JINI AUTOMATION OF REMOTE DEVICE VIA IMPROPMTUE COMMUNITY OF OBJECTS

by

CAPTAIN AHSAN IQBAL SHAH

CAPTAIN USMAN ALI

Submitted to the Faculty of Computer Science Department Military College of Signals, National University of Sciences and Technology, Rawalpindi in Partial Fulfillment for the Requirements of B.E Degree in Computer Software Engineering

MAY 2005

ACKNOWLEDGEMENT

All praise to ***ALLAH,*** the ruler of universe, who made us the super creatures, blessed us with knowledge and able us to accomplish this task.

No words to describe respect to families, teachers and Military College Of Signals because without their shadow of love and moral support, perhaps, it could have not been possible to attain this target.

We express profound gratitude to supervisor Asst Prof. Arif Raza, who was affectionate and cooperative during this project and on all other occasions.

Ahsan Iqbal Shah

Usman Ali

April 2005

TABLE OF CONTANTS

CHAPTER 6 ROBOTIC SERVICE

CHAPTER 7 SYSTEM IMPLEMENTATION

CHAPTER 8 FUTURE WORK AND CONCLUSION

LIST OF FIGURES

CHAPTER 1

OVERVIEW

1.1 Introduction

Computing technology has undergone dramatic changes over the past 35 years. Computers are smaller, faster, and more energy efficient. The power that once required a mainframe or minicomputer housed in a frigid, air-conditioned computer room is now contained in a small, consumer device you can carry in your shirt pocket. Such devices

are finding their way into everyday lives. Small portable devices offer many benefits to users, but they present a new set of difficulties for developers. One difficulty is connecting these devices into a network. Small consumer devices demand a flexible, resilient networking architecture so that users can do things like use personal digital assistants to exchange information with existing networks and other computers.

Here comes the Jini vision: When you walk up to an interaction device that is part of a Jini system, all of its services are available to you as if they were on your own computer when the devices are somewhere on the network—and services include not only software but hardware devices as well, including disk drives, DVD players, VCRs, cell phones, printers, scanners, digital cameras, and almost anything else you could imagine that can be connected to a network and passes information in and out.

Jini is a dynamic distributed object oriented technology which will change the software landscape forever. Jini technology provides networking mechanisms that let devices connect to one another to form an impromptu community -- a community that puts itself together without any planning, installation, or human intervention. The network architecture is based on a non-centralized control so is fail safe. Each device provides services that other devices in the community may use, in addition to its own interface. Jini technology focuses on the delivery and interaction of services across the network running any communication protocol, on any device, for any service, ensuring compatibility and reliability for hardware and software that has been built according to Jini specifications. It doesn't matter where on the network a service is implemented, because each service provides everything required to interact with it; thus creating a location transparent environment comprising of services, that can be accessed remotely.

1.2 Jini a Solution to Distributed Computing

Jini is a programming technology based on Java language which is meant to solve a lot of complex issues in a type of network based computing usually referred to as distributed computing. In modern IT business, there are many applications which have to work together but the servers comprising the application reside at different locations. One such example is a bank with servers residing in almost all the seven continents of the world. When such servers need to communicate with each other, we are talking about distributed computing. There are many solutions. Some can be solved by the use of powerful distributed databases, some by using websites developed in Servlets, JSPs or any other such technologies. Similarly there are problems in this networked world which are addressed by Jini technology. Jini is a broad based technology and it can be used to develop many different types of applications. But it must be understood that Jini is written in Java language and infect it is an extension of RMI. So, to run any Jini based application, you have to make sure that the executing environment has support for a Java Virtual Machine and also support for RMI. Having said that, do keep in mind that Jini is also capable of wrapping a code written in other languages by developing wrapper classes using Jini.

1.3 Requirement of Jini

Jini was developed keeping in mind the research conducted under the *Oak project* [1] which resulted in the development of Java language. One vital feature of the Oak project was that it observed huge potential of small consumer and digital devices in the public domains. It observed that such devices, most of whom have embedded processors in them were increasing in popularity making quantum leaps forward and the computer industry was not defining ways to get them connected. Sun’s team led by James Gosling (father of Java) observed keenly that a mechanism was needed to provide connectivity of these devices with the existing computer networks and with each other. They thought that

such a connected world would make lives very easy and would allow us to practice full control over many areas by using these simple devices. These devices included products from many different manufacturers, like PDAs, digital diaries, cell phones, laptop and notebook computers, digital video players, digital TVs, clocks, kitchen utilities, microwave ovens etc. All these devices had processors and related circuitry to run a piece of code! But, there were three very major problems. There was a big black hole in the IT industry. There was no programming mechanism to provide a platform (e.g., APIs) to write network based/ socket based code for these small devices, that had little connectivity features. Moreover, there is no way to administer network communication and no way to find out how to contact other devices to talk to. Secondly, the devices posed a great difficulty in resources. There was little processing power, small memory, very little or no disk space and power/battery problems which meant that these devices could not remain connected to internet always; all the time. The devices were developed by so many different manufacturers that every one had a different style of circuitry, instruction sets, display abilities, and not much was common between a device developed by vendor A and B.

So, Sun figured out a very important historical solution. They would need a higher layer of abstraction above all these different hardware's to provide a common execution environment for a code to run on these devices. The answer lied in Java. As Java is platform independent, a code written for a cell phone in Java will run on the Java Virtual Machine residing inside the phone that may be developed by Nokia, Samsung, Motorola or anyone else. It sounded perfect but for other devices that were even smaller in their hardware resources like a 32KB memory Lego RCX, or a 1 MB based TINI microcontroller, or for microwave ovens, CD players, digital lock markets, home automation systems, industry automation systems, Sun had to make contracts with

vendors to develop JVMs specialized for their devices only, so to overcome memory and other restrictions that these devices had.

Moreover, for addressing the bigger market i.e. the PC user market in particular, Sun had already developed JVM so it was not long that they developed Jini technology kit for the programmers. Using this kit and the Java environment for the PCs, developers throughout the world developed applications that were never thought possible before. The Jini technology kit is an open standard kit and more than 60,000 developers throughout the world take part in its improvement and development along with the Sun's team. So, its getting better and better with every newer version. Even though the manufacturers/vendors of various consumer digital devices are working with Sun to develop Jini based JVMs for their particular devices, their output is slow and only the cell phone sector has achieved some considerable success. But Sun has set the world into action and there is a hope to see results in coming years. Jini is as many researchers say "ahead of its time".

The breakthrough in adopting Jini has been made using alternate approaches. One such approach is the Proxy Architecture which allows Jini based application to utilize even those small devices that cant directly get connected to computer networks or are not able to support Java code from within themselves. This has brought Jini to another level of fame in the developer's zone and applications in Robotics, Tele-robotics, home automation, security surveillance, device to device interactions, remote sensing and remote control etc., have already been seen conducted using proxy architecture. However, the vision remains to enable each device make use of its self contained resources and not depend upon other computers etc. This is being worked upon under a project called the "Surrogate Project".

So, this is why and where Jini mainly focuses but it is not restricted to hardware alone. Software services can use Jini features in an equal manner since Jini makes no difference between a hardware and software service.

1.4 Jini Composition

Jini is a modular technology i.e. its kit has been divided into several modules which work together to run the Jini application. The Jini kit consists of a Jini Lookup service, a TCP/IP class server and the classes required for implementing Jini servers and clients. There are various such classes usually referred to as utility or helper classes. They provide implementation for granting leases, distributed events, transaction manager, registration, discovery, lookup and other purposes.

So, to start the Jini application RMID thread is run firstly, then class server(s) is run, after which Jini Lookup service is run and lastly Jini server and client are run. The server and client program is written according to the customized requirement. The provision of reference implementations by Sun has made the work easy and there is no need to develop everything from the scratch. Jini provides a higher level of abstraction and programmers need only focus on the application logic without worrying about network issues and issues that spontaneous computing poses.

1.5 Scope of the Project

The possibilities are immeasurable. Like, for use at the Working Point of a nuclear weapon accident response, nuclear power accident response, nuclear materials accident response, Improvised Explosive Device (IED) threat response and threat response involving other devices of mass destruction.

Telerobotics has many potential uses, including medical, manufacturing, security, and biohazard applications, among many others. These applications, naturally, have much quality of service constraints, requiring near-real-time responsiveness to avoid potentially disastrous consequences. Security, surveillance, remote monitoring, bomb disposal, remote farming, monitoring places that are hard or dangerous to reach toxic waste ponds, main holes, etc.

CHAPTER 2

REQUIREMENT ANALYSIS

2.1 Software Requirements

Software requirements of the project include Windows 2000 operating system, JDK 1.3; jini connection technology, LeJOS SDK, LeJOS vision API, Sun's robotics development kit, java twain API, JMF, Brick's Music Studio, JSP and Java Beans.

2.2 Hardware Requirements

Hardware requirements include at least a Pentium II processor having 128-system memory with 2 USB ports, a 100Mbp/s network adapter, a local network comprising of at least 2 computers, LEGO Mindstorm Robotic Invention System kit 2.0 consisting of 700+ LEGO pieces, gears, 3 motors and an IR tower, a vision command camera compatible with LEGO Mindstorm Robotic Invention System kit 2.0 [3].

2.3 Other Obligatory Requirements

Other requirements include the design of the robot should be identical as described in the constructopedia, a proper line of sight should be maintained between the IR port of the RCX chip and the IR tower, the robot is designed to move on smooth surfaces, a 9V battery is required to run the RCX and stub must be received by the client in order to operate the robot.

2.4 Non Functional Requirements

As the system design is going to be object oriented therefore it could be reused and extended in future. More functionality can be added in order to provide the user to enhance and modify his system by adding the new features and wireless camera is used to make the robot wireless.

2.5 Project Boundaries and Constraints

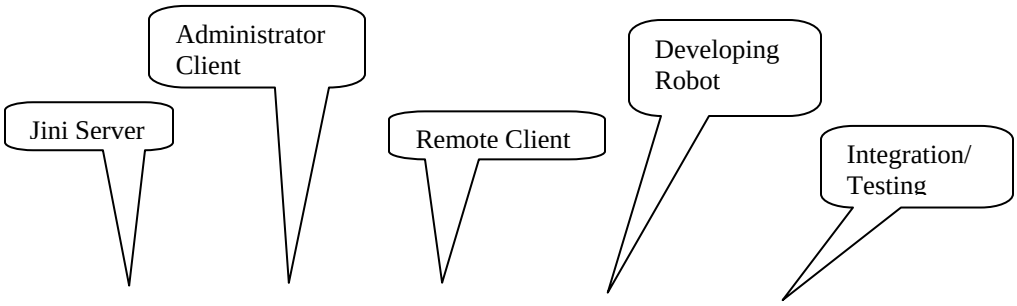
Automating remote devices (embedded and non-embedded) and bringing them under control of remote users, using small wireless consumer devices in the federation e.g. RCX, unleashing the power of Java in small devices with limited resources, developing Java applications in the areas of Robotics and Image Processing, a true distributed system supporting “network is the computer” slogan and object mobility using JINI (Loosely coupled hardware with loosely coupled software), a self configuring, self installing, self healing, location transparent, lease based resource consumption and transaction integrity maintenance system comprising of a community of remote objects, To simulate the big picture in automation of motor vehicles, new battlefield equipments e.g., unmanned planes, tanks, destroyers, etc., robots for outer space explorations, home automation, network based operating systems, applications, security systems, etc. and remote control devices.

At the end of project biggest limitation turned out to be the LEGO Mind storms Robotics Invention System. Disadvantages include its small amount of memory (only 16kb user memory area available for coding), as well as the limited number of motors

that can be used at once (3 motor outputs only). Also, array length restricted to 256 not more than 32 variables allowed. Lack of garbage collector in the Software kit and 16MHz system bus presented another challenge. Another problem encountered was that the LEGO RCX uses CIR (Consumer Infrared) modulation. CIR modulation is used mainly in remote controls and other low-end IR devices, while IrDA is the current standard for all computer IR devices. The frequencies and wavelengths used by IrDA and CIR are not compatible to each other.

2.6 Work Breakdown Structure

The project was divided into 9 activities which were developing a Jini server, developing an administrator client, developing a remote client, developing robot's navigation controller module, its motion detection and tracking module, its security surveillance module, developing the device (robot) itself, integration of all the modules with the hardware and its testing and finally to make the project report/user manual. These are explained in the pert diagram in figure 2.1.



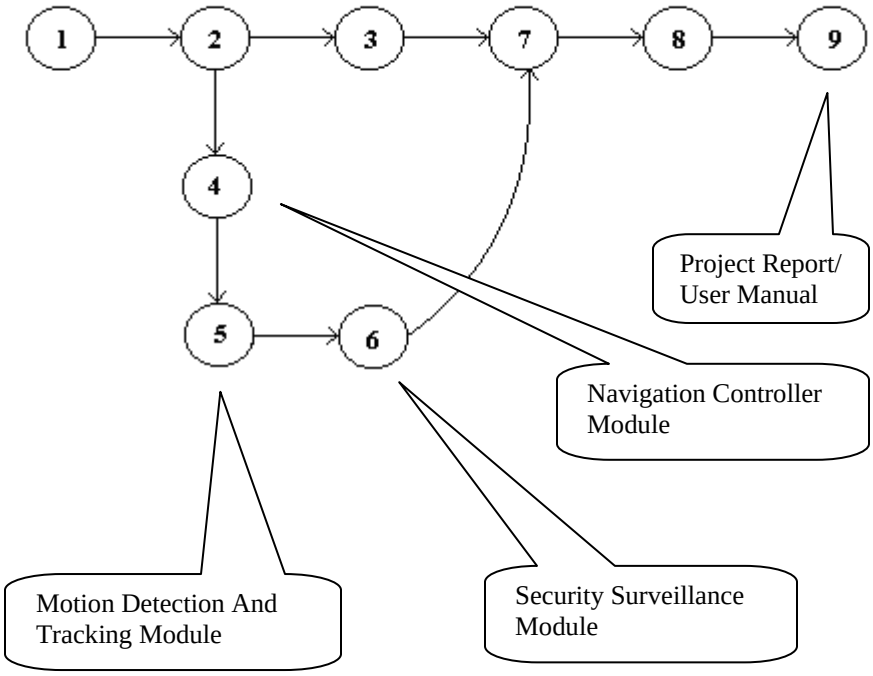


Figure 2.1: Pert Diagram – Work Breakdown Structure

2.7 Project Deliverables

What should the Lego Robot Service be able to do? Start the robot, stop the robot, move the robot in the all the four directions i.e. forward, backwards, left and right, whenever the robot moves, it updates the interface applet of its current position, if a user wishes to move the robot in any direction and there is a hurdle in way, the robot should detect the hurdle using its proximity sensor and not move. Infect, it is to send an error message to the client's applet to inform him, this requires a constant communication between the code running in the robot and the client (using the applet interface and the stub) via the IR tower. Devise a program that can accomplish this "Live

Communication", though the robot lacks memory resources, even then explore the java.net package to check if the Java Service controlling the robot can run within the microcomputer of the robot, to accomplish the last point, try to get the robot connected to a web server running in the PC and display the data in the robot's LCD display, practice more with TCP/IP sockets and check connection possibilities.

These are tough tasks. Besides simple navigation, the robot should be able to do some more useful tasks like e.g. resolving an alternate path if client is unable to make it out of harm's way, allow clients an optional camera view of its navigation using JMF and vision processing; capable of identifying colored objects.

CHAPTER 3

LITERATURE REVIEW

With the world of specialized information appliances poised to take over the technology landscape in the coming years, coordination between devices has become a serious research issue. A number of architectures addressing mobile and specialized devices have emerged recently. These architectures are essentially coordination frameworks that propose certain ways and means of device interaction with the ultimate aim of simple, seamless and scaleable device inter-operability.

Among the well known contenders, Universal Plug and Play [2], Jini (www.jini.org) and Salutation (www.salutation.org) architectures are prominent, coming primarily from the industry. There is also a great amount of active research happening "silently" in the academic world. Whichever gets to become standard(s), it will be useful to understand what these frameworks are all about.

3.1 Jini

Jini from Sun Microsystems, underneath all the hype, is but a coordination framework evolved and adapted from academic research and tailored specifically to Java. The original inspiration comes from the works of David Gelernter (Yale University) and Nick Carriero who built the Linda coordination model using Tuple Spaces, about a decade ago. An inspired result was the JavaSpaces technology with its later evolution into Jini.

Jini uses the term federation to imply coordination between devices. A federation is a collection of autonomous devices which can become aware of one another and

cooperate if need be. To facilitate this, a Jini subsystem contains a set of lookup services that maintain dynamic information about available devices. These services are key to the proper functioning of the Jini subsystem.

Jini Lookup services must be running on a network for the concept to really work. Using well-defined protocols, devices can discover these key services and enter into a dialogue with them. Every device must discover one or more such lookup services before it can enter a federation. The location of these services could be known before hand, or they may be discovered using a multicast. (Note: multicast is a feature that allows controlled broadcast to a group of devices within a selected range. This range can potentially be the whole Internet or just the local LAN). It is possible to assign group names to lookup services so that a device may look for a specific group in its vacinity. For e.g. in a company, administrative decisions may force certain hardcopy devices to belong to some special group, with restricted access. Once a device has located a lookup service of interest it can now tell the service about itself (register), or query the service for information on other devices in the federation. When registering itself, it can associate a set of properties (name/value pairs) with this information which can be matched against queries from other devices.

During the registration process it is possible for a device to upload some Java code to the lookup service. This code is essentially a "proxy" that can be used to contact an interface on the device and invoke methods of that interface. So a querying device can automatically download this proxy and call methods inside device. This is accomplished using Remote Method Invocation (RMI) in Java that allows a Java program to call a method in a remotely running Java program through some exposed interface. The main issue here is that both the devices must have Java embedded for this to work. However, it is possible to avoid the use of RMI and use the Jini lookup services simply to extract

information about a device in question. A proprietary protocol can then be used to contact the device. In this case the lookup service is reduced to a directory service.

The Jini lookup services can form an ensemble, passing on queries up a hierarchy for resolution. These services also try to make sure that they have an accurate snapshot of the current set of active devices. This is done by providing registration leases with expiry. Devices should renew their leases periodically for the lookup services to maintain their registration information. Any information on devices which simply vanish without explicit de-registration will automatically be purged from the databases after the lease time expires.

3.2 Universal Plug and Play

UPnP, pushed primarily by Microsoft, is a framework defined at a much lower level than Jini. Though it is considered at some levels to be a natural extension of Microsoft Plug and Play to the networking scenario, its implementation framework is so different from Microsoft Plug and Play, that the so-called "extension" becomes nothing other than an imagined concept. UPnP is a different beast altogether.

If you do not have a "run-anywhere" solution like Java, then you must look towards leveraging code that is implemented almost anywhere. What better choice exists than the TCP/IP protocol suite? UPnP works primarily with these lower layer network protocols suites, implementing standards at this level instead of at the application level. This primarily involves additions to the suite, certain optional protocols which can be implemented natively by devices. The keyword here is "natively". UPnP attempts to make sure that all device manufacturers can quickly adhere to the proposed standard

without major hassles. It also makes sure that it will be able to ride the next generation Internet Protocol (IPv6) wave when it hits.

By providing a set of defined network protocols, UPnP allows devices to build their own APIs that implement these protocols - in whatever language or platform they choose. As of this writing, the particulars of these protocols are still in a flux. Prototypes do exist. Draft versions are available on the Internet, primarily through pointers at www.upnp.org and Microsoft. But lets take a look at the key ingredients that form the recipe of UPnP.

UPnP uses a special protocol known as SSDP (Simple Service Discovery Protocol) that enables devices to announce their presence to the network as well as discover available devices. This can work with or without a lookup (directory) service in between. If there is such a directory service (called a proxy) available on the network, then devices will use it, otherwise they will follow a proxy-less operation.

This discovery protocol (SSDP) will use HTTP over both Unicast and multicast UDP. This may sound surprising, HTTP is a TCP protocol. However, the same data format can be followed in UDP as well, even in multicast. These protocols can be referred to as HTTPU (Unicast UDP) and HTTPMU (Multicast UDP) respectively. The registration/query process will send and receive data in HTTP format, but having special semantics. An announcement message called ANNOUNCE and a query message called OPTIONS are embedded in HTML to facilitate this.

A device joining the network can then send out a multicast ANNOUNCE - telling the world about itself. A directory service, if present can record it, or other devices may directly see this as well. The multicast is sent out on a reserved multicast channel (address) to which all devices must listen. The ANNOUNCE message must also contain a

URI that identifies the resource (e.g. "dmtf:printer") and a URL to an XML file that provides a description of the announcing device. The latter essentially uses a style sheet tailored to various types of devices. A query for device discovery can also be multicast (an OPTIONS message) - to which devices may respond directly, or it may be directed towards a directory service if present. Queries however do not seem to be very powerful. They do not seem to be targeted at the XML descriptions, but must use URIs like "/ietf/ipp/printer". The XML description is looked at only after discovery takes place.

UPnP also addresses the problem of automatic assignment of IP addresses and DNS names to a device being plugged in. For these, a few more protocols are introduced. One way to assign an IP address is to look for a DHCP server on the network. If not present, then the device can choose an IP from a reserved IP address range known as the LINKLOCAL net (address range 169.254.x.x ?). It must use the ARP protocol to find an unused address in this range locally and assign to itself. This is known as AutoIP. Note that this is not a routable IP address and therefore packets will not cross gateways. If a DHCP server comes up anytime, the devices will attempt to switch their IP address (Easier said than done).

To address issues in naming, a multicast DNS proposal has been drafted. This would allow plugged-in devices to a) discover DNS servers via multicast and also b) resolve DNS queries about itself via multicast. So when plugged into a network having no DNS servers locally, a device can send out a multicast packet and discover a remote DNS server and then use it for Internet name resolution (like resolving URLs). The device itself can optionally listen on the multicast channel and respond to queries for its own name! However no security issues seem to have been addressed by UPnP yet. Once the discovery process is through and the XML description of a device is received, proprietary protocols can take over in communicating with the devices.

3.3 Salutation

The Salutation coordination framework [8] seems to be a more rigorous and useful framework from a coordination perspective than either Jini or UPnP. It seems to have drawn more from research on intelligent agents than other frameworks mentioned here. Salutation trod a middle way between device autonomy and standardization. This would allow many vendors to adapt many of their products more or less easily to the specification and inter-operate with one another.

In Salutation, a device almost always talks directly to a Salutation Manager, which may be in the same device or located remotely. They act like agents that do everything on behalf of their clients. Even data transfer between devices, including those across different media and transports, is mediated through them. The framework provides call-backs into the devices to notify of events like data arriving or devices becoming unavailable.

All registration is done with the local or nearest available SLM. SLMs discover other nearby SLMs and exchange registration information, even with those on different transport media. The latter is done using appropriate transport-dependent modules called Transport Managers which may use broadcast internally. Broadcast RPC is optionally used for this. The concept of a 'service' is broken down into a collection of Functional Units, each unit representing some essential feature (e.g., Fax, Print, Scan or even sub-features like Rasterizing). A service description is then a collection of functional unit descriptions, each having a collection of attribute records (name, value). These records can be queried and matched against during the service discovery process. Certain well defined comparison functions can be associated with a query that searches for a service. The discovery request is sent to the local SLM which in turn will be directed to other

SLMs. SLMs talk to one another using Sun's ONC RPC (Remote Procedure Call). Salutation defines APIs for clients to invoke these operations and gather the results.

One of the key features that Salutation tries to provide is transport protocol independence. The communication between clients and functional units (services) can be in a number of ways. In the native mode these may use native protocols and talk to one another directly without the SLMs getting involved in data transfer. In the emulated mode, the SLMs will manage the session and act as a conduit for the data, delivering them as messages to either party. This provides something very important - transport protocol independence. In the salutation mode, SLMs not only carry the data, but also define the data formats to be used in the transmission. In this mode well-defined standards of interoperation with functional units are to be followed, allowing generic inter-operability. All communication in the latter two modes involves APIs and events which are well defined. Data descriptions follow a popular notation/encoding called ASN.1 (Abstract Syntax Notation One) from OSI. Instead of lease management, SLMs can be asked to periodically check the availability of functional units and report the status. This allows a client or functional unit (service) to become aware when either have left the scene.

Salutation has already defined a number of Functional Units for various purposes like faxing and voice mail. These also allow the use of vendor specific features. The consortium seems to have taken great pains in defining acceptable (?) and general functional unit specifications.

3.4 CORBA

Java Spaces may also be confused with CORBA (Common Object Request Broker Architecture). The Object Management Group, an association of over 800

computer industry members, designed CORBA to span several component and/or object computing models and allow interactivity across them. CORBA defines a method to write client and server objects in a number of programming languages, including Java, C++, Smalltalk, Ada, and, believe it or not, Cobol. CORBA makes it possible for these languages to run on a number of different platforms and still communicate and interact with each other through a common object brokering infrastructure. All this works through the industry-accepted standard known as the Internet Inter-ORB Protocol (IIOP).

Java Spaces provides server-side processing in a distributed environment. You are still passing objects around the network, but unless you use a Java object request broker like Visigenic Visibroker, Sun's Java IDL, or IBM Component Broker, these objects can only work in a Java environment. CORBA allows software development teams to develop using several languages or new code to be integrated with existing code by wrapping legacy code in CORBA objects. Java Spaces actually fits into a separate brokering system for objects on the server side. Jini, for its part, works similarly to *CORBA services*, a set of system objects for resource identification, leasing, transactions, and so on.

Also comparable to Java Spaces-Jini is the Parallel Virtual Machine (PVM) architecture from Oak Ridge National Labs. PVM is actually a set of programming libraries that allow you to build *multicomputers*, a set of independent machines on a network that share application processing, creating true heterogeneous network computing. Using PVM, you can create a distributed application, with each PVM daemon (server) automatically handling the job of network processing synchronization, communications, and fault tolerance. PVM participants may be Unix or Windows machines communicating over TCP sockets. There are programming interfaces to PVM in many languages including Java, C, C++, Perl, and Python.

Java Spaces is similar to PVM. Java Spaces is based on *Tuple space,* a concept created by Dr. David Gelertner of Yale University in his public domain Linda project. Tuple space is an abstract concept of a location, whereby processes can communicate in a distributed shared memory. The memory may be on a single machine, but more realistically is on several computers on a network, each possibly a completely different computing platform. PVM and Java Spaces solve the same problem of building heterogeneous network computers using different means and mechanisms.

Finally, let's compare Java Spaces-Jini with Lucent Technology's Inferno. Inferno is a combined programming language (Limbo), virtual machine (Dis), and communications protocol (Styx) for small computing device models. It was designed as the core operating system for small multimedia and embedded devices such as pagers, cell phones, and set-top boxes. You can see that it shares many things in common with Java. Inferno itself is designed for a single device in a networked environment. It can locate resources kept on other Inferno devices across the network and communicate with them to exchange data. In this respect, Inferno has capabilities similar to Jini. Yet, it is lacking a mechanism to distribute the application processing across multiple machines.

For all other comparisons, it's important to note that Java Spaces-Jini is not middleware. It does not create communication links between common commercial databases such as Oracle, Sybase, DB2, and so on, and object models like CORBA, DCOM, and SOM. It is, instead, the underlying technology that will allow us to build the new generation of application; one that can be distributed across a network of computers. This is the essence of network computing.

CHAPTER 4

PROJECT DOMAIN OVERVIEW

4.1 Preamble

Jini technology is being used to build communities of services out of simple pieces of hardware and software that have been built according to the Jini specification. Examples range from business enterprises, medical systems, financial applications, and battlefield implementations, to a Jini technology-enabled car. Also Connection of disparate embedded devices to Internet, Spontaneous networks, Ubiquitous Computing, network plug and play etc. JINI is also applicable for controlling networks for home and building, for example household appliances, entertainment electronics, information and communication, Installation domains, security and safety domains, sundry special domains etc, as shown in figure 4.1.

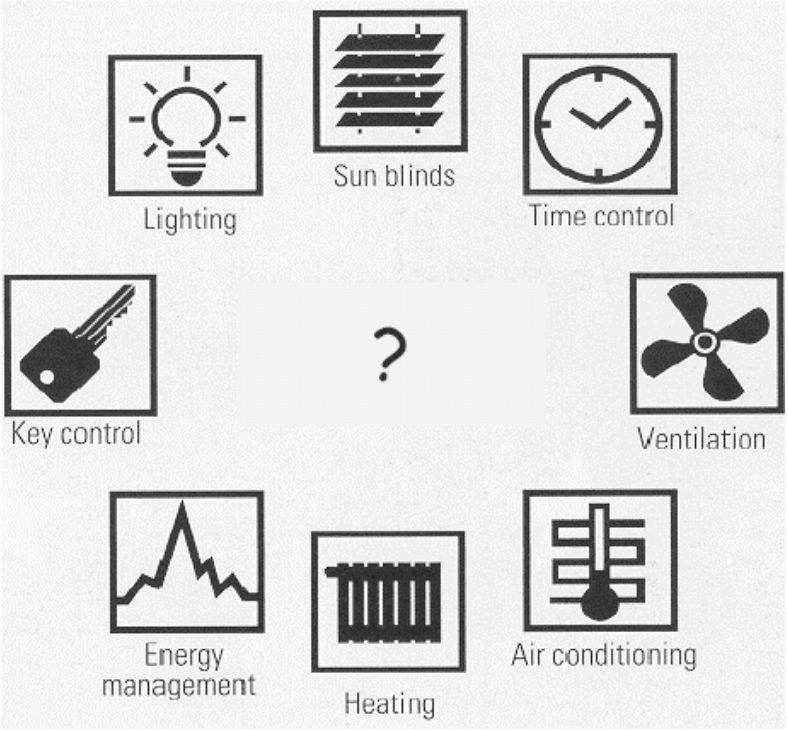


Figure 4.1: Domain Overview

4.2 Telerobotics

Telerobotics have many potential uses, including medical, manufacturing, security, Web based manufacturing, Control of underwater ROV's, Entertainment, Remote Manipulators, Underground mining, and bio-hazard applications, among many others. These applications, naturally, have much quality of service constraints, requiring near-real-time responsiveness to avoid potentially disastrous consequences. Telerobotics also have a larger scope in Security, surveillance, remote monitoring, bomb disposal, remote farming, monitoring places that are hard or dangerous to reach...toxic waste ponds, main holes, etc.

4.3 Problem Description

There are situations when firms or laboratories have to resort to remote manipulation. Such cases appear when dangerous objects have to be handled or/and when the environment is too aggressive for humans. Typical applications belong to the nuclear domain (for instance in the dismantling of a nuclear plant), deep-sea domain (work on underwater structures of oil rigs) and spatial domain (exploration of distant planets) [4]. Teleportation has the supplementary advantage of giving the possibility of sharing an experiment between several operators located in distinct places. This way, heavy outdoor experimentations could be easily shared between several laboratories and costs could be reduced as much. However, long distance control of a remote system requires the use of different transmission media, which causes two main technical problems in teleportation. *Limited bandwidth* and *transmission delays* due to the propagation, packetisation and many other events digital links may inflict on data. Moreover bandwidth and delays may vary according to events occurring all along the transmission lines. In acoustic transmission, round-trip delays greater than 10s and bit-rates smaller than 10kbits/s are common.

These technical constraints result on one hand in difficulties for the operator to securely control the remote system and, on the other, make classical controls unstable. Many researches have proposed solutions when delays are small or constant, but when delays go beyond a few seconds and vary a lot, as over long distances asynchronous links, solutions not based on teleprogramming are fewer. Such delays make one of the most exciting, baffling challenges in robotics the *navigation*, which is important to mobile robots for obvious reasons. Without it, a robot will only be capable of wandering around aimlessly. Is this navigation? By strict definition, navigation is the science of directing the course of a vehicle. With robot navigation, a robot should be able to go from one point to another and have the ability to return to the original point. Furthermore, it should be able to find another position relative to the original position. This requires

keeping track of where the robot is, in one form or another. The design of a behavior system tailored for particular robotic service, managing of thread priority, low level communication with the proxy code running on the PC, and detecting motion on a real time based video stream captured through the web-cam were also some of the big challenges that had to be faced.

4.4 Project Proposal

The aim of ***J***ARDICO project is to illustrate device automation using Jini technology. There are many different types of devices available today which are programmable. Devices that are PSDs (Program Storage Devices) are embedded devices that can be programmed using any low level language like C/C++ or Assembly (if support is provided) or any other firmware language with all the machine level interfaces to communicate with the device. Devices that have a processor and memory like microwave oven, etc. can be tailored or customized to support the execution of a piece of code because they already have the basic processing components. Devices like PDAs and palms can also be programmed using the KVM (Kilo Virtual Machine – a thin version of JVM), which is an assistant technology used to run Java code on Jini devices.

A PSD has been used because of the availability in local market and cost. The factors basing on which the selection of PSD was done are, it should have a useful purpose, it could simulate a big picture, it is within the financial constraints and capable of making definitive and tangible effects.

4.5 Designing a Jini Architecture for *J*ARDICO

Next a network has to be designed to provide a distributed environment for the application. Once a LAN was developed, an architecture was designed using Jini

technology to support multiple objects from anywhere around the network to get registered with the application and advertise their services to the rest of the objects. Object here represents any hardware whose connectivity will be provided. All the distributed server side programming was done in this stage. This development served as the backbone for system and has to be designed in a seamless manner to ensure that QOS, fail safe and self healing features provided by Jini are well utilized. The Jini architecture, which was used here, is represented in the figure 4.2.

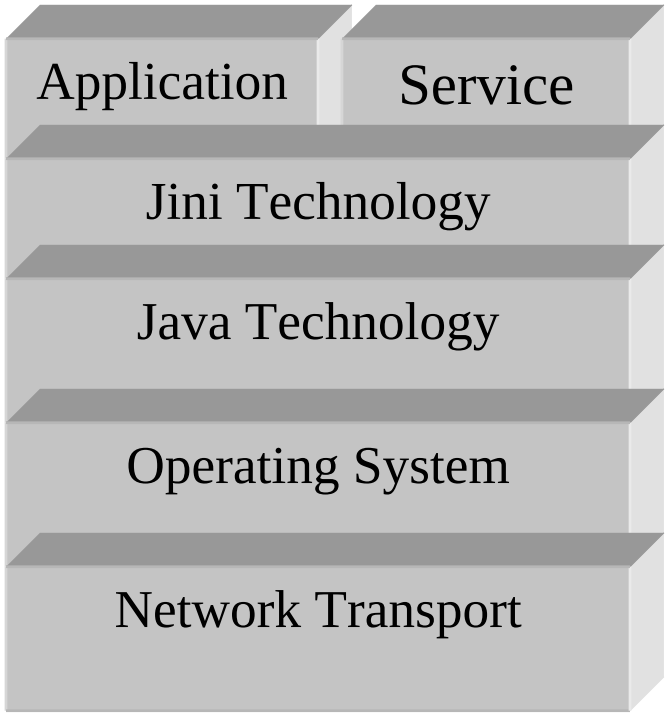


Figure 4.2: Layers of Jini Network

The network transport layer acts as the lowest layer, which provides based for the higher operating system layer. Next higher-level layer is comprised of the Java

technology, with Java Virtual Machine (JVM) installed in the operating system's run time environment. The Jini technology acts as an even abstract layer since it is based on Java. Finally comes the application and services developed and deployed using Jini.

4.6 Programming the Device at System Level

The PSD used was programmed here. This stage posed a challenge since there are many different manufacturers of PSDs and each may use its own proprietary protocol to communicate with them or to program them. To write close to the hardware C/C++ or Assembly was the first choices but if memory restrictions are met, Java can be used along with Kilo Virtual Machine (KVM), which is a tailored version of JVM, thin enough to install in just Kilobytes.

Code to provide interfaces that grant full access to all the motors and related resources of the underlying hardware had to be developed here. The code would also have to provide flawless connectivity using the hardware protocols of the device, to loosely coupled (LAN) environment. This could be achieved using wired or wireless medium. The later again depends on financial constraints. Lastly, the code has to provide reliable two-way communication with the ***J***ARDICO application running on the network because this is what Jini is all about. i.e., letting devices accessed by remote users and the users being updated by the devices about their status.

4.7 Registering the Device's Service

The Jini technology uses three basic protocols to accomplish its tasks. The discovery, join and lookup protocols. Once a device is programmed, its job is to create a proxy object representing its service and get it registered with the Jini Lookup service. For this purpose, it first uses the Discovery protocol to find the Lookup service. Next, it will join the Lookup Service (LUS) using the Join protocol. Once registered, it will advertise its existence. Now, when a client wishes to access the service offered, it has to find the Lookup Service to find the service it is interested in. For this purpose, it also uses the Discovery protocol to find the LUS. Once found, the client requests the LUS to find for him his desired service. Upon success, the LUS sends registered service's proxy to the client. Now, the client can access the service through its proxy by making method calls, remotely.

4.8 Embedded Robotic Service

Is responsible for the navigation of telerobot, responds when collision is detected, manages the communication link through the IR port, Automatically calculates the turning position in the direction of specified object and maneuver the object (i.e. fetches the desired object) using its claws.

4.9 PC Based Robotic Service

Also manages the communication link through the IR tower. Also allows a security monitoring feature and if the security is breached, sounds an alarm, captures the image of the intruder. This actor also provides remote events to occur. It is responsible to register a service with LUS as well as unregistered it.

4.10 The Lookup Service (LUS)

This is where all the services are registered in order to be used by remote clients. The power of LUS lies in port mapping. The LUS advertises/publishes the services that are registered with it, to the remote clients. It also takes special care of lease management, which are given to services. LUS constantly monitors the clients and the services allotted to them, the devices can come online or go offline without disrupting the network (i.e.: Network Plug-N-Play).

4.11 The Client

In order to use any service the client has to first search for the LUS. When the client has the LUS a service proxy is sent to it using Dynamic Class Loading. Finally the client is ready to access the service remotely.

4.12 Benefits

Jini is intended to be for general use, applicable in a wide range of distributed systems environments. In the home, it can help simplify the job of "systems administration" sufficiently that non-sophisticated homeowners were able to compose and use a network of consumer devices. In the enterprise, it can make it easier for traditional systems administrators to manage a dynamic network environment. Thus, the home and the enterprise are two major network environments where Jini may make sense, but Jinni’s potential is not limited to these environments. Basically, anywhere where there is a network, Jini could potentially provide the plumbing for the distributed systems running on that network. For example, networks are emerging in cars, which nowadays contain many embedded microprocessors. Were you to plug a new CD player into your car, Jini could, for example, enable the user interface for the CD player to come up on your car's dashboard.

Jini Attempts to perform distributed garbage collection of remote server objects using the mechanisms bundled in the JVM. Jini is used to overcome the plug and play concept. Jini enabled device can be used instantly anywhere and any time regardless of underlying O/S. Jini doesn't depend on specific protocol. Connecting embedded and consumer devices with PCs and providing remote access. From disk centric to network centric; network is the computer i.e., devices don’t need to have disk in them to store code. Running code independent of user interface, and hardware platform is used thus breaking browser and desktop encapsulations. Developing fail safe, self-healing, self configuring, self installing and updating distributed systems. IT creates an object framework where objects could represent any service (s/w, h/w) and interact with other objects. Jini attempts to increase the object mobility over the network, ensuring transaction integrity among objects and developing decentralized applications.

4.13 Robotics

Telerobotics have many potential uses, including medical, manufacturing, security, Web based manufacturing, Control of underwater ROV's, Entertainment, Remote Manipulators, Underground mining, and bio-hazard applications, among many others. These applications, naturally, have much quality of service constraints, requiring near-real-time responsiveness to avoid potentially disastrous consequences. Telerobotics also have a larger scope in Security, surveillance, remote monitoring, bomb disposal, remote farming, monitoring places that are hard or dangerous to reach...toxic waste ponds, main holes, etc.

CHAPTER 5

JINI

5.1 Foreword

Jini addresses the problems of distributed computing with a set of simple interfaces and protocols. Jini is built on top of the Java™ 2 Platform Jini technology is complementary to Enterprise Java Beans™. Jini enables spontaneous networks of devices and software services to assemble into working groups of objects, or Federations. Jini enables self-healing when one or more devices are removed from the Federation.

Jini technology provides a simple infrastructure for providing services in a network. It enables spontaneous interactions between applications. The result is a network of services connected together dynamically. It acts as a middle layer between applications and operating system as shown in figure 5.1. Services can join or leave the network in a robust manner. Clients can rely upon the availability of visible services. The purpose of Jini is to federate groups of hardware and/or software components into a single, dynamic, distributed system. The resulting federation provides the simplicity of access and ease of administration. It guarantees the reliability and scalability of the whole system.

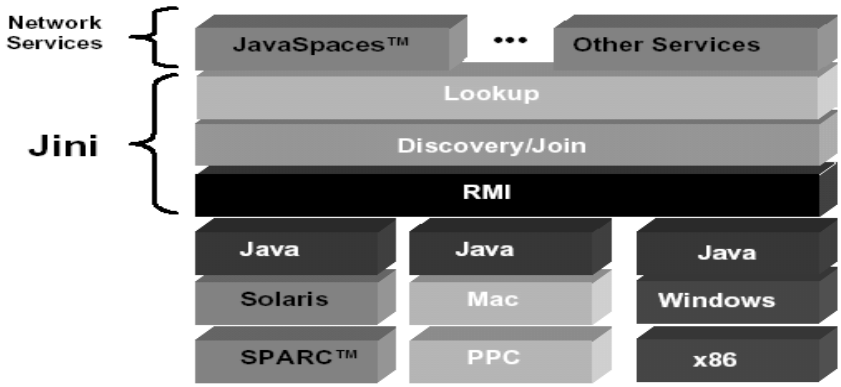


Figure 5.1: Jini - Middle Layer

It is a distributed system (as shown in figure 5.2) based on the idea of federating groups of users and resources required by those users. The overall goal is to turn the network into a flexible, easily administered tool where human and computational clients can find resources. Resources can be implemented as hardware devices, software programs, or combination of the two. The focus of the system is to make the network a more dynamic entity that better reflects the dynamic nature of the work group by enabling the ability to add and delete services flexibly. For example: Print out a picture from a digital camera, connect a Palmtop to the Internet, Plug and Play for networking devices.

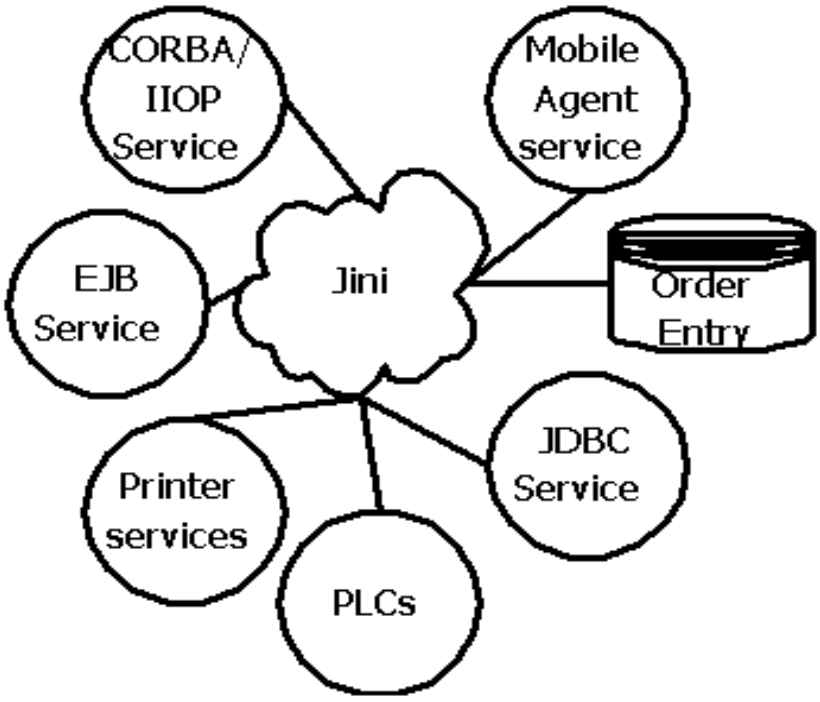


Figure 5.2: Jini Distributed Environment

5.2 Characteristics of Jini

There are certain characteristics which distinguish jini in the distributed environment and particularly in the field of home automation. Though few of these are offered by some of the other parallel technologies (as described in chapter 3) but jini by and large lead from the front.

5.2.1 Network Plug-and-Play

A service is visible after it is plugged into the network. There is no need to configure the system. The “network” announces the availability of a new service. Any interested client is then able to use the service. To deploy a new service, it only has to be plugged into the Jini-enabled network.

5.2.2 Spontaneous Networking

When services plug into the network they become available spontaneously. They can be discovered and used by clients and by other services. Clients and services work in a flexible network. They can organize themselves in the most appropriate way for the set of services that are actually available in the environment. When a plugged service is connected, the network withdraws automatically the service.

5.2.3 Service-Based Architecture

Products can be designed as services instead of stand-alone applications. As Jini enables services to collaborate with each other to perform particular tasks, service developers gain in reusability and modularity.

5.2.4 Simplicity

Jini is concerned with how services connect to one another. There is no constraint on what services provide and how they should work. Jini is based on Java. However, services can be written in languages other than Java, if they provide a chunk of Java code that can participate in the Jini mechanisms.

5.2.5 Reliability

Jini supports interactions between distributed services. It helps programs to find services, and ensures spontaneous availability. Services can appear to and disappear from the Jini federation in a very lightweight way. Interested parties be automatically notified when the set of cooperating services changes. When damage occurs, Jini is able to repair itself. The main concepts of Jini and their relationships are described in the conceptual model shown in figure 5.3.

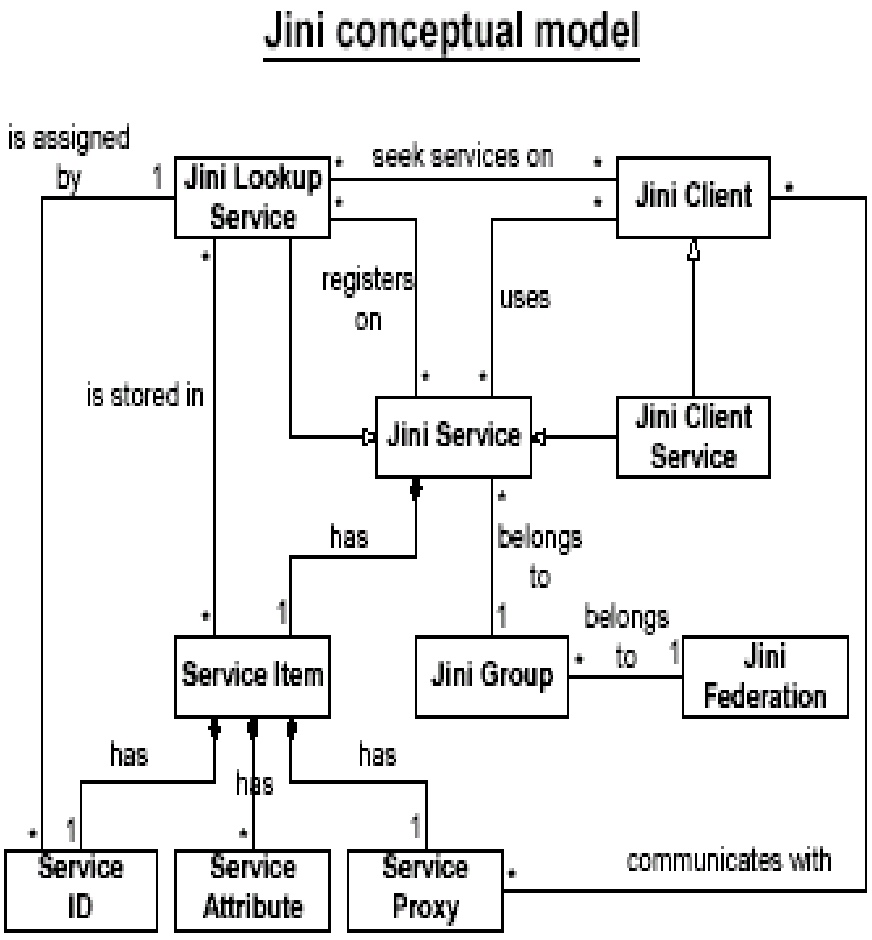


Figure 5.3: Conceptual Model of Jini

5.3 Salient Features of Jini

Jini architecture constitute a number of interrelated modules and protocols which are explained briefly in subsequent paragraphs.

5.3.1 Djinn

A Djinn is a collection of networked Jini devices (Illustration in figure 5.4).

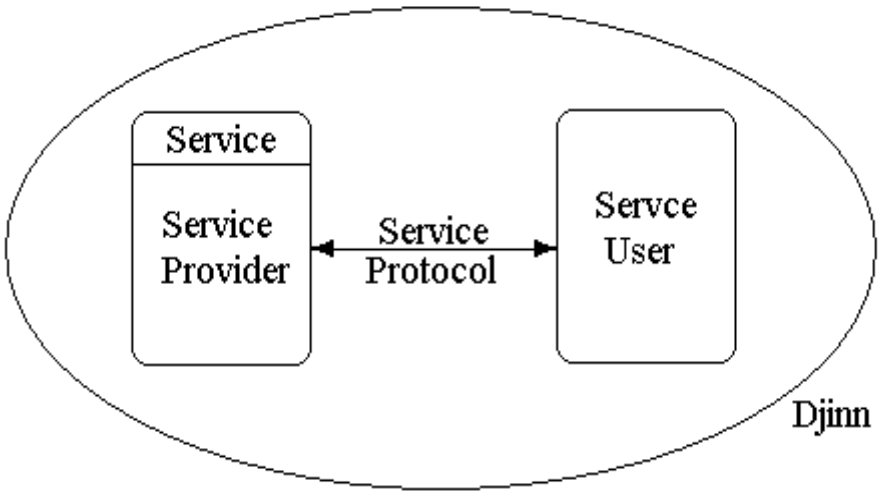


Figure 5.4: Djinn

5.3.2 Services

A service is any kind of resource that can be accessed via the network. Examples: communication channels, hardware, software. Service Users A service user accesses services with the help of the corresponding service protocol. This protocol consists of various interfaces that can be loaded dynamically by the service user.

5.3.3 Jini Service

A Jini service is an entity that can be used by a person, a program or another service. A service may be a computing program, a hardware device or a component of the Jini system. Members of a Jini system federate to share access to services. A Jini system

consists of services that can be collected together for the performance of a particular task. The dynamic nature of a Jini system enables services to be added or withdrawn from the federation at any time. Services in a Jini system communicate with each other by using a service protocol represented by a set of interfaces (written in Java).

5.3.4 Service Item

A Jini service has a service item. A service item is an object that represents this service in the Jini federation. A service item is composed of a service identifier, a set of attributes and a proxy object.

5.3.5 Service ID

A service identifier is a global unique identifier for a service. This identifier is assigned by a lookup service (further detailed) the first time a service registers in the Jini federation. Once a service ID is assigned to a service, the service must remember it.

5.3.6 Service Proxy

A service proxy is an object that encapsulates the mechanisms that a service and a client communicate with. When a client system looks for a service it receives from a lookup service a proxy object enabling the communication with the requested service.

5.3.7 Service Attributes

Service attributes represent relevant characteristics of a service; relevant features that distinguish one service from another in ways that are not reflected by the type of the proxy. In this way, Jini clients can perform rather complex searches of services based on these attributes.

5.3.8 Jini Group

Jini services are structured within groups. A group usually represents a rather small (typically the size of a workgroup) community of services. The default group is called “public”. Group names are only unique within the naming space of a network.

5.3.9 Jini Federation

The Jini federation is an abstract concept that represents the full set of communities (or groups) of Jini Services.

5.3.10 Jini Client

A Jini client is a system that uses a Jini service. A Jini client can eventually be another Jini service or just another entity. In an ideal Jini federation there would be nothing but Jini services that collaborate to perform certain tasks.

5.3.11 Lookup Service

Services are found and resolved by a *Jini Lookup Service (LUS)*. The LUS is the central bootstrapping mechanism for the system. It provides the major point of contact between services. It is essentially a process that keeps track of all of the services that have joined the Jini community. The LUS resembles a *name server*. However, the matching is more powerful. Services that implement some interfaces, or belong to special classes can be searched. A search based on service attributes can also be specified. Services register themselves within a LUS by publishing their *service item* (for illustration see figure 5.5).

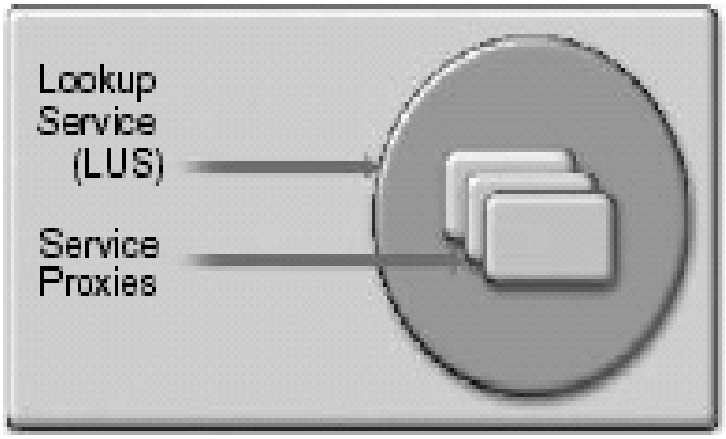


Figure 5.5: Lookup Service

5.4 Mechanism of Jini

The important actors in jini environment that distinguish jini in distributed computing are its discovery protocol, join protocol, lookup service, leasing, remote events and concept of transactions.

5.4.1 Discovery Protocol

Entities that wish to start participating in a distributed Jini system must first obtain references to one or more LUSs**.** The protocols that govern the acquisition of these references are known as the discovery protocols. A Jini discovery protocol is the means by which Jini entities find Jini communities. There are several discovery protocols that can be used according to each particular situation. The "*Multicast Request Protocol*" is employed by entities that seek to discover nearby (on the local network) LUSs as shown in figure 5.6. The "*Unicast Discovery Protocol*" is used when an entity already knows the

particular LUS it wishes to talk to. The "*Multicast Announcement Protocol*" is used by LUSs to announce their presence. As a result of the discovery process, a Jini entity disposes of references to LUSs.

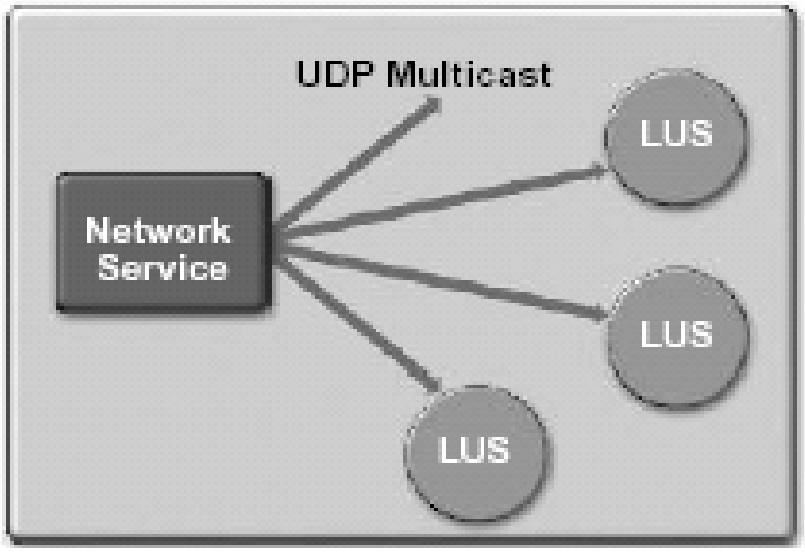


Figure 5.6: Multicast Discovery Protocol

5.4.2 Join Protocol

A Jini entity, using references to LUSs obtained during the discovery process can advertise the services it offers**.** The *join protocol* regulates how services join Jini communities. A service joins a LUS by publishing its service item. The first time a given service joins a LUS, the LUS will assign it a *serviceID*. The service must remember this and use it when it registers itself with all LUSs in the future (see figure 5.7)

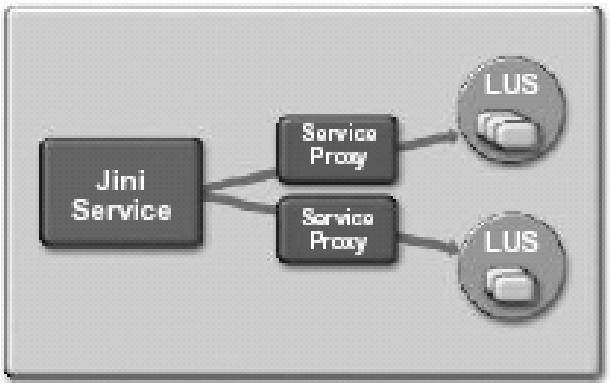


Figure 5.7: Join Protocol

5.4.3 Lookup Protocol

A Jini entity, using a reference to a LUS obtained during the discovery process can search all the service items provided by this LUS to find services of interest. This search can be based on the type of the service proxy, on the unique identifier of a service, or the attributes contained in a service item.

5.4.4 Leasing

In distributed systems, there are situations when different parts of a cooperating group are unable to communicate - either because one of the members has crashed or because the connection between the members of the group has failed. To deal with these problems, the notion of *lease* was introduced. Rather than granting services or resources until that grant has explicitly cancelled, a leased resource or service grant is *time based*. When the time for the lease has expired, the service ends or the resource is freed. The time period for the lease is negotiated when the lease is first granted. Leases may be renewed or cancelled before they expire by the holder of the lease. In the case of no action, caused by a service crash or a network failure, the lease simply expires.

5.4.5 Remote Events

Jini entities occasionally need to be notified when certain interesting changes happen in their environment. To do that, Jini has the notion of remote events. An event is an object that contains information about some external state change that an entity may be interested in, e.g., an expected service joins the Jini community. The notification is sent to listeners that have subscribed interest in receiving a particular event**.**

5.4.6 Transactions

Jini supports the notion of transactions. Transactions provide a way to group a series of related operations so either all the operations succeed, or all the operations fail. Jini uses a two-phase commit process. First, a transaction manager signals each of the participants to go into a *pre-commit* phase. Then, each of these participants notifies the transaction manager whether the operation was successfully completed or if it failed. If any of the operations failed, the transaction manager tells every participant to *abort*. Otherwise the transaction manager tells the participants to *commit,* which causes them to make their changes permanent. Jini only defines the transaction protocol as an interface but it does not implement the protocol. The transaction protocol must be implemented by each service.

CHAPTER 6

ROBOTIC SERVICE

6.1 The Robot

The LEGO Mindstorms Robotics Invention System (RIS) represents a different attitude. The RIS is a boxed set that includes standard LEGOs, various motors and sensors, and a micro controller to control it all (known as the RCX), for $200. As a result of Lego's openness to (and in some cases, even support of) third party developers, the RIS has quickly and quietly become the defacto standard for home robotic enthusiasts [5]. Developers have a wide range of language choices, ranging from the point and click interface LEGO provides, to TCL, to Java. Educators of all age groups have chosen to use the RIS to help teach concepts such as Artificial Intelligence, programming, and robotics to their students. Researchers use the RIS to help test and refine ideas. LEGO has successfully broken down the barriers between the general population and robotics.

6.2 Teleoperation

Means "doing work at a distance". Although by "work" we mean almost anything. What we mean by "distance" is also vague: it can refer to a physical distance, where the operator is separate from the robot by a large distance, but it can also refer to a change in scale, where for a example a surgeon may use micro-manipulator technology to conduct surgery on a microscopic level.

6.3 Telerobotics

Mean "robotics at a distance". Many people, including us, tend to use this interchangeably with "teleportation", which is probably a bit sloppy of us. Telerobotics is

an interesting next generation application in which some form of robotic device is controlled from a remote location over some kind of telecommunications network.

The next generation of intelligent robotic and software agents will be characterized by being more general than the current generation in at least three respects. First, they will be able to successfully carry out multiple, diverse, and possibly interfering tasks in changing and partly unknown environments. Second, they will be able to improve their performance by autonomously adapting their control software for the kinds of tasks they are given and the environments they are to operate in. Third, they will be able to robustly perceive a substantially larger part of their environment.

6.4 Telepresence

Mean "feeling like you are somewhere else". Some people have a very technical interpretation of this, where they insist that you must have head-mounted displays in order to have telepresence. Other people have a task-specific meaning, where "presence" requires feeling that you are emotionally and socially connected with the remote world. It's all a little vague at this time.

Telerobotics is an interesting next generation application in which some form of robotic device is controlled from a remote location over some kind of telecommunications network. Telerobotics has many potential uses, including medical, manufacturing, security, and bio-hazard applications, among many others [7]. These applications, naturally, have much quality of service constraints, requiring near-real-time responsiveness to avoid potentially disastrous consequences.

6.5 Deliberative Architectures

Deliberative architectures are based on symbolic artificial intelligence, which is in turn, based on the physical symbol system hypothesis.

"*A physical symbol system has the necessary and sufficient means for intelligent action*"

Deliberative control takes into consideration all of the available sensory information and amalgamates them with all the controller's internal "knowledge" to create a plan of action. A symbolic model of the world is explicitly represented and decisions are made based on logical reasoning. The control searches through all the possible actions plans until it finds a suitable one. This search sequence can take a long time and is hence not suitable in situations where the robots are expected to react quickly. Furthermore, there is often a problem in translating the real physical world into an accurate and sufficient symbolic representation for the robot to make meaningful decisions. Prodigy is an example of a deliberative learning architecture.

6.6 Reactive Architectures

In sharp contrast to deliberative architectures are reactive architectures, where the perception is tied closely to the effectors action. The architecture does not entail any kind of symbolic world model and does not use complex reasoning. The reactive control is essentially a reflex mechanism where stimulus-response pairs govern actions. The main advantage of robots with reactive control is that they respond quickly to a changing environment where no a priori information is available. The system requires small amount of memory and does not compute or store representations of the world. The inability to learn over time is perhaps is main drawback of reactive architectures.

6.7 Behavior-Based Architectures

A variety of robot control technologies have been developed that are significantly lower in computation than the traditional sense/plan/act cycle that has been used on Robby, Ambler etc. These techniques include the Subsumption architecture control and several others. Collectively they are known as behavior control. The main differences between behavior control techniques and more traditional planning techniques are traditional systems have several expensive processes that operate serially (the perception system feeds into the modeler which feeds into the planner, etc). Behavior techniques use robot capabilities that run in parallel (the go to goal, avoid obstacles, keep moving, etc behaviors all run all the time). Traditional systems make extensive world models, behavior control systems use minimal world models or none at all. Traditional systems are highly deliberative (they plan out sequences of action, compare the results of those actions to their expectations, and then replan accordingly).

A behavior-based architecture is one that is built from a collection of behaviors that achieve or perform certain goals. By combining and controlling these behaviors, complex robotic actions can emerge. This class of architecture originated from the subsumption architecture developed in the mid-1980s. In this architecture, the robot control is built in a bottom-up fashion, with task-achieving actions/behaviors as components. These components can be executed in parallel and newly added components and layers are built on top of existing ones, without modifying the way the existing components behave. The notion of internal model is absent in this architecture as “the world is its own best model”. The main limitation of this type of architecture is that the robot’s goals must be capable of being represented implicitly in the controller’s structure according to a fixed, pre-compiled ranking scheme. The Genghis is a six-legged robot architecture using the subsumption model [10].

CHAPTER 7

SYSTEM IMPLEMENTATION

7.1 Introduction

The purpose of system development is to transform the complete design into executable computer program which may than be tested and implemented as a new system. For the software development it is necessary to identify functional requirement of the system and for each individual task a program module is developed. Thus software development is divided in modules and each module is allocated a single task. This chapter describes the coded program and modules developed for the implementation of new system. Several modules have been written for creation and maintenance of the system. Each module consists of several programs and each program performs a separate function.

7.1.1 Robot's Embedded Real Time Software

This module is responsible for conducting the execution of commands from within the robot's microcomputer and generates the appropriate responses back to the remote PC based module. This module comprises of a thread scheduling system which acts as a task scheduler. It maintains two basic threads based on priorities; the navigation and hurdle detection thread. The latter one has higher priority and can overrule the former one changing per real time scenario. For development, Java was first of all installed inside the micro-computer on 16kb, using the world's smallest virtual machine namely LEJOS [6]. The following files were developed in this module:

TeleRobotBehavior.java : The real time embedded thread scheduling system.

BumperHit.java : Thread program for hurdle detection.

NavigationController.java : Thread controlling the navigation of robot.

7.1.2 PC Side Module

This module keeps contact with the embedded module and all the remote method calls are received at this side and then the commands are sent to the robot. The Jini service runs at this side too. All other services like the Video capturing, image processing and security surveillance services run from here. These codes were developed using a variety of software technologies like JMF, LEJOS Vision API, JavaTwain, Swing, JINI and Sun's Robotics Development Kit.

RCXPortInterface.java : The basic interface for the robotic services.

RemoteRCXPort.java : A remote version of the above interface.

RCXPortImpl.java : Implementation of robot service to act as proxy.

JiniService.java : The Jini service registration code.

SecuritySystem.java : The security system which uses motion detection.

