# Seminar Report

# On

# Jini Technology

## Career Institute of Technology and Management

## Aravalli Hills, Faridabad

## Department of Information Technology

**(JAN-2011-JUNE 2011)**


**Submitted To:**

**Mrs. Bindiya Ahuja**

**(Assistant Prof.)**

**Submitted By:**

**Nikhil Girdhar**

**8 IT-B1**

**274096**

# INDEX

# CHAPTER 1

# Introduction To The Project

## 1.1 **Jini Technology** : An Overview

Jini Technology is based on Java™ technology. It Enables all types of devices to simply connect into impromptu networks, making access to and delivery of new network services as simple as plugging in a telephone.It Enables all types of digital devices to work together in a community put together without extensive planning, installation, or human intervention.
Each device provides services that other devices in the community may use.

An impromptu community is another way to describe what happens when two or more devices using Jini technology come together to share their services. It is impromptu, because the devices do not need any prior knowledge of each other in order to participate. Jini technology allows devices to dynamically establish communication to share and exchange services across a network. The impromptu community is a dynamic environment that eliminates the need for configuring devices or installing drivers. For example, imagine plugging a camera into a network. The camera instantly joins the network without drivers to install, floppies or a CD-ROM to insert, or keyboard commands to type. The camera identifies itself and offers its services. If the camera could talk, it would be saying: "I'm a camera, anyone need pictures?" You might have a laptop that uses Jini technology to join this network. You could access the camera, snap a photo, route it to your own disk drive (which also uses Jini technology to communicate with other devices), or send it to another device for printing .
These devices provide their own user or programmatic interfaces, which ensures reliability and compatibility.
The overall goal is to turn the network into a flexible, easily administered tool on which resources can be found by human and computational clients. The focus of the system is to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.

## 1.2) History of Jini Technology

- The idea of the Jini system sprang from Sun cofounder Bill Joy at Sun Aspen Smallworks R&D lab in 1994.
- Under the leadership of Bill Joy and Jim Waldo
  - Ann Wollrath, the inventor and designer for Java Remote Method Invocation (Java RMI)
  - Ken Arnold, the designer of JavaSpaces™ technology.
  - Bob Scheifler, a principal of the X Consortium and designer of Lookup/Discovery.
- Based on the potential to create a ubiquitous network by leveraging the unique distributed computing characteristics of Java technology .
- On January 25, 1999, Jini was officially launched and the technology is available for download.
- Sun currently has agreement with a wide range of technology companies to develop Jini technology services, both hardware and software . Company includes Axis, Canon, Datek, Epson, FedEx, Mitsubishi, Norwest Mortgage, