The Wireless Jini Projection Service

David Roger Nordstedt

High Honors Report
Computer and Information Science and Engineering
Fall 2000

Advisor: Dr. A. Helal, *email*: helal@cise.ufl.edu
Department of CISE
University of Florida, Gainesville, FL 32611

Date of Talk: 14 Nov 2000

Abstract

Jini is a distributed computing platform introduced by Sun Microsystems, Inc. in January 1999. A Jini system will allow systems or devices to enter and leave networks while the network environment remains stable. These devices can be used by clients or other services on the network. The device itself can be a client, a service, or both. The goal of this project is to Jini-enable an LCD projector as a service and provide client software for another device on the network to use the LCD projector and its presentation software. The network environment for this project will be a wireless network using TCP/IP and the IEEE 802.11b wireless protocol. The LCD service will be enabled by connecting the SVGA output of a small x86 linux computer to an LCD projector. The client device will be a laptop running the Windows 98 operating system with a PCMCIA wireless network card. A client wishing to use this service will execute a java program to find all available Jini presentation services on the network. The client can then use a GUI interface to choose which service to use and control the presentation displaying on the projector screen.

Introduction

Pervasive computing is the main theme underlying this project. When computing devices and systems are diffused throughout everyday activities, then pervasive computing is present. The implementation details and technical interface should be hidden from view. User interaction with the devices should be simple and sometimes unrecognizable. Devices should be able to communicate without user intervention and configuration on a regular basis. These same devices should be able to enter and leave the network without causing failure on any part of the network. Devices should recover from a network crash or device crash with all necessary state information saved to persistent store for recovery. To do all of this, discovery and leasing protocols must be defined and implemented for the participants in the system. There must be a way for devices to discover other devices on the network, what kind of service they provide, and how to interface with these devices. These devices will not want to retrieve references to

services that have crashed or left the network. The dynamic environment of wireless networking demands such a system. As devices drift in and out of service areas and hence in and out of range of other devices, there must be a way for the network to remain a stable working environment. When a device is no longer available to a certain network, the references that exist in that network should fade quickly so time and network resources are not wasted by trying to interface with an object that no longer exists.

This project takes an everyday scenario for a conference room presentation and adapts it to a pervasive computing environment where the hardware and software used to give the presentation are combined to represent a device on the network. The person wishing to control the presentation uses a client program to find the type of service needed, upload a presentation data file, and start the presentation. The developer writing the client software will only need to be aware of the interface, i.e. the methods defined for the service, to be able to use the service. All necessary service-specific code will be downloaded across the network automatically as needed by the client.

Jini

Jini is, in the most basic sense, a set of classes that provides an API (Application Programming Interface) for building distributed, mobile, and pervasive systems. Jini requires a JVM (Java Virtual Machine) to be present and uses RMI (Remote Method Invocation) and object serialization extensively as its basic communication system. RMI allows the passing of not only data through the network, but code as well. As long as a device has a JVM with RMI installed or access to another network computer that can execute the Java code for them, it is relatively simple to set up a basic Jini system since the Jini and Java API's encapsulate a lot of the underlying transport and concurrency details. The source code for all the Jini classes is available free of charge, although the SCSL (Sun Community Source License) must be accepted beforehand. Jini will work with any network transfer protocol.

Clients on a network must be able to find services needed very easily. Jini fulfils this requirement by utilizing a lookup service. A device offering a service will register with all lookup services it finds when entering a network or recovering from a crash. If a

device on the network wishes to use a particular kind of service, it will make a request to all available lookup services to see if the desired service or multiple services of the type requested are available. If the type of service exists on the network and is known about by the lookup service, a Java object with the service interface encapsulated within will be returned to the device that made the request. This object contains all the code and device-specific information required for the client device to use the service. The client and service may now interact directly using a protocol defined by the service. This may be by remote method invocation, TCP/IP socket communication, Datagram Packets, or any other protocol that both the service and client are capable of executing. The service can also be a stand-alone service where once the client has retrieved the service code, then all service execution takes place on the client machine.

Jini provides leasing specifications where each service device must obtain a lease from a lookup service for a length of time defined by the lookup service. The device must periodically renew the lease or it will be removed from the lookup service. This helps to ensure that if a device becomes unplugged from the network or fails, other devices on the network won't waste time trying to communicate with the failed device. The leasing period should be an amount of time that matches the needs of a particular type of network. A very busy environment with many lookups and new connections will want a short lease so that time isn't wasted with references to devices that are no longer there. The drawback to this scheme is that more leasing traffic is now entered onto an already busy network. A proper lease time for a lookup service can be obtained by monitoring activity on the lookup service and making changes as needed by each particular network. This may not be a problem for most networks, but it is something to be aware of during design and implementation of a Jini system.

The Jini specifications are available at http://java.sun.com in pdf and ps documentation formats. There is also an organization at http://www.jini.org that encourages the collaboration and sharing of code between Jini developers.

The Service

The Wireless Jini Projection Service puts together a lookup service, a service, and a client on a wireless network. The lookup service is entirely software and will reside on a wireless networked computer. The service itself is software combined with the projector hardware and also resides on a wireless networked computer. The client is software to be executed on a computing device capable of wireless network communications and accepting user input and interaction through a Java AWT (Abstract Windowing Toolkit) GUI. This system can be implemented on any type of network as long as the operating systems for each device provide a network interface and support a J2SE (Java 1.2 Standard Edition) implementation. This project is realized on a TCP/IP network because it is so widespread and accessible. There is no reason why this implementation could not also be used on a Bluetooth, HomeRF, or other type of wireless network. An abstract view of the system is shown in Figure 1.

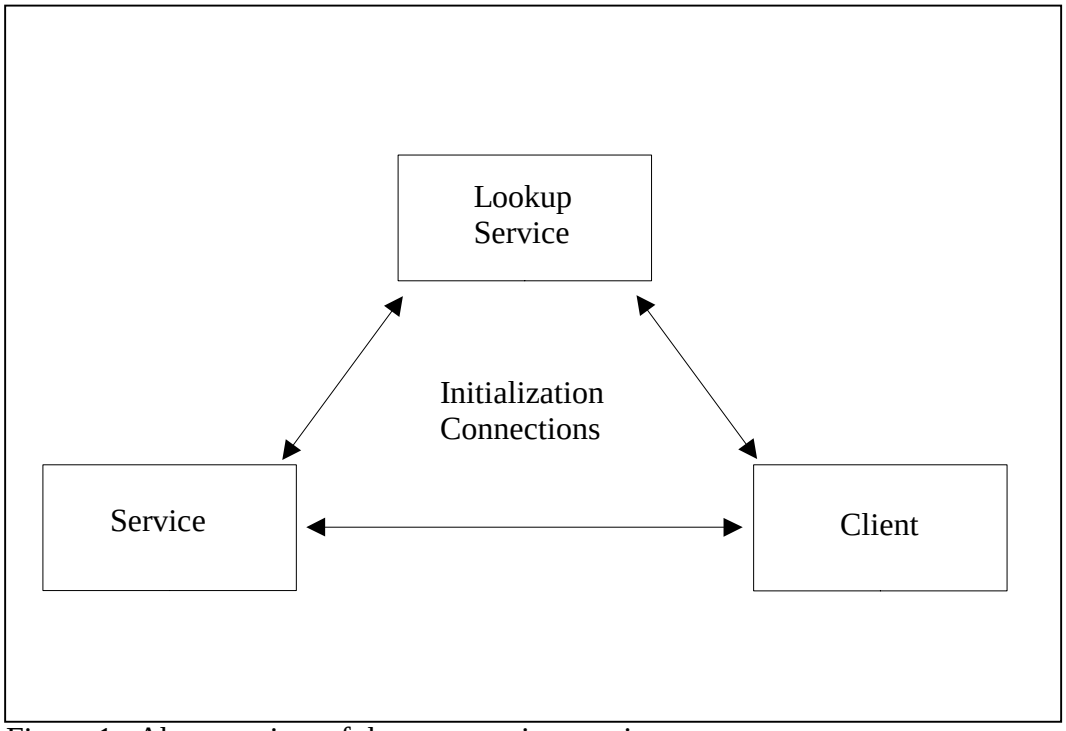

Figure 1. Abstract view of the presentation service

Leasing connections in the system are shown in Figure 2. Leases are required between the lookup service and the service, but they are optional between the service and client.

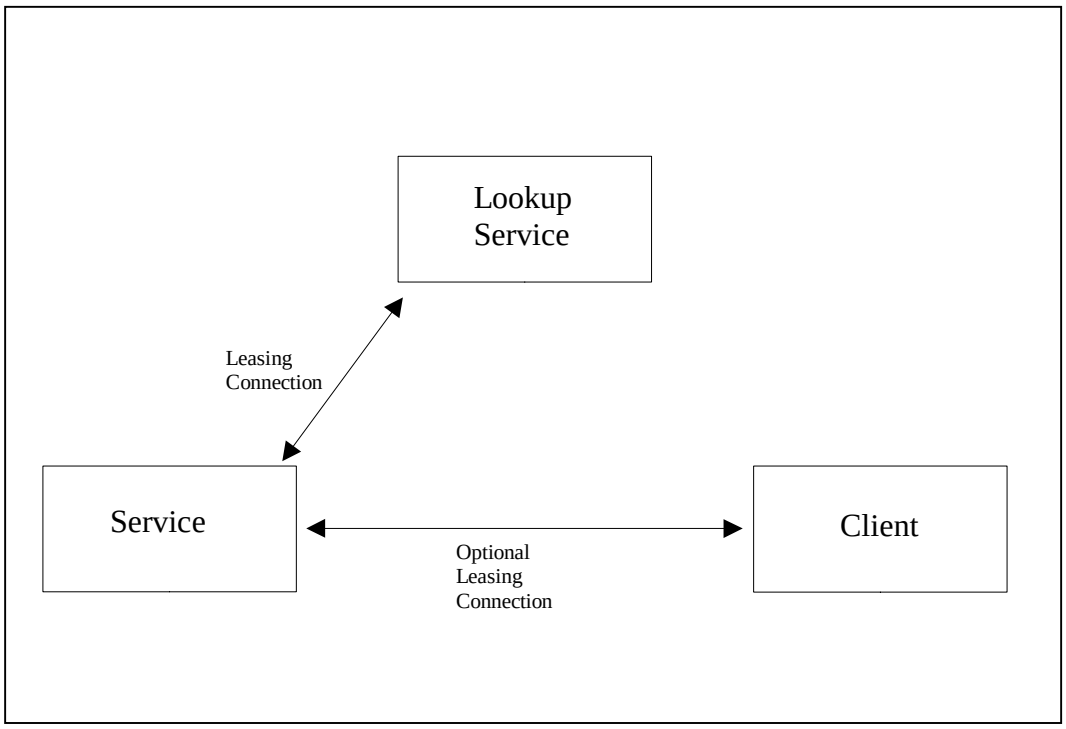

Figure 2. Leasing implementation details

The components used in this project were picked to be the easiest to set up and configure due to the time constraints of the project. A Pentium-class x86 computer will be the main computing machine for the presentation service and lookup service. A linux system will be installed on this machine as well as a Java development and runtime environment, X server, X client, and Sun's StarOffice Suite. All of these software packages are freely available.

The client is a thin-client model. This will demonstrate that a device with low resources and computing power can be used to start and control a presentation session. After connecting to the network, the client will search for all presentation services

available in the area and choose the appropriate one. The GUI client is shown in Figure 3 after discovering a presentation on the local network.

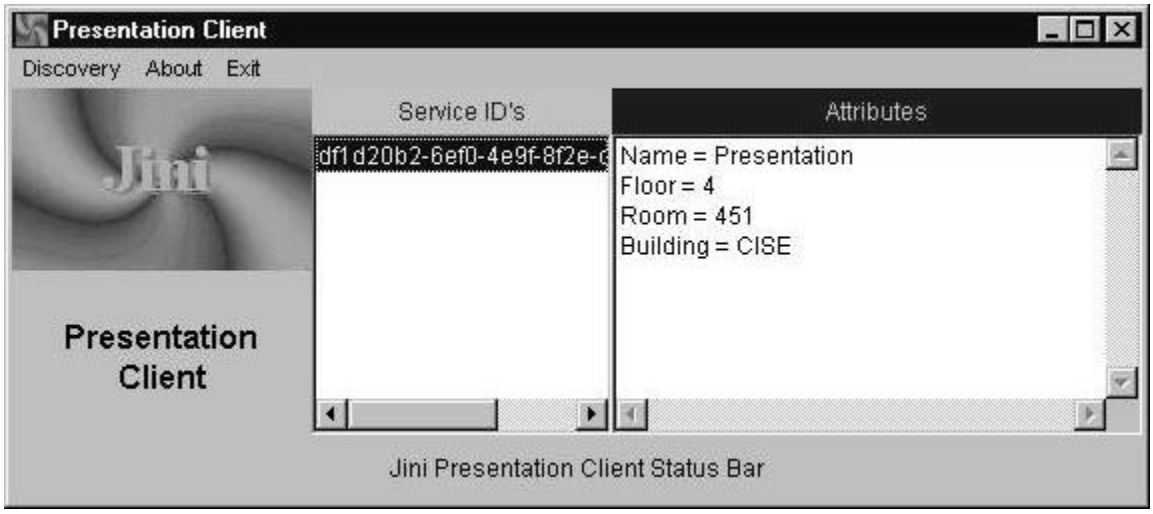

Figure 3. GUI client after discovering a presentation service. The service ID of the service discovered is shown in the center panel and the attributes of the service are shown in the rightmost panel.

The realized system with service machine, lookup service, presentation service, projector, and client is shown in Figure 4.

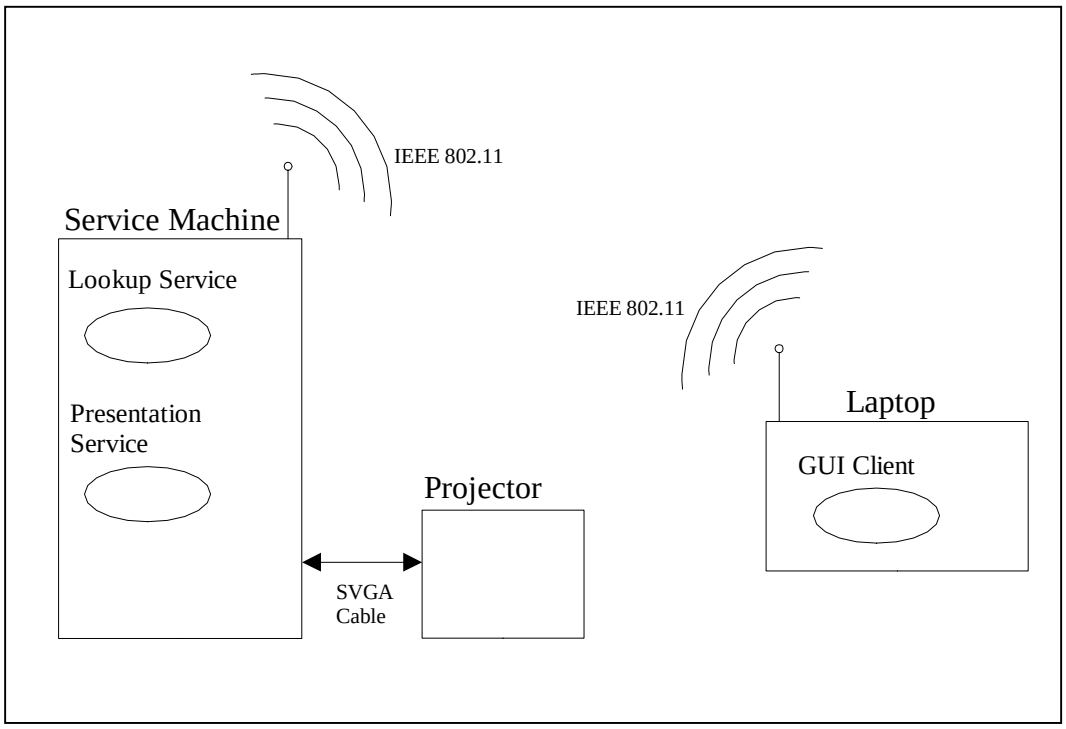

Figure 4. Wireless Jini Presentation Service implementation layout

The client in this system is a laptop computer with many more resources than a thin-client device such as a PDA or mobile phone would have available. The laptop has the Windows 98 operating system installed and is used for this project because of its availability, ease of configuration, and the widely available development tools for the Windows 98 platform.

Implementation

The main piece of hardware is the x86 based computer. This computer will run the linux operating system, the presentation software, and send the output to the projector for display. After looking at the offerings of many different vendors available through the internet, the Advantech Technologies, Inc (http://www.advantech.com) PCM-5820 board was selected as the main computer. This is a single board computer with a Cyrix 233MHz Pentium-class processor, SVGA capability, ethernet, serial ports, parallel port, floppy disk controller, IDE controller, USB ports, irDA port, and PC/104 module support. To complete the package, a PC/104 module with a PCMCIA 2-card adapter was added as well as 128MB DRAM, 6GB HD, a chassis to conveniently hold all the components, and a small 50W power supply.