7.1.3 The Client Side Module

This module covered the requirements for the remote client. The robot's client needed to access the remote service, so it had to be provided a mechanism to download all the required classes and interfaces over the network, which it does using a class

server. Apart from class server, a lookup service, a web service and the robot's service are also required to be up and running over the network for executing this module. It also comprises of providing proper and interactive but easy to navigate user interfaces to the client. The files that were developed in this module are:

JiniServiceClient.java	:	Discovers robot's service and gets the required stub
RemoteControl.java	:	GUI for remote security monitoring and object fetch
NavigationWindow.java	:	GUI for controlling robot's navigation
GUIInterface.java	:	The program containing above files.
DiscoveryApplet.java	:	The applet that calls GUI Interface from client side.

7.2 Software Selection

The project's development required software technologies from various sources, but to maintain the compatibility of the whole code and many other reasons, the base technology selected was Java. All the kits used in this project are either based directly on Java or provide a Java based implementation of other technologies. The reasons for deciding upon Java were portability, support for code mobility over the network, platform independence, powerful and secure programs, pure Object oriented approach, good Multimedia and Image processing capabilities, reusable components/code and exploring Java in connecting resource constraint devices.

7.3 Graphical User Interfaces (GUI)

GUI in figure 7.1 provides an interface to the user to look for the required service on the network.

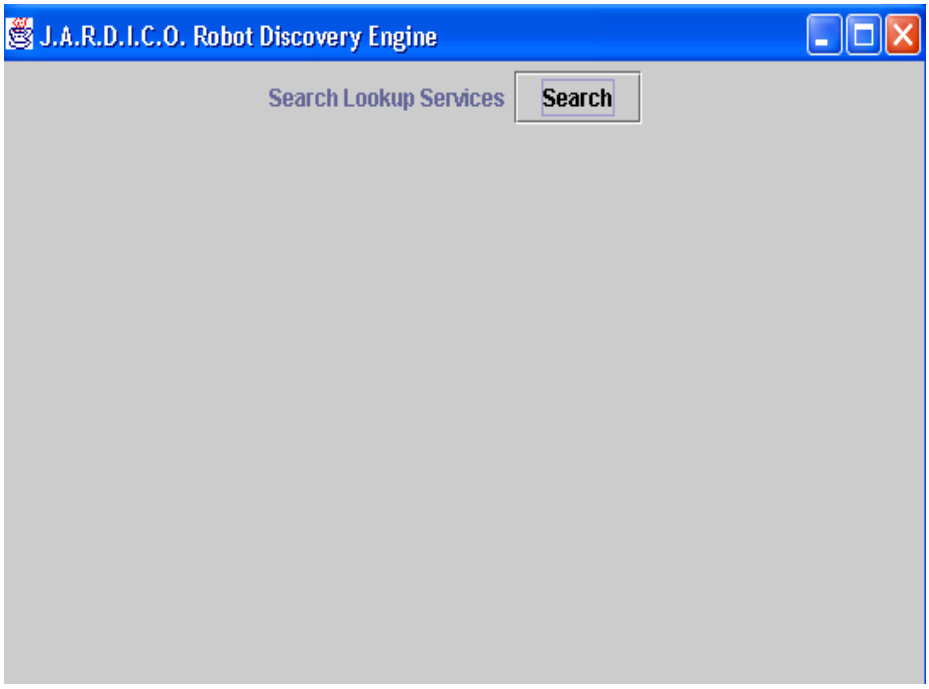


Figure 7.1: *JARDICO Search for Lookup Service - GUI*

Figure 7.2 shows that the lookup service has been found by the discovery process and unique ID of the LUS is shown, and also the required service can be retrieved through this interface.

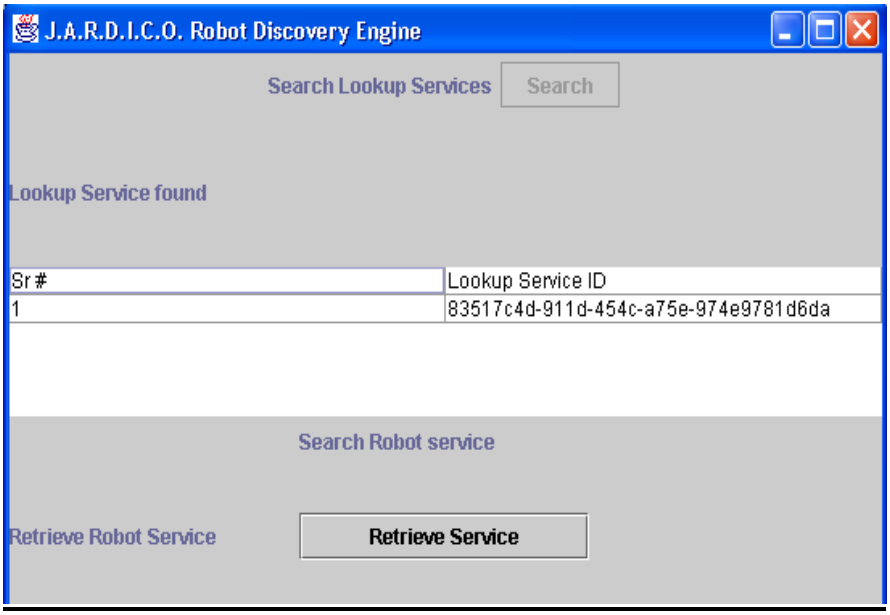


Figure 7.2: *J*ARDICO - GUI to Retrieve Service Proxy

Figure 7.3 shows the main interface through which any of the various services can be selected. In the background http, jini, reggi servers are shown.

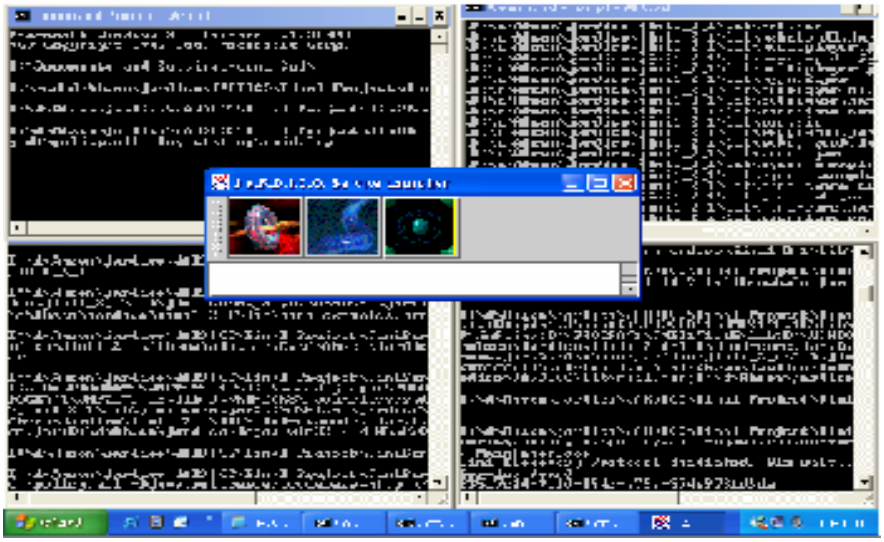


Figure 7.3: GUI for Robot Service Launcher

Figure 7.4 shows the interface for the remote navigation controller module.

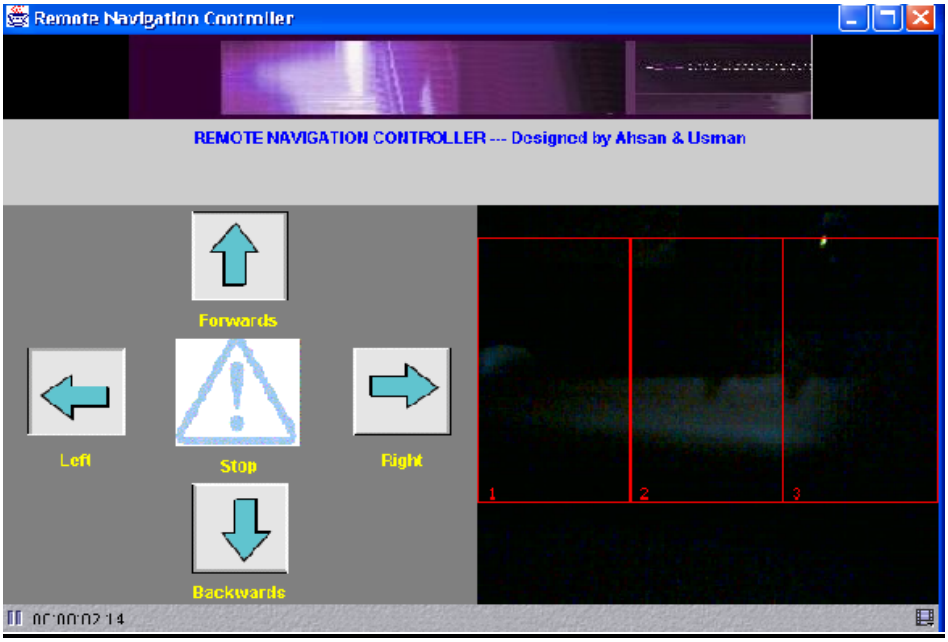


Figure 7.4: GUI for Remote Navigation Controller

Figure 7.5 shows the interface for the security surveillance module.

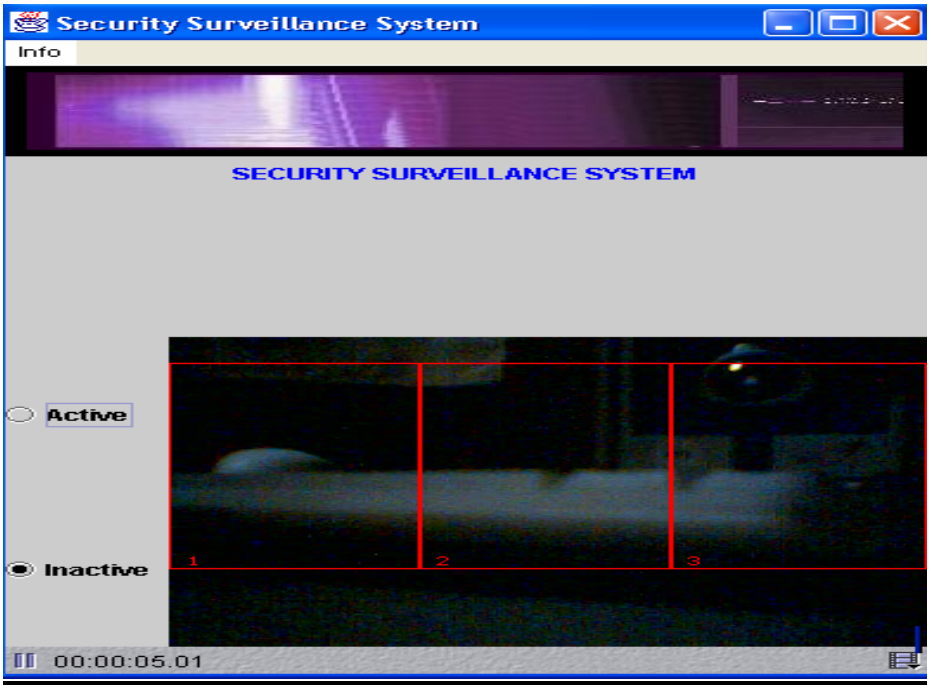


Figure 7.5: GUI for Security and Surveillance Module

Figure 7.6 shows the interface for the motion detection and tracking module.

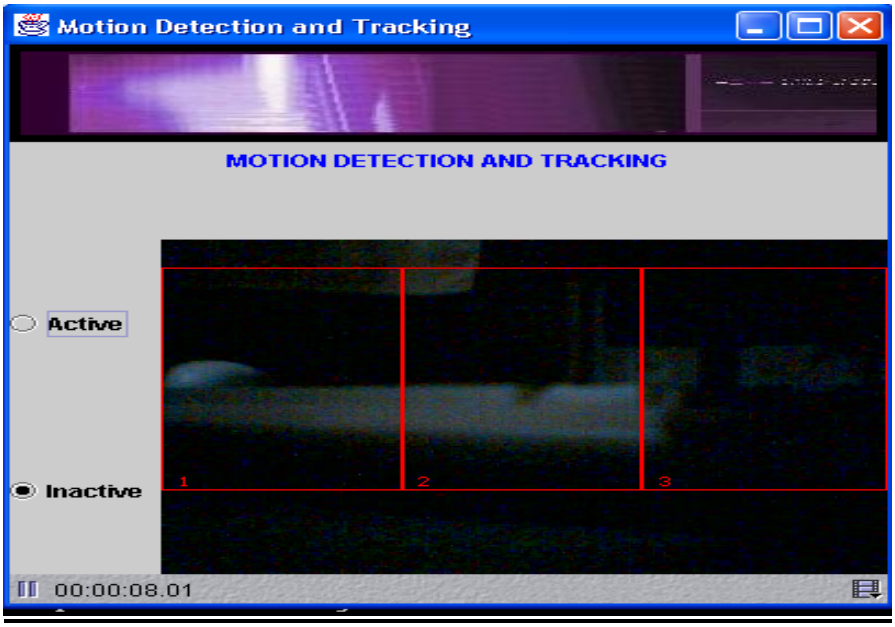


Figure 7.6: GUI for Motion Detection and Tracking Module

CHAPTER 8

FUTURE WORK AND CONCLUSION

8.1 Future Work

Success stories in favor of Jini have started popping, fueling to the already well established media hype surrounding the technology since day one. And besides all the crushing effort by its competitors, which includes some software giants, Jini is still holding fort quite firmly and staying resolute in the market. Infect it will not be an overstatement if we say that Jini has established a nice and clear cut sense of programming hardware/software services for a distributed world. It's a unique technology and its features are matchless which can create great tides in the ways we live and the day to day life styles we are accustomed to [9].

8.1.1 Jini and Echelon Power the Home Network

Echelon's vision is simple-extend the power and benefits of networking so that even everyday devices can enjoy connectivity to anything, anywhere, anytime. Echelon uses Jini technology to group devices together into a service network, secured through the Java™ language.

Echelon Corporation is a worldwide leader in networking everyday devices, with thousands of companies developing products using Echelon technologies and millions of nodes installed worldwide in building, industrial, transportation, home, and other environments. . To achieve this goal, the network must be plug and play, erase distinctions between hardware and software, allow spontaneous networking, enable a service-based architecture and provide the simplest system possible

8.1.2 US Army Employing Jini for High Tech Warfare

Designing and maintaining a mobile command post so that it functions under sometimes chaotic battlefield conditions is a unique challenge. Yet that's just what the U.S. Army Program Executive Office for Command, Control, and Communications Systems (C3S) asked Sun and IBM to do. The two companies were recently awarded a contract to utilize Sun's Jini technology in a dot-com solution linking a network of complex, sophisticated equipment in its battlefield command spots, known as Tactical Operation Centers (TOCs). Designed to be mobile, TOCs must also provide a stable, secure, fixed-configuration communications network that can automatically adapt to changes and be self healing in the event that a TOC is destroyed or a device in the network federation is removed, be compatible with off-the-shelf computing devices that are spontaneously added to the network and accommodate new TOCs that are added to the network, without requiring intervention by technical personnel.

8.1.3 PROSYST Developing Next Generation Electronics Using Jini

ProSyst Software AG is a leading vendor of Java technology-based server software. The company's goal is to link all devices to each other, as well as to the Internet. The ProSyst product family includes mBedded Server, a small footprint server for consumer network services, and mBedded Developer, a development tool for creating services for embedded systems. To create the mBedded Server, ProSyst turned to Sun's Java and Jini technology. Using the versatility and scalability inherent in the Java architecture, ProSyst is building the foundation for connectivity that will provide unlimited offerings of new products and services in an ever-changing, demanding world.

8.1.4 Appropria's Integration of Distributed Database Components

Enter Appropria, a software company headquartered in Pleasanton, California, that is engaged in the design, building, and marketing of applications that create an Intelligent Interface between users and corporate information databases. Appropria's Plant Advantage is the first product based on the company's innovative Workplace Information Integration technology.

The benefits of using Plant Advantage are many, with the most immediate in time savings. Appropria's initial research suggests that the average worker spends almost an hour a day finding or manipulating information. Plant Advantage could reduce this by half, cutting up to three weeks per year, per worker. In addition to time and productivity gains, customers will benefit in other ways.

8.1.5 EKO Systems Connects Medical Data and Equipment

EKO systems, incorporated, standardized on Sun Microsystems' hardware and software systems: workstations, servers, and storage systems, as well as the Solaris™ Operating Environment, the Java™ architecture, and Jini™ network technology.

Jini technology gave the system, called Frontiers, the capability to connect to any type of medical equipment—including physiologic monitors, ventilators, and anesthesia machines—without requiring clinicians to even think about configuration data or software drivers. "A critical part of making these systems usable is making them manageable," said Koch, cofounder and chief medical officer at eko systems, in Fairfax, Va. "The whole point of using Jini technology is that the device interface can be managed by the clinicians, and all they have to know is how to plug things in and read an LCD (liquid crystal display) screen."

8.1.6 Sun and Raytheon Create Open, Adaptive, Self Healing Architecture

The U.S. Navy has introduced major changes to the way it will design, operate and maintain its surface warships in the 21st century. One significant challenge facing Navy is the need to reduce total ownership cost while increasing capability. To do so, they must leverage commercial cutting-edge technology in a computing architecture that provides an adaptive, flexible system. Recognizing the need for an innovative solution, the Navy outlined very high-level, strategic design goals. For example, the Navy needs a computing architecture that is adaptive to legacy systems and is designed to adapt to future technologies without knowing what those technologies are. To help maintain development and operational costs and reduce manpower requirements, the Navy wants to make better use of commercial technology, particularly information technology. And the Navy has determined that its total ship computing architecture (TSCA) will be based on open-systems architecture.

Architects from Sun Professional Services' Java Center have provided Raytheon and the U.S. Navy with an architecture that can be used to validate key aspects of the DD 21 total ship computing architecture. Sun is providing the leadership and vision to craft the software infrastructure and programming model required to meet the Navy's requirements. This model empowers the development and deployment of Jini service components, enabling the development of both tactical and non-tactical software along with integration of legacy systems for the TSCA. At a Navy-focused industry show in April 2000, Sun and Raytheon demonstrated an architecture that showed integrated concepts to increase the quality of life for the sailors onboard ship, allow the crew to manage the ship more effectively and provide the underlying tools to enable the crew to fight more effectively. The architecture included implementation of Internet technology through the iPlanet™ portal server, Sunray thin client architecture and spontaneous networking, provided by Jini technology.

The network being designed by Sun and Raytheon will include multiple secure data centers on board each ship and in shore-based facilities. The expected hardware baseline includes Sun Enterprise™ servers ranging from Ultra™ 5s to 4500s to Sun Enterprise™ 10000s using MAJC™ processors and running Sun's Solaris Operating Environment. Other network components include Sunray enterprise appliances, smart cards based on Java Card™ technology, advanced terminal emulators and wireless connectivity through iPlanet portal server.

8.2 Conclusion

Jini technology leverages the unique capabilities of the Java platform to enable Java Dynamic Networking, a fundamentally different approach to networking that only the Java platform can support (see "What makes Java Dynamic Networking different?"). Jini technology is interesting today first because the breadth and credibility of successful Jini technology-based applications now deployed has validated the technology among early majority users, and second, because dynamic networking capabilities are needed by a many major new markets, from RFID systems at the network edge to utility computing infrastructures at its core.

Jini technology is today supporting many companies' needs to keep their systems up and running while dynamically integrating in house or third-party services whenever they are available. The technology is helping these users deal with service failures, updates, and implementation details that are beyond their control, often stemming from their need to integrate services from external sources. Also, by isolating clients from the need to implement any of the multiple proprietary communications protocols that many different service providers use, Jini technology is simplifying these users' approach to integrating services that use a variety of communication protocols.

The Sun Java Center has shown its expert approach towards innovative architecture, mentoring, development and execution excellence for this engagement. The preliminary test results demonstrate the utility of Sun’s technology and services in meeting the challenges of true mission-critical applications with the Navy’s next generation architecture. Bruce Cooper of Raytheon agrees, “We’re very encouraged by the success of the architecture and being able to achieve the challenging goals for the DD 21 ship computing environment. That’s a function of the power of the Java and Jini technologies and the knowledge, skills and expertise of the Sun Java Center architects."

APPENDIX

HARDWARE SPECIFICATIONS

RCX - The Robot Microcomputer

Lego RCX with Sensors and Motors

Specifications

Hitachi

16 x Kbytes ROM

32 x Kbytes RAM

3 x Sensor inputs, 3 x Motor Outputs

1 x IR port for communication with the Infra red tower

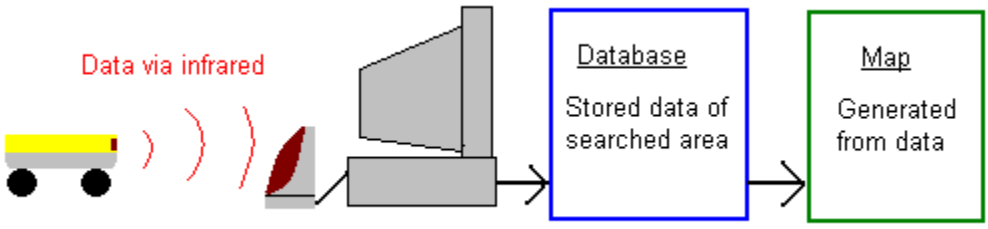
5 x digit LCD

16 x Mhz Speed

Firmware (16K) is downloaded via IR

USB IR Tower

Infrared Tower



Basic Network Diagram

BIBLIOGRAPHY

[1] Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides

[2] www.Upnp.org

[3] LEGO Mindstorms. Homepage.h*ttp://mindstorms.lego.com*

[4] Jini in a Nutshell by Scott Oaks and Henry Wong

[5] A Programmer's Guide to Jini Technology by Jan Newmarch

[6] Lejos Operating System Homepage. *http://lejos.sourceforge.net.*

[7] Object Mobility in the Jini Environment Simplify the Installation and Configuration of Jini Applications by Frank Sommers. Published in JavaWorld, January 2001

[8] www.salutation.org

[9] MIT Media Labs

[10] http://www.llnl.gov/automation-robotics/tc.html