CHAPTER 1

INTRODUCTION

1.1 Background

Jini\textsuperscript{TM} which is based on Java platform is a network technology from Sun Microsystems. Jini\textsuperscript{TM} technology is designed to provide a simple infrastructure for devices to deliver services in a network and make themselves available for use [Nicolai]. Jini\textsuperscript{TM} technology enables a “plug and play” network. Jini\textsuperscript{TM} technology will run on any network with at least one Java virtual machine.

Unfortunately, not every device supports a full Java 2 Standard Edition (J2SE) virtual machine, even if a device does support Java\textsuperscript{TM} technology (J2ME) such as PDA, it may not have enough memory to support a full Jini\textsuperscript{TM} implementation [Day]. Devices with limited memory are not able to run full java virtual machine so that the Jini\textsuperscript{TM} technology advantages are not accessible for these devices yet.

For these devices which cannot run java virtual machine, a third party is introduced. Jini\textsuperscript{TM} surrogate architecture is an attempt to integrate devices that could not directly participate on Jini\textsuperscript{TM} network. Jini\textsuperscript{TM} surrogate architecture specification defines a surrogate host which sites in the host capable machine, provides an execution environment for downloaded code on behalf of the device. Under this model, a mobile
device like a PDA uploads a device surrogate which is a Java object, along with its codebase, into the surrogate host. The surrogate host provides the J2SE environment which allows the device surrogate participating Jini™ “community” on the device’s behalf, registering the device’s services with the Jini™ lookup service, executing other service’s (device) proxy objects, and communicating back to the device. The surrogate architecture lowers the requirements that a device must meet to participate in a Jini™ network [Arnold].

Madison is the surrogate host developed by Sun and a number of participating companies. The Madison software includes a Jini™ technology surrogate host and an IP surrogate that conform to both the Jini™ Technology Surrogate Architecture specification and Jini™ technology IP interconnect specification. Madison host provides the environment for executing surrogate, manages the surrogate life cycle, and provides other host resources to assist the components in the architecture.

Figure 1 shows the surrogate architecture

![Figure 1: Surrogate architecture components](adapted-from-Thompson)
1.2 Problem

This project is part of larger research effort, the Madison framework to connect Jini™ with PDA. The other team member’s implementation and study is based on Jini™’s Madison surrogate architecture. To be able to use Madison host as a technological platform, it is necessary first to explore and examine the technology in greater detail, to experiment with its use. The required software must be downloaded and installed, and an initial test must be done. That means exploring and studying Madison host become very important part of the whole research, and whether Madison host can be set up is essential for later work. The goal of this study is to evaluate the use of Madison surrogate host in the Microsoft Windows platform as a framework for other students to use in future projects. This paper provides detailed information about the Madison software for the rest of the research team.

1.3 Scope

This study will be concerned only with discussion of the Jini™ surrogate architecture, research of the Madison software, including the explanation of the Madison software contents, the steps for setting up Madison, and running the sample program in Microsoft Windows platform only. No attempt will be made to implement the Madison application.

1.4 Approach

- Get the Madison software from the Internet, extract it and install it.
- Install other necessary software.
- Configure the Madison, set up Madison host on Microsoft Windows platform by writing the scripts.
- Test Madison by running a simulated device and the surrogate samples.
1.5 Equipment

The computer software necessary for study includes as the following:

- Windows 2000 professional, Version 5.00.2195
- Java 2 SDK, Standard Edition, Version 1.4.0-beta3 (current at ASU East campus)
- Jini™ starter kit (Jini1_2)
- Madison 1_0_1(jini_surr_madison1_0_1)

1.6 Sequence of Presentation

This report is organized in four chapters.

- Chapter 1 introduces the background, the objective, and the problem involved.
- Chapter 2 discusses the concepts of the Jini™ technology, the Jini™ surrogate architecture.
- Chapter 3 presents the details of the Madison software, including the Madison components, the steps of setting up Madison, and the results of running sample program.
- Chapter 4 provides the conclusion and suggestions for future study.
CHAPTER 2
LITERATURE REVIEW

Jini™ is designed to allow any kind of Jini™-enabled device plug into a TCP/IP network, automatically announce it presence, and be available to anyone who can reach it [Jini™ network]. Jini™ technology requires at least one full J2SE virtual machine. For many personal devices, limited memory is the problem for participating in Jini™ technology. Jini™ surrogate architecture addresses these devices [The Jini™]. This chapter introduces the Jini™ technology and takes closer look at the Jini™ surrogate architecture.

2.1.1 The Jini™ Vision

The Jini™ technology vision is: every device in the Jini™ “community” is a service, once plugged into the Jini network, any Jini™ enabled device will be able to see other devices and be able to use them as desired. For example, in a Jini network, a digital camera takes pictures, plugs into the Jini™ network, and it then will instantly have access to the files and can choose to use a color printer to print the pictures, and finds a disk driver to save the pictures and shares the images through distributed disk service. The digital camera can also turn on the light before it takes the picture. Another example: if plug a Jini™-enabled PDA in a Jini™-enabled building, like the Student Union network, the PDA will immediately offer the services, such as the location of the cafeteria, dining services, and gain access to the facilities for post services, fax, printer, copy center. It will be able to use any of these services immediately.
2.1.2 Jini™ technology overview

Jini™, which was introduced by Sun Microsystems in January of 1999, is a Java-based technology that provides mechanisms for the dynamic loading of services over the network. Jini™'s essential goal is to allow any device or software component to be connected to a network and announce its presence; other components that wish to make use of it can then locate it and call it to perform tasks [Rodden]. Jini™ provides platform-independent, spontaneous federated networking built on Java and RMI [Waldo]. In a Jini™ community, services autonomously discover other services as they become available or unavailable; no system administrator required. Code can be downloaded dynamically to allow the clients to use the service, eliminating the need for configuration.

The main components of Jini™

- **Service** is defined by one or more java™ language interfaces or classes. Services are functionally independent that are made available to the other users and can be accessed remotely across the network. Services include devices and software components. For example: Printer, fax, toaster, microwave.

- **Client** is a device or software components that would like to make use of a service.

- **Lookup service** provides a place for services to advertise their presence in a network. Services register themselves as a serialized proxy object into one or more lookup services. In other words, the service proxies are stored in a lookup service.

Discovery /join is the mechanism used by services and devices to find Lookup services on the network. Figure 2 shows the Jini™ lookup.
Jini verses other lookup services [Lindquist]

1) Jini lookup by attributes, service ID and/or interface type; RMI registry uses an arbitrary name.

2) Finding Jini lookup services by direct location or multicast discovery; RMI registry usually requires that clients know the registry’s location, the hostname and port.

3) Leased services, registrations on Jini are under a limited timeframe, which is represented by a lease object.

4) Registrations on Jini are proxy objects that can combine remote references with serialized objects, so service can be remote or local; RMI registry can usually store RMI stubs.
Requirements for participating in Jini™ network

In order for a device to join Jini™ technology network, it must satisfy several critical requirements [Rodden]:

- It must be able to participate in the Jini™ discovery and join protocols.
- It must be able to download and execute classes written in the Java programming language.
- It may need the ability to export classes written in the Java programming language.

Most personal devices with limited memory and processing powers cannot satisfy one or more requirements, therefore, they cannot participate directly in a Jini™ community [Rodden].

Jini™ surrogate architecture addresses devices that are not capable to download and execute the proxy. This task is given to a host-capable machine that provides an execution platform for downloaded code on behalf of the devices. The details of the surrogate architecture are discussed in the following section.

2.2 Surrogate Architecture

Jini™ technology’s efforts are to enable spontaneous networking of a wide variety of hardware and software - anything that can be connected. Therefore, the extension of Jini™ technology to small, portable wireless devices should allow these devices to plug into a distributed framework where they can dynamically communicate and cooperate over the network. However, Jini™ technology requires a full Java 2Standard edition (J2SE™) virtual machine (VM) to be present on the device. For many personal devices,
their memory and processing power are limited [Kehr]. Therefore the limited memory of
these devices blocks the participation of mobile devices in Jini™ network.

To allow a small, non-Jini-capable device to participate in a Jini™ federation, the Jini
community has defined the Jini™ surrogate architecture, released in May 2001[Jini™].
Jini™ surrogate architecture addresses devices that are not capable of download and
executing the proxy. This task is given to a host-capable machine that provides an
execution platform for downloaded code on behalf of the devices. The Jini™ surrogate
architecture provides both the necessary facilities to implement a communication
gateway for interconnects, as well as a place where processing power can be apportioned
to a surrogate that acts on behalf of an attached device [Armold].

Figure 3 shows how a PDA uses the surrogate host service to connect to a Jini™
network.

![Figure 3 PDA using Jini™ service via Jini Surrogate Architecture](image)

The PDA uploads its surrogate, Surrogate P, to the surrogate host. Surrogate P goes to
the Jini lookup service, finds and downloads the service proxy object from Jini services,
Proxy x. Surrogate P communicates wirelessly with the mobile device PDA. On the PDA’s behalf, Surrogate P invokes Proxy x, which communicates with the Jini™ service over the network. Surrogate P acts here as a Jini clients.

Surrogate can act as a Jini service. Consider a limited device (Device A) wants to uses another limited Device P’s service. Figure 4 shows how the surrogate architecture handles this situation. Device A and Device P load their surrogates, Surrogate A and Surrogate P, into the surrogate host. The surrogates register their respective devices proxies proxy A and Proxy P with the Jini lookup service. Device A want s to use the Device P’s service, finds and downloads Proxy P from lookup service. Device A communicate with Surrogate A, on the Device A’s behalf, Surrogate A invokes Proxy P, which talk to Surrogate P, Surrogate P communicate with Device P, which performs the service.

Figure 4  Surrogate P acts as Jini Service

2.2.1 Goals of the surrogate architecture:
1. Device type independence: allows wide range of devices to participate in Jini federations.

2. Network type independence: supports various types of networks and multiple protocols on same physical network.


### 2.2.2 Surrogate Architecture Components

Surrogate architecture defines the following components:

- **Device** refers to a hardware or software component that is not capable of directly participating in a Jini™ federation. These devices are called “limited device” [Thompson]. They are characterized by limited memory, non-virtual-memory operating system and no Java™ technology at all [Thompson].

- **Surrogate** is a Java Object that represents a device, moves from a device to a surrogate host. Each device that is using the surrogate architecture has its own surrogate. Surrogates participate in Jini network as service, client, or both.

- **Host-capable machine** is a system that allows the downloading of code, can run a surrogate, is part of Jini™ federation, and is accessible to the device that is offering surrogate.

- **Surrogate host** is a framework that resides on the host-capable machine and provides a Java 2 platform runtime environment, a Jini runtime environment, export server and surrogate API’s. Surrogate can be loaded into a surrogate host and executed. The surrogate can participate fully in Jini™ network provided by the surrogate host, so that enabling the device represented by the surrogate to
participate in a Jini™ network. The surrogate host manage the life cycle of surrogates by loading, starting, stopping and disposing of surrogates. The surrogate host may also provide other host resources as necessary.

- **Interconnect** is the logical and physical connection between surrogate host and a device. There may be one or more interconnects defined for a physical connection and an interconnect may be on the same physical connection that forms the Jini™ network [Jini™].

- **Export server** is a component or set of components that is responsible for exporting surrogate resources for downloading to remote entities. For example, if a device is a service provider, its proxy has some class file or interface. It is the export server’s responsibility to make these resources available to clients.

- **Interconnect adapter** is an interconnect-specific component or set of components that is part of the surrogate host that implements the surrogate protocol. The interconnect adapter is responsible for implementing the surrogate protocol for a specific interconnection with a device [Rodden]. See figure 5.

![Figure 5 Surrogate architecture components](image)
2.2.3 Device discovery

In the surrogate architecture, discovery is the protocol used by the device and host to find each other over the internet. The simplest form is the device using a known IP address and port of a specific surrogate host to register directly with that host [Jini™], but more useful is that both the device and host use IP multicast for dynamic discovery [Landis].

Jini™ technology IP interconnect specification [Jini™] defines the IP interconnect protocol and the IP-specific surrogate programming model. To use the Jini™ technology IP interconnect specification, Jini™ technology IP specification defines: “a surrogate host and a device must support the multicast host announcement and multicast host request protocol” [Jini™].

Assume that a surrogate host is present on a Jini host-capable machine, and the machine is connected to both the device interconnect and the Jini network. The device and surrogate host are capable to send and receive both multicast and unicast UDP (User Datagram Protocol) message; they must be able to establish a TCP connection.

The device and the surrogate host find each other by a discovery protocol. The surrogate host uses a multicast host announcement protocol to announce its presence on the interconnect, and indicating its available as a host. That means the surrogate host creates a listener to listen for devices registering requests, constructs a message that contains its IP address and port number information, and multicasts at regular intervals. A device using this protocol creates a listener for host announcement messages and when the device received a host announcement message, it sends a registration to the host using the IP address and port number contained in the message. See figure 6.
A different case is that the device initiates the discovery. The devices use the multicast host request protocol to find a surrogate host at an unknown location. In this case, the device creates a UDP listener to receive the host response message, constructs a host request message that contains the IP address and the port number of the host response listener, and multicasts this at regular intervals. Upon receiving a host response message, the device sends a unicast UDP registration request to the IP address and port contained in the host response message. The surrogate host performing this protocol creates a listener for registration requests, creates a listener for host request messages and upon receiving such a message, sends a host response message providing its IP address and port for registration to the address in the request message. The type of discovery is dependent on whether the device can perform discovery or not. However, once a device has obtained the address and port of the surrogate host, it can send a registration request either as unicast UDP message or open a TCP connection to the surrogate host and send the message across using this connection.

See figure 7
2.2.4 Surrogate loading

Once the discovery has been performed, the device’s surrogate must be retrieved. Depending on the interconnect and device’s capabilities, loading a surrogate may be one of the operations [Rodden]:

- Push: the device uploads the surrogate to the surrogate host
- Pull: the surrogate host extracts or downloads the surrogate from the device.

If the URL is provided, the surrogate may be retrieved from the location other than the device. See figure 8.
2.2.5 **Surrogate execution:** Once the surrogate is loaded, the interconnect adapter delivers the surrogate to the surrogate host. The surrogate is activated in the contents of the surrogate host and may use the resources provided by the surrogate host, such as the classes contained in the Jini™ technology starter kit. Once the surrogate has been activated, the surrogate can now access to Jini™ network act for the device. The surrogate communicates back to the device using a private protocol.

See Figure 9.

![Figure 9 Surrogate execution](image)

### 2.2.6 Surrogate reachability

- **Device presence:** when the surrogate is loaded and activated, the surrogate must continually test for the device’s existence. If the device can’t be reached, the surrogate must be deactivated. The resource held by the surrogate including the surrogate host resource and remote resource will be released.

- **Surrogate presence:** the surrogate must keep the device informed that it is still alive in the surrogate host. The device must be able to determine that it is no
longer in contact with the surrogate, and must resume discovery to restart a new
surrogate.

2.2.7 Surrogate Packaging

Surrogate is packaged in a Jar file that contains:

- A manifest file, which points to the surrogate object. The manifest file contains:
  - the **Surrogate-Class**, which specifies the class that is the surrogate object
  - the **Surrogat-Codebase**, which specifies the codebase to annotate
    the surrogate classes.
- Surrogate object
- Supporting resources: download classes, dependent classes, and other resources.

2.2.8 Surrogate activation

The surrogate host searches the manifest for **Surrogate-Class** header, and **Surrogat-
Codebase** header. It creates an execution environment, instantiates surrogate object, sets
surrogate’s codebase annotation, and calls the method activate to activate the surrogate.

2.3 Madison Surrogate Host

Madison is Sun Microsystems’s contributed **surrogate host**. Madison provides both
the surrogate executing environment and the surrogate life cycle management. Madison
provides a context for surrogate object so they can perform Jini lookup and discovery as
well as an export service that a surrogate object registering in a Jini lookup service must
use for its code to be downloadable within the Jini network. Madison has to running on
an Internet-accessible host-capable machine. A web server that is reachable by Madison must be serving the surrogate. A Jini service must be reachable from Madison. Landis and Vasudevan [Landis] reported that they used Madison as framework to connect cellular phone with Jini, but they did not state any detailed information about Madison software and how to set up Madison host.

Chapter 3 gives detailed information about Madison software, the steps to set up Madison and running the sample programs.
"Madison" is the name of the contributed Jini™ technology *surrogate host* implementation from Sun Microsystems. The Madison software includes a Jini™ technology surrogate host and an IP surrogate that conforms to both the *Jini™ Technology Surrogate Architecture Specification* and the *Jini™ Technology IP Interconnect Specification*. [Madison]

This chapter describes the Jini™ surrogate host-Madison in detail answering the questions: where can I get it, how do I install it on the computer, what are the contents of Madison, how to set it up, and how to run the sample code?

### 3.1 Platform requirements

The platform that will be executing Madison must have installed:

- Windows 2000 professional, Version 5.00.2195. (Currently at ASU East campus)

### 3.2 Getting and Installing Madison

Download Madison software from:


This project and paper is based on Madison version ”jini_surr_madison1_0_1” which was released on January 14, 2002. Once downloaded the ZIP file (called
jini_surr_madison1_0_1.zip), extract it to the m:\ directory on the computer or any directory, then it will create a directory called “jini_surr_madison1_0_1” to hold the extracted software. The important contents of the jini_surr_madison1_0_1 directory are shown in the following.

Layout for Madison1_0_1

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jini_surr_madison1_0_1</td>
<td>The root directory of the installation</td>
</tr>
<tr>
<td>doc</td>
<td>Contains documentation for the Madison release, including JavaDocs for the APIs</td>
</tr>
<tr>
<td>example</td>
<td>Example programs</td>
</tr>
<tr>
<td>lib</td>
<td>Contains the Jar files</td>
</tr>
<tr>
<td>policy</td>
<td>Contains two policy files</td>
</tr>
<tr>
<td>source</td>
<td>Contains the source code for the Madison surrogate, under Sun’s community source license.</td>
</tr>
<tr>
<td>index.html</td>
<td>The root of all documentation that ships with the Madison release.</td>
</tr>
</tbody>
</table>

### 3.3 Important components in Madison package:

- Surrogate host
- A sample surrogate that implements the IP interconnect
- A Simulated IP device to use the surrogate
- A sample client for the Simulated device services
- APIs
3.4 The Madison JAR files

The Madison release provides all of the JAR files that are needed to use the surrogate host. The JAR files distributed in the Madison release are:

- **surrogate.jar**: This file contains the Jini™ surrogate technology extension interfaces and classes.

- **IPinterconnect.jar**: This file contains the Jini™ surrogate IP interconnect extension interfaces and classes.

- **madison-boot.jar**: This file contains the boot classes for Madison. This file has been configured as an "executable" JAR file. When used with the -jar command-line option for the Java™ virtual machine, it will start Madison.

- **madison-impl.jar**: This file contains the interfaces and classes of Madison, a surrogate host implementation

- **madison-device.jar**: A simulated IP device implementation that can register multiple surrogates with discovered surrogate hosts, interact with the surrogates, monitor their reachability, and re-register unreachable surrogates, if needed.

- **madison-surrogate.jar**: An example surrogate implementation. When registered and activated in a surrogate host, the surrogate will interact with the device that registered it, monitor the reachability of the device, and deactivate itself once the device becomes unreachable.
• **madison-surrogate-dl.jar**: This file contains the stubs and proxy classes for the example surrogate.

• **madison-client.jar**: An example classes for a client of the simulated device service which use the example surrogate. This client employs a Jini service discovery manager to find all services that implement the com.sun.jini.madison.examples.devices.DeviceService interface, and allows the user to interact with the simulated device through the corresponding surrogate. This file has been configured as an "executable" JAR file. When used with the -jar command-line option for JVM, it will start up the example client.

• **madison-client-dl.jar**: This file contains the listener stub class (which the Jini service discovery manager needs to export), as well as the Jini technology classes on which the stub class depends.

3.6 **Set up the environment variables and write batch files**

Create a directory named “mad”, and place all the batch files in the directory.

The necessary batch files include:

• setEnvt.bat: set class path to ensure all the Java binaries are accessible, for example, set path=c:\jdk1_4\bin; %path%; set the java runtime being used; the classpath must include the classes needed from Jini (jini-core.jar, jini-ext.jar, and sun-util.jar).

the parameters that should be set in this file:

```
set JAVA_HOME=c:\jdk1_4
set JINISHOME=m:\jini1_2
```
set CODEBASE=m:\jini1_2\lib
set DOWNLOADHOST=%COMPUTERNAME%
set RUNTIME_JAR=%JAVA_HOME%\jre\lib\rt.jar
set CLASSPATH=.;%RUNTIME_JAR%;%JINIHOME_BACKSLASH%lib\jini-core.jar;%JINIHOME_BACKSLASH%lib\jini-ext.jar;%JINIHOME_BACKSLASH%lib\sun-util.jar

• jiniservices.bat[Lindquist]: set environment variable by calling
  
  setEnvt.bat, and then start one HTTP server, a RMI daemon, a Jini lookup service(Riggie) 

  **Start one HTTP server** for download code:

  "\%JAVA_HOME\bin\java" -jar "%CODEBASE%\tools.jar" -port %1 -dir

  "\%CODEBASE%" -trees -verbose

  **Start a RMI daemon**: the lookup service relies on the activation to recover it state after crashes or restarts:

  "\%JAVA_HOME%\bin\rmid" -C Djava.security.policy="\%PARENTDIR%\policy.all" - C-Djawa.rmi.activation.port=%3 -J-Dsun.rmi.activation.execPolicy=none -port %3

  **Start Jini™ lookup services**: keeps track of the currently active Jini™ services that are available over the Jini™ network.

  "\%JAVA_HOME%\bin\java" -Djawa.rmi.activation.port=%3 -jar

  "\%CODEBASE%\reggie.jar" "http://\downloadhost:\%host\reggie-dl.jar" policy.all

  reggie_log public

• runspacebrowser.bat: when executed, it brings up a lookup service browser that allows to select the service registrar to observe.

  call setEnvt.bat
%JAVA_HOME%/bin/java -classpath .;%CLASSPATH%;%JINIHOME%/lib/space-examples.jar  
-Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://%DOWNLOADHOST%:8080/space-examples-dl.jar 
com.sun.jini.example.spaceBrowser.Browser -admin 

- runMadison.bat: use it to start Madison host  
  call setEnvt.bat 
  java -classpath "%CLASSPATH% -Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://%downloadhost%:8080/madison-boot-dl.jar m:\jinni_surr_madison1_0_1\lib\Madison-boot.jar -prop madison.prop 

- runDevice.bat: start the simulated IP device  
  call setEnvt.bat 
  java -classpath "%CLASSPATH% -Djava.security.policy=policy.all -jar m:\jinni_surr_madison1_0_1\lib\Madison-device.jar -s http://%downloadhost%:8080/madison-surrogate.jar -prop simulated-device.prop 

- runMadclient.bat: start the client  
  call setEnvt.bat 
  "%JAVA_HOME%\bin\java" -classpath "%CLASSPATH%" -Djava.security.policy=policy.all -Djava.rmi.server.codebase=http://%downloadhost%:8080/madison-client-dl.jar -Dcom.sun.jini.madison.debug=true -jar m:\jini_surr_madison1_0_1\lib\madison-client.jar -groups public -locators jini://localhost 

3.7 Test Madison: 

Madison host provides surrogate execution environment, including Java™ runtime environment(J2SE), Jini™ APIs(.net.jini), and surrogate life cycle management. Madison host doesn’t register as a Jini service upon startup, but it requires the Jini network to be
running. Need to connect to the Internet before starting. The necessary batch files are ready, then the steps to start Madison and run the sample program are as following:

- Start a Dos command prompt, change to directory “Mad”
- First start Jini by run “jiniservices.bat” file, which will start the HTTP server, a RMI registry, a RMI daemon, and a Jini™ lookup services (“Reggie”).
- Launch another Dos prompt, start the lookup service browser by run “runspacebrowser.bat”. This gives a visual feedback of what is happening in the Jini network. See figure 10.

![Lookup browser](image)

Figure 10 Lookup browser

- Start Madison by run “runMadison.bat”: the Madison host listens for surrogate registration via TCP and via UDP. The port numbers for TCP and UDP may be same or different. Figure 11 is the output of the Madison host.
• Start the Simulated IP device:

Start another DOS Prompt, go to directory “Mad”, run “rundevice.bat” file.

When started, the simulated IP device opens and listens on two sockets: one for receiving application instructions from registered surrogates, and one for monitoring the surrogates reachability. See figure 12.
The device discovers Madison and registers the surrogate object using the IP interconnect. If registration is successful, the surrogate object will now appear in the lookup browser. See figure 13.

![Figure 13 Device registered](image)

- Start client: Start a new Dos Prompt, change to directory “Mad”, run “rnmadiclien.bat” file. The test client obtains the reference to the surrogate and allows making a remote call, and obtaining the description. See figure12.
Explanation

1. `${JINIHOME_BACKSLASH%` is where the Jini™ Technology Starter Kit is installed, ex: `M:\jini1_2`

2. `CODEBASE=m:\jini1_2\lib`.