To install linux, a static IP for our network was obtained to use for a Redhat 6.2 installation over a ftp network connection. A Redhat 6.2 network boot disk was then created using tools available from the vendor. To start the setup, a RJ-45 cable was connected between the lab's network connection and the ethernet port on the PCM-5820. A floppy drive was attached to the PCM-5820 and the network boot disk was inserted. After turning on the power to the machine, the text-based Redhat installation program started from the floppy disk and the output was displayed on the attached monitor. The installation program requested the network parameters such as IP address, netmask, default gateway, and DNS server. It then prompted for the ftp site and directory of the Redhat installation files. For this project installation the IP address is ftp.cise.ufl.edu for the ftp site and /pub/linux/Redhat/Redhat-6.2/i386 for the install directory. Once the ftp site and directory were confirmed by the installation program, the installation continued and the usual Redhat installation prompts were worked through. Standard Redhat installation instructions can be found at URL http://www.redhat.com.

Once the Redhat installation program finished, there were some settings that had to be adjusted for correct performance. The setup program did not correctly configure for a video resolution of more than 640x480 using XFree86 3.3.6, the mediagx drivers, and the Dell M780 monitor. The documentation for the video hardware claims to support 1280x1024 resolution. The modeline in the /etc/X11/XF86Config file had to be changed

to correctly support each monitor that would be used with the video hardware. There may be a problem between the video hardware and the driver, and some information was found at the URL http://www.crosswinds.net/~zoyd/writeups/w2.html.

To obtain a wireless network connection for this machine, the settings for the PCMCIA have to be set and the drivers installed. The wireless network used for this project is supported by Lucent Technologies and the wireless card is the waveLAN IEEE 802.11 that supports the 11Mbps standard. The newest drivers can be obtained at URL ftp://ftp.wavelan.com and have to be compiled for the specific linux system they will be used on. To do this, you need a complete linux source tree since the driver compilation will require some information found in the linux kernel source files. The kernel sources can be found at URL http://www.redhat.com in a RPM distribution. The instructions for compiling and installing the modules are included with the waveLAN drivers. The PCMCIA HOW-TO at URL http://www.linuxdoc.org is also a very helpful document when performing this task. This machine will have a static IP address and this address was chosen from one on the private network set up in our laboratory.

After the initial linux and networking setup is finished, there are several pieces of software required for the projector service that need to be downloaded and installed on the machine. The J2SE (Java 2 Standard Edition) and the JSK 1.0.1 (Jini Starter Kit) should both be obtained from Sun Microsystems at http://www.sun.com. StarOffice 5.2 can also be obtained from Sun. VNC 3.3.x (Virtual Networking Computing) can be retrieved at http://www.uk.research.att.com/vnc/.

The J2SE should be installed on both the client and the service. A good CLASSPATH will include the current directory. The JSK can be unpacked anywhere on both the client and service machines. To run the code successfully, the CLASSPATH environment variable on both the client and service machines should be set to include three jar files that come with the JSK: jini-core.jar, jini-ext.jar, and sun-util.jar. These files include the main API and utility classes in the Jini framework and by including them in the CLASSPATH, they will not need to be included on every command line. StarOffice 5.2 has a standard installation package for all platforms, and will only need to be installed on the linux service machine. The VNC 3.3.x server should be installed on

the linux service machine in any directory and the running user's path set to include the vncserver and vncviewer binary executable files.

The service machine will need the service files available in the gzipped package WPservice1.0.tgz. They can all be installed in the same directory. The client machine will require the files in the gzipped package WPclient1.0.tgz for linux, and the zipped file WPclient.1.0.zip for windows. These files can be installed in any directory the user chooses. These files can be found at the project web site located at the CISE Harris Lab homepage at http://www.harris.cise.ufl.edu/.

For Windows clients, the client can be run by unzipping the client package into a directory and running the sc.bat batch file at the command line. For Linux clients, the client can be run by unzipping and untarring the client package into a directory and running the sc script at the command line. The Windows and Linux packages are identical except for the script used to start the client. The Jini 1.0.1 core and standard extension libraries (jar files) are included with the clients so the client machine is not required to have a Jini installation to use this client. The path to these jar files is included in the script startup code.

The linux machine is set up to boot initially or recover from a crash with the lookup service running, the projector service running, and the VNC viewer full-screen output displaying on the SVGA output connected to the projector SVGA input. There are some scripts in the server package that will be called on system startup. The superuser (root) will need the supplied .xinitrc script in their home directory to be used when the initial X session is started from the scripts. The supplied xstartup file should replace the default file in the projection user's .vnc directory. The /etc/inittab file must be changed to called the startmyx script supplied in the package when the machine boots up. This can be accomplished by adding a line to the bottom of the /etc/inittab file to replace the line that starts the graphical login screen. The Redhat 6.2 linux distribution will have this line at the bottom of the file:

```
x:5:respawn:/etc/X11/prefdm -nodaemon
```

The line above should be replaced with the line below, where 'username' is the user who will be running the VNC session for the Wireless Jini Presentation Service and service/bin is the directory under the user's home directory where the service package was installed:

x:5:respawn:su username -c /home/username/service/bin/startmyx

The line in the file /etc/inittab that sets the default initialization runlevel should also be changed to start in runlevel 5. For example, this default line for runlevel 3 should be changed to the next line shown below:

id:3:initdefault:

id:5:initdefault:

The startmyx script will call several other scripts that are in the server package and will be located wherever the server package was installed. One script will start rmid, the lookup service, the lookup service http server, the service, and the service http server. Another script will start the vncserver to start a background X session and a vncviewer session to display locally the output of the vncserver X session.

Project Difficulties

Along the course of the project there were many difficulties and problems encountered that should be noted here in this section. The intent is to enable others to solve their problems in a shorter amount of time than the author was able to solve them.

The first step of the project was to find and order a small x86 computer that could be hidden from view as much as possible and control and display the presentation on the projector. Finding a good product took a few weeks, obtaining a quote took at least a week, and it took 2 to 3 weeks for the order to get through the university's system. After the order had been placed, there were still more delays with the shipping and billing

arrangements that had to do with the university's policies vs. the company's policies. After that was straightened out, the actual computer arrived within a week, although we paid for 2-day shipping. The lesson here is to find someone who has done this before, get their advice, and constantly check on one's order even though others may claim to have it under control.

The next step was writing the service and client software. This part was fairly straight-forward, although a lot of reading and experimentation was done before the programming was started. Examples were taken from the Jini in a Nutshell[3] book and adapted to the requirements of this system. The client GUI was also started from examples (AWT UI examples downloaded from Sun Microsystems). Troubles arose during the testing of the lookup service, service, and client. Network configuration and command line parameters are a big part of the Jini system. For example, if the wrong RMI codebase is entered on the command line as a property, then others won't know where to dynamically download their classes. A good understanding of basic concepts of RMI and serialization should be obtained before jumping into the Jini arena. An excellent starting point is to experiment with existing programs and watch how they behave. These simple ideas will save Jini newcomers a lot of wasted time with seemingly mysterious problems.

Virtual private networks are also becoming more common and there can be naming and connectivity issues not experienced when each device in a world has its own unique IP address and a name available on DNS servers. For one example, suppose the service is started on a private network machine and broadcasts its announcement on a LAN. If a lookup service on the network with a unique IP address picks it up, then it will keep a record of the service in its database. When a request comes to the lookup service for that type of service(from a client), the lookup service will return a reference to the service. Several things can go wrong at this point. If the client is not on the private network but can reach the lookup service, then the client may not have a path to the service. Or if the client does have a path to the service machine but the service passed a machine name instead of IP address, then the client must also be connected to a functional name resolution service for that particular VPN. There are always many

network issues to contend with, so it is always better to start testing in a known environment and then move to more dynamic systems.

The setup and configuration of the linux system on the small x86 computer has taken the majority of the time of this project. Redhat Linux 6.2 was chosen because of its popularity on the university campus and widespread notoriety. The single-board computer is different from what the majority of linux users own, so there were expected complications. The initial install went well, but the video setup was complicated and not well documented anywhere for this particular hardware. After finding help on the internet and experimenting over several months, a nice screen resolution of 1024x768 pixels was finally obtained.

The fonts in the X setup on the machine were very difficult to set up. After many tens of hours spent on the font setup, smooth large fonts are still unobtainable in the current system. The fonts have a "staircase" look to them that is legible but truly unacceptable for a commercial presentation system. Functionality can be stressed as a selling point, but if the output is not aesthetically pleasing to the eye then the service will not be used. After much research, the most logical guess is that the X server and drivers that shipped with Redhat 6.2 for the mediaGX chip are not performing properly with the hardware. A newer version of Redhat (7.0) has been recently installed on another machine and the X screen resolution and smooth fonts have been much easier to obtain. This is most likely because of the newer X server (XFree86 4.0) and upgrades made to the Redhat system from version 6.2 to 7.0.

A major difficulty in this project was setting up the linux machine to boot up into an X session with the lookup service, presentation service, VNC server, and VNC full screen client running. Much of the problem had to do with standard linux security issues on the machine and what a user was allowed to do on bootup of the machine. This task required a lot of searching newsgroups and asking linux administrators and users on campus of their experience with this. Of course, this is not a normal setup and the information was scarce. Luckily the internet newsgroups are a very good resource and finally yielded enough information for this project to be completed.

Performance Notes

The VNC viewer used on the client side is java-based for portability and to enable the java viewer classes to be automatically downloaded to the client from the service side using the underlying RMI and Object Serialization technology. This convenience comes at the cost of the screen drawing on the client side. Since java code has to work through an additional layer of abstraction versus a native viewer which could access the client - side graphics more directly, there is a noticeable delay in drawing the VNC viewer window on the client side. A side-effect of this is the slowdown of the service-side screen drawing as well. This may be because there are two viewers accessing the same server display and the VNC server will be sending the information at the same time. If the client is not ready for the information from the server, this may cause the server to stop and wait before sending more data to either client. Another noticeable side-effect is that while the client screen is redrawing, it won't respond to keyboard and mouse events. These events are lost and are not transmitted to the vncserver.

The actual time delay between the presentation event of advancing a slide is rather difficult to measure, but the effect of a large graphics slide is much more noticeable than a plain text slide. A plain text slide with a plain color background will redraw in under a second. A graphics-intensive slide will take as much time as 15 seconds to redraw in tests run in the lab. Animated slides are, of course, not worth considering in this version of the client VNC application.

To remedy the slowness of the client side screen redraws, the vncviewer java code has been researched and modified to provide a way to toggle the client's request for screen updates. The client vncviewer window starts in the mode of requesting frequent screen updates. But now there is an additional button on the window that will allow that action to be toggled on and off. Since the presentation output is on the projection screen, there is not really a need to have the client displaying the same information. This option had dramatically improved the speed of our presentation -- especially in the case of graphics-intensive slides in the presentation file. The only delays in redrawing the screen on the service side are the time it takes to transmit the keyboard and mouse events to the service and for the time it takes the native vncviewer to draw the screen. The delays in

most cases are unnoticeable from a presentation controlled from the service machine's console.

Conclusion

The implementation of this project shows a glimpse into the future that is not far away. As computers shrink in size and gather more processing speed and computing resources along the way, smaller hardware devices will become capable of this project's ideas and, as history shows, much more than we can imagine at this time. Pervasive computing involves computing in the background where human users are not always involved. In the future, many mobile devices will be constantly searching the local area for new services that are available. When a user needs a service, it will already be on a list from which the service can be automatically selected. In many cases, the client will already have devices set up to interact with services as they are encountered. There may be services provided by retail stores along a city street such as a store catalog that enable a device passing by to find information. The mobile device can scan each store catalog service looking for a predetermined item or perhaps for an item at a certain cost. The device will sound an alarm when an item is discovered or perhaps just store the information in a database that can be checked at a later time. It may be there evolves a clothing line that has the Jini Thread Chip woven into all the fabrics so the clothing can communicate. If the clothing doesn't match, the user can be alerted that perhaps he or she has bad taste in style. This may prove even more useful to those who are color-blind and cannot always tell the difference between blue and green socks. These are only two examples of an infinite number of possible pervasive computing scenarios. What will be the pervasive devices that are available in the future is hard to say, but these devices will become a reality very soon.

For additional details on implementation and setting up a service such as this or just running the client software, the packages can be download from the project URL[1]. For immediate information, Appendix A contains the README file that is included with the service and client software packages.

Future Work

VNC was chosen as the interface tool so the project could be implemented using out-of-the-box software. Since the display is already projected on the screen, there will be no need for a client screen to display the same information. Essentially all that is needed is a way to send keyboard and mouse events to the VNC server. The VNC Java client can be modified to only refresh the screen if desired and the rest of the time to display a solid background in the window. The window space will be preserved so there is still an area on the client to receive the keyboard and mouse events. This enhancement was added at the very end of the project as an additional feature that was not initially required. Because of this, the additional feature probably needs further work. This aspect of the service can be researched some more for additional improvements and added features.

Writing a service to supply a PDA or cellular data phone (actual thin clients) with this service should not be very difficult. In this case there would no display on the phone or PDA. The service interface would be methods to send keyboard and mouse events only and perhaps some other special features.

An authentication service can be written to add extra flexibility in security to the system. Only users who can be authenticated will be allowed to use the system. This will also be a Jini service and can connect to a company authentication server or an organizational server such as the University of Florida's Gatorlink system.

A 'one-click' option to start the presentation on the machine would also be a convenient option. After selecting a file, one button will be pressed in the client GUI that sends the file to the service, and the service will automatically start the presentation software with the received file.

Acknowledgements

The author would like to thank Dr. Abdelsalam Helal for all his input, ideas, and support in working through this project. It has been a great opportunity to participate in an exciting project that involves so many new concepts and issues. Thanks also go to Dr. Doug Dankel from the CISE department and Dr. Eric Schwartz from the EE department for their creative thoughts and instruction in courses taught at UF. Thanks also for their time to sit on the committee for this project. Choonhwa Lee, the author of the original demonstration model for this project, helped with an essential initial understanding of the components involved in the project and with questions on the linux setup and the network configuration in the lab. David Hon, Daniel Karrels, and Brent Nelson from the University of Florida astronomy department were also of great help in solving many of the linux and networking related setup problems. Thanks also go to Hitakshi Buch for proofreading this document and providing much appreciated feedback and suggestions.

Appendix A

Below is listed the contents of the README file that comes with both the service and client packages which can be downloaded at the project URL[1].

README file
==========

```
CIS4930 Senior Design
University of Florida
Summer 2000


Author:      David Nordstedt
email:       davidrn@ieee.org
URL:         http://www.cise.ufl.edu/~drn

Advisor:     Dr. Abdelsalam Helal
             helal@cise.ufl.edu

Project URL: http://www.harris.cise.ufl.edu/projects/jiniproj.htm


*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

               The Wireless Jini Presentation Service 1.0

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*


-----------------------------------------
Client Files (Contained in WPclient1.0.tgz)
-----------------------------------------
README ........................... This file.  An overall description of the system.

EntryDialog.java.................. Dialog box that accepts the unicast IP, Building,
                                   Floor, Room, and Name of the service we are
                                   trying to find in the jini system (the djinn)

MC_Dialog.java.................... Dialog box that handles the multicast discovery.

UC_Dialog.java.................... Dialog box that handles the unicast discovery.

MsgDialog.java.................... Dialog box for displaying a multiline message to
                                   the user and waiting for a button click on the
                                   OK button.

Presentation.java................. Interface for service.  This is implemented by
                                   PresentationImpl on the server side and used
                                   to interface with the service on the client side.

PresentationClient.java........... GUI and main program for client to discover services
                                   and interface with a particular service.

PrintEntries.java................. Utility that prints information about an Entry
                                   or returns a String containing formatted output
                                   when given an Entry object

ServiceFinder.java................ Class that does multicast or unicast discovery
                                   for services on the network.



-----------------------------------------
Server Files (Contained in WPservice1.0.tgz)
```

```
-----------------------------------------
README ........................... This file.  An overall description of the system.

JoinManagerUtil.java.............. Utility class to be used with the JoinManager
                                   object created in PresentationExec.

Presentation.java................. Interface for service.  This is implemented by
                                   PresentationImpl on the server side and used
                                   to interface with the service on the client side.

PresentationExec.java............. Creates and controls the components of the
                                   service.

PresentationImpl.java............. Implements the Pesentation interface.

PresentationLog.java.............. Handles persistent store for unique service ID
                                   for service.

PresentationSockets.java.......... Server that listens on a TCP/IP socket and receives
                                   a file from a client connection.

PrintEntries.java................. Utility that prints information about an Entry
                                   or returns a String containing formatted output
                                   when given an Entry object

ServerLandlord.java............... Landlord class for leasing.  Not used at this time.

ServiceData.java.................. Wrapper for a ServiceID that can be used with
                                   the PresentationLog class.

ServiceLogHandler.java............ LogHandler implementation to be used with
                                   PresentationLog class.

DesCipher.java.................... VNC viewer file -- no modifications made
animatedMemoryImageSource.java..... VNC viewer file -- no modifications made
authenticationPanel.java.......... VNC viewer file -- no modifications made
clipboardFrame.java............... VNC viewer file -- no modifications made
optionsFrame.java................. VNC viewer file -- no modifications made
rfbProto.java..................... VNC viewer file -- see modification list below
vncCanvas.java.................... VNC viewer file -- see modification list below
vncviewer.java.................... VNC viewer file -- see modification list below

srh............................... shell script to start http server for reggie

srmid............................. shell script to start rmid daemon

sr................................ shell script to start reggie (lookup service)

sh................................ shell script to start http server for service

ss................................ shell script to start service
                                   (takes a port parameter to pass to service)

StartJini......................... shell script to run the 5 scripts above
                                   (srh, srmid, sr, sh, and ss)

startVNChost...................... start the vncserver

startVNCclient.................... start the vncviewer in full screen mode

startmyx.......................... set up a default path and CLASSPATH and run
                                   the three scripts above
                                   (startVNChost, startVNCclient, and StartJini)

java.policy.all................... security policy file that allows everything
                                   (should be changed to permit only certain
                                   actions once the system is up and running
                                   in a stable state and development is complete)


*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Client
------
The client program is entirely written in java and can be run on any platform
supporting the Java 1.2 JVM and a network connection.

To run the program, the jini libraries jini-core.jar, jini-ext.jar,
and sun-util.jar will need to be in the current CLASSPATH.  The command line
is then:

java -Djava.security.policy=java.policy.all PresentationClient

The startup script that comes with the client package(sc for linux or sc.bat for
Windows) will include the path to the Jini jar files that are also distributed
with the client packages and can remain in the client directory if the Jini
libraries are not already installed on the cliend machine.

There are menu items for multicast and unicast discovery.  Once services have
been discovered, they will appear in the center list box of the GUI as ServiceID
numbers.  Clicking on one of the ServiceID's will show the attributes in the
text area box on the right side of the GUI.  Double-clicking on one of the service
ID's will bring up a service interface window through which the service can be used.
From this window the user can choose a file and send it, open the VNC viewer window,
close the VNC viewer window, or quit the service interface window.


*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-


*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Service
-------
The service programs are written in java, but the shell scripts are written
for a linux system.

Some of the scripts expect that certain programs are in certain paths or
have been installed at certain locations in the file system.  There are
not that many programs that are executed, so the administrator setting
up this service should walk through these scripts and ensure that they
will execute properly.

To run the program, start the scripts in this order:

startVNChost
startJini
startVNCclient


To configure a machine to boot up with the services running and the screen
display showing the output of the vncviewer program, a aystem script called
/etc/inittab should be changed to call our startmyx script located in the service
directory.  From inittab, this is called by init and it starts the script as root,
so the .xinitrc file in the superusers home directory(usually /root) should be
replaced with the one supplied in the service distribution.  Below is shown the
contents of the .xinitrc file found in the root directory of the installed service
package:

    /home/projection/service/bin/startJini &
    /home/projection/service/bin/startVNChost &
    /home/projection/service/bin/startVNCclient &
    exec /usr/bin/gnome-session


This script assumes that the user directory is 'projection' and the
service package was unzipped and untarred in the directory service/bin
under projection's home directory.

The /etc/inittab file should be changed so the default runlevel on
startup is level 5.  A line very similar to the one below is the usual
way to do this:
```

```
id:5:initdefault:
```

There is usually a line near the bottome of the inittab file that gives the execution command for level 5. This should be commented out with a '#' character at the beginning of the line and another line similar to the one below added to replace it. The first characters on the line before the first colon can be any label, but must be a unique label in the file.

```
u4:5:respawn:/bin/su - root -c /home/projection/service/bin/startmyx
```

This will execute the startmyx script in the service directory which will start an X session. Below are the contents of the startmyx script:

```
PATH=.:/usr/bin:/usr/local/bin:/bin:/usr/X11R6/bin:/usr/local/jdk/bin:/home/
projection/vnc_x86_linux_2.0:/sbin:/usr/sbin

export PATH

startx -- -bpp 16
```

This script sets up a path for the superuser and uses the system startx script to start an X session with 16 bits per pixel. The superuser's .xinitrc file should contain the instructions to start the other scripts in the service directory as shown earlier in this file and repeated below for convenience:

```
/home/projection/service/bin/startJini &
/home/projection/service/bin/startVNChost &
/home/projection/service/bin/startVNCclient &
exec /usr/bin/gnome-session
```

The startJini script will start rmid, the Jini lookup service(reggie) and its http server, and the presentation service along with its http server. The startVNChost will start a new VNC server session as user projection. The startVNCclient will start a new VNC viewer full screen as the superuser and display it on the services machine's SVGA output. The last line is telling the X server which window manager we want to use for our X session.

When the VNC server session is started as user projection, the vncserver program will look in projection's .vnc directory for a xstartup file that contains the programs to be loaded on startup. The file supplied with the service distrubution contains only a minimum:

```
#!/bin/sh
xrdb $HOME/.Xresources
xsetroot -solid grey
xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
/etc/X11/xdm/Xsession gnome
```

This is just enough to set some basic X paramters, start up an xterm window, and tell the vncserver to use gnome as its window manager.

A final note: There are a lot of scripts here and this is not a usual way of starting up any time of linux system. So look through the scripts and try to understand what is going on before running the scripts.

*-
Modifications made to the VNC java client files:

===============
vncviewer.java:
===============

1. Some variables changed to public modifier so they can be set from the service code that creates the instance of vncviewer:

```
boolean inAnApplet = true;
```

```
            public Frame f;
            public String[] mainArgs;
```

2. Removed the button in the run() method for disconnecting. This will be handled in the client window through the Presentation interface method 'killViewer'.

```
            //         disconnectButton = new Button("Disconnect");
            //         disconnectButton.disable();
            //         buttonPanel.add(disconnectButton);
            //         disconnectButton.enable();
```

3. Remove the disconnectButton handling code in the action() method

4. Added a button in the run() method so we can toggle the updates to our screen on the client side:

```
            enableDisableUpdates = new Button("Enable/Disable Screen Updates");
            enableDisableUpdates.disable();
            buttonPanel.add(enableDisableUpdates);
```

5. Added the button handling code in the action() method to handle the new button we created above. This sets a flag in the rfbProto instance that determines whether or not to keep requesting new client screen updates:

```
            else if (evt.target == enableDisableUpdates) {
                // toggle the sendUpdateRequests flag in rfbProto class
                if (rfb.sendUpdateRequests == true) {
                    rfb.sendUpdateRequests = false;
                }
                else {
                    rfb.sendUpdateRequests = true;
                    try {
                        // Get the updates going again
                        rfb.writeFramebufferUpdateRequest(0, 0, rfb.framebufferWidth,
                                                          rfb.framebufferHeight, true);
                    }
                    catch(IOException ioe) {
                        System.out.println("Error! Can't write Framebuffer Update
Request> " + ioe);
                    }
                }
            }
```

==========
rfbProto.java
==========

1. Add public variable that we can use in the class and from vncviewer instance to control whether we request new screen updates on the client side:

```
            public boolean sendUpdateRequests = true;
```

2. Check the new variable we created in item 1 above in the method writeFramebufferUpdateRequest(int x, int y, int w, int h,boolean incremental):

```
            if (sendUpdateRequests == true) {
                byte[] b = new byte[10];

                b[0] = (byte) FramebufferUpdateRequest;
                b[1] = (byte) (incremental ? 1 : 0);
                b[2] = (byte) ((x >> 8) & 0xff);
                b[3] = (byte) (x & 0xff);
                b[4] = (byte) ((y >> 8) & 0xff);
                b[5] = (byte) (y & 0xff);
                b[6] = (byte) ((w >> 8) & 0xff);
                b[7] = (byte) (w & 0xff);
                b[8] = (byte) ((h >> 8) & 0xff);
                b[9] = (byte) (h & 0xff);
```

```
            os.write(b);
         }
```

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Known bugs:

    1.  The vncviewer window doesn't provide scroll bars, so if the
        service window is larger than the client screen, there will be
        parts of the screen that the user cannot get to with the mouse.
        Since we are writing a new faster server/client system, we can
        add modifications to fix this at a later date.  Another
        alternative for a quick-fix is to have the vncviewer code check
        for the server window size and the client window size and to
        translate the mouse movements and locations as if the client
        window was the same size as the service window.

    2.  Not really a bug, but there is not much security in this system.
        There is a vncviewer authentication panel that requires a
        password, but once you are logged in you effectively are running
        a session as a user on the service machine.  This is ok as long
        as the system is set up so users cannot do any damage to the
        system.  And in most cases you have to trust the people you are
        allowing onto the system in the first place.

        Ideally, we want the client to send a file and that will be
        automatically started on the service machine and the client will
        only have access to controlling this presentation  program only.
        This may require a little work on the VNC server code or may be
        enabled by some security setup on the linux system.


*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Additional notes:   In this project, our linux machine is dedicated to this
                    service.  It's primary purpose is to run the service and
                    anything else is of lower priority.  When the mchine is
                    first turned on or reboots from a system crash, scripts
                    should be run to recover and start the service again.
                    To start up an X session running as a certain user with
                    all the service programs and VNC viewer running will
                    sometimes be difficult with the ssecurity already in
                    place on a standard distribution of linux.

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

==================

End of README file

References

[1] The Jini Presentation Project". 8 Oct. 2000.
Online Posting.
<http://www.harris.cise.ufl.edu/projects/jiniproj.htm>

[2] Flanagan, David. *Java in a Nutshell*, Sebastopol, CA: O'Reilly Publishers (1997).

[3] Oaks, Scott, and Wong, Henry. *Jini in a Nutshell*, Sebastopol, CA: O'Reilly Publishers (2000).

[4] Edwards, W. Keith. *Core Jini*, Upper Saddle River, NJ: Prentice Hall PTR (1999).

[5] "UPnP, Jini and Salutation - A look at popular Coordination Frameworks for future Networked Devices." Posting date unknown.
Online Posting. < http://www.cswl.com/whiteppr/tech/upnp.html> (2 Apr. 2000).

[6] "Jini(tm) Technology Frequently Asked Questions." Posting date unknown.
Online Posting. < http://www.sun.com/jini/faqs/>
(5 Apr. 2000).

[7] "Jini Architecture Specification." Posting date unknown.
Online Posting. < http://www.sun.com/jini/specs/jini101.pdf>
(5 Apr. 2000).

[8] "Jini Lookup Service Specification." Posting date unknown.
Online Posting. < http://www.sun.com/jini/specs/lookup101.pdf>
(5 Apr. 2000).

[9] "Jini Discovery and Join Specification." Posting date unknown.
Online Posting. < http://www.sun.com/jini/specs/boot101.pdf>
(5 Apr. 2000).

[10] "Jini Glossary." Posting date unknown.
Online Posting. < http://www.sun.com/jini/specs/jini.glossary101.pdf>
(5 Apr. 2000).