQuoteService` object that can exist across the network on some other computer.

The Jini lookup service behaves as a control center that matches client requests with the registered proxy object for the requested service. This simple process frees the J2ME device developer from reinventing shared communication functionality, which makes writing Jini technology software much easier when dealing with distributed objects that wish to communicate with each other.

## 6.9 Simple Code Example

The following simple code example starts a service on a J2ME technology-enabled device and allows it to be a full RMI service that can be called from anywhere on the network by an RMI client.

```
import java.io.*;
import java.lang.*;

import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

import jini.net.JoinManager;
import net.jini.core.discovery.*;
import net.jini.core.entry.*;
import net.jini.core.lookup.*;
import net.jini.lookup.entry.*;
import com.sun.jini.lookup.entry.BasicServiceType;

//
// To create a RMI service, make sure to do the following:
// 1. Extend the class using java.rmi.server.UnicastRemoteObject
// 2. Implement your own remote interface which in turn extends
//    Remote (in this case WirelessStockInterface)
// 3. Implement the RMI Service class using Serializable interface
// 4. Add your functionality to the implementation of the remote
//    interface (in this case getStockQuote())
// 5. Register this RMI Service with a Jini lookup service
//    (in this case see main())
//
public class WirelessStockQuoteService extends UnicastRemoteObject
    implements WirelessStockInterface, Serializable {

    public WirelessStockQuoteService() throws RemoteException {
        super();
    }

    public String getStockQuote(String symbolName) {
        String retStr = null;

        // Set RMI security manager if there isn't one
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
    }
}
```

```
// Functionality here to retrieve the stock quote
//      of symbolName
// ...

return(retStr);
}

public static void main (String[] args) {
WirelessStockQuoteService wStockQuoteService = new
    WirelessStockQuoteService();
ServiceInfo serviceInfo = new ServiceInfo("PRODUCT", "MANUFACTURER",
    "VENDOR", "VERSION", "", "");
BasicServiceType basicServiceType = new
    BasicServiceType("WirelessStockQuoteService");
Entry entry[] = new Entry[] {serviceInfo, basicServiceType};
try {
    JoinManager joinManager = new JoinManager
        (wStockQuoteService, entry,
        new ServiceIDHandler(), new LeaseRenewalManager());
} catch (java.io.IOException ioe) { }
}
```

The first step is to register the RMI service using RMI Optional Package, which is done by calling the following Jini technology API:

```
JoinManager joinManager = new JoinManager
    (wStockQuoteService, entry,
    new ServiceIDHandler(), new LeaseRenewalManager());
```

## 6.10 Proxy

The term *proxy* is used in many ways in discussions of computer programming. In the context of RMI technology, the term *proxy* means a computer programming object that contains data and programming behavior that enables it to run on a computer system separate from the originating calling object. This enables many computer systems to run in distributed programming environments instead of running a single program on just one computer system.

## Quick-Start Guide for the Jini Technology Environment

Here is a quick-start guide for how to run the most important pieces of the Jini technology environment. First, make sure to download the most recent version of the Jini Technology Starter Kit from [www.sun.com/software/communitysource/jini/download.html](http://www.sun.com/software/communitysource/jini/download.html).

This book uses the Jini Technology Kit version 1.2. Follow the instructions for how to install the kit for a Linux system. Note that you must also have a J2SE version 1.3 environment on your system. After those two environments are installed, bring up the Jini technology environment with the following instructions:

1. Start the HTTPD server:

```
java -jar <toplevel>/jini1_2/lib/tools.jar -port 8080 -dir  
<toplevel>/jini1_2/lib &
```

2. Remove the RMI daemon log if it exists:

```
rm -rf ~/log
```

3. Start the RMI daemon:

```
rmid -J-Dsun.rmi.activation.execPolicy=none &
```

4. Start the Jini lookup service:

```
java -jar -Djava.security.policy=<toplevel>/jini1_2/example/  
lookup/policy.all <toplevel>/jini1_2/lib/reggie.jar http://  
<your_web_server>:8080/reggie-dl.jar <toplevel>/jini1_2/  
example/lookup/policy.all /tmp/reggie_log public &
```

5. Start the RMI registry (optional if running RMI services)

```
rmiregistry &
```

6. Run the Jini services browser example:

```
java -cp <toplevel>/jini1_2/lib/jini-core.jar:<toplevel>/  
jini1_2/lib/jini-examples.jar:<toplevel>/jini1_2/lib/  
reggie.jar -Djava.security.policy=<toplevel>/jini1_2/example/  
browser/policy -Djava.rmi.server.codebase=http://  
<your_web_server>:8080/jini-examples-dl.jar  
com.sun.jini.example.browser.Browser &
```

### 7. Start a service (for example, PrintService):

```
cd <your_example_workspace>/work/jini/src/PrintService
java -cp .:<your_example_workspace>/work/jini/src/
PrintService:<toplevel>/jini1_2/lib/jini-core.jar:<toplevel>/
jini1_2/lib/sun-util.jar:<toplevel>/jini1_2/lib/jini-
ext.jar:<toplevel>/jini1_2/lib/reggie.jar -
Djava.rmi.server.codebase=http://<your_web_server>/
<your_example_workspace>/jini/src/PrintService/ -
Djava.security.policy=<your_example_workspace>/work/jini/src/
PrintService/po.policy Print |& tee ~/jini_printservice.log &
```

## 6.11 Summary

This chapter gave a brief overview of the Jini network technology. The basics of Jini technology were covered, including the Plug-and-Work model lookup service, the discovery protocol, and the proxy object. How leases work and how attributes can be used to define a Jini service were also covered. Finally, this chapter described how lookup groups logically organize Jini services.

Other materials are available for more information on RMI and how to best use RMI Optional Package with J2ME technologies. One place to start is the Web page at [java.sun.com/products/jdk/rmi/index.html](http://java.sun.com/products/jdk/rmi/index.html), where you will find information regarding the larger version of RMI for J2SE technology. The Web site relates to J2ME technology in that it enables a J2ME technology-enabled device to run as a full RMI client.