3. Set `DOWNLOADHOST` for dynamic class loading
   ```
   set DOWNLOADHOST=%COMPUTERNAME%.
   ```

4. The `-Djava.rmi.server.codebase` switch is an URL to the downloadable files. The client needs to download to be able to use the service (proxy etc…).

5. The `-Djava.security.policy` switch is a security file, in the examples everything is allowed.

6. The classes needed from Jini are:
   - `\lib\jini-core.jar`
   - `\lib\jini-ext.jar`
3.8 Analysis of Madison Usability

The goal of this project is to get enough experience with set up Madison host in order to use it as technical platform later on.

- Madison includes three pieces to start:
  a. Madison host,
  b. A simulated device that implements the IP interconnect,
  c. A test client for the device.

- The Madison software provides the executable jar file for host, sample device and sample client. For test purpose, no compiling is needed.

- The sample program simulated IP device makes easy to test the surrogate object. But it was limited to letting the device access to the Jini™ network as server, not offer any service.

- The work of setting up Madison directory and classpath is repetitive and is necessary for running the Madison, device and client. So the batch files were prepared.

- During the setup procedure, it was realized that it was very important to set the correct classpath, to Java run time environment, to three Jini jar files that include jini-core.jar, jini-ext.jar, and sun-util.jar.

- When starting Madison, it is important to define classpath to three jini jar files that include jini-core.jar, jini-ext.jar, and sun-util.jar. It is found easy to put the...
jar files into Madison “lib” directory. Without these three jar files, it will fail to boot the host.

- The part of javadocs for how to start Madison is not very clear, so it is difficult to follow.

- Most of the documents available on the Internet are about the Jini surrogate architecture, there is one paper [Landis] reported using the Madison, but it not presented the basic and detailed information about Madison. Non of the books found has example about Madison host.

- The Madison host and the Jini lookup service use full Jini™ technology, must run in the general purpose computer. So anywhere the Jini infrastructure is not installed, or anywhere the wireless connectivity the mobile device use is not installed, this architecture will not work.
CHAPTER 4
CONCLUSIONS

4.1 Conclusions

Jini™ is a distributed computing technology, based on Java™ technology, helps devices to interact over the network and connect simply. Jini™ technology requires full J2SE Virtual machine to be present in the device. Devices with limited memory and processing power such as PDA, cell phone cannot run a full Java virtual machine, so they cannot participate directly in the Jini network. Jini™ surrogate architecture is the solution for this problem. Madison host is Sun Microsystems contributed implementation for the surrogate architecture. Madison host provides the Java run time environment for surrogate to participate in Jini network as well as the management of the surrogate life cycles.

In this paper, Jini™ technology is briefly reviewed. The problem faced by the mobile devices for participate in Jini™ network is presented. The need of the Jini™ surrogate architecture is introduced. Jini™ surrogate architecture is discussed with its complete architectural details, as well the need for the Madison host. To explore Madison host, Madison software was downloaded, installed and tested successfully. The sample Madison simulated IP device and sample client provided by Madison were used to test the system. Finally, the Madison usability is discussed.

4.2 Recommendations

The Madison host is successfully run on the Microsoft Windows platform. Madison seems like forms a strong base for further use as technology framework. The Madison
software provides the source code of the surrogate host, and the device application that interacts directly with the surrogate host should be useful in the further project. Since our goal is to connect PDA with a Jini network via Madison host, allow PDA to use the services on the Jini community such as plug and play printer, email, etc., so we will need to define a surrogate adapter for the PDA, and convert the simulated device to an actual PDA, this will finally enable a PDA to participate in a Jini network.
REFERENCES


Jini™ network technology. Retrieved January 17, 2002 from internet:

www.sun.com/smi/Press/sunflash/2001-05/sunflash.20010530.5.html


Kehr, R. etc. Integration of Small Devices into IT infrastructure. Retrieved February 12, 2002 from internet:


Lindquist, T. *Distributed Objects: Java/Corba-class notes*. Retrieved November 14, 2001 from internet:


Madison installation instructions. (January 14, 2002). Retrieved January 24, 2002 from internet:


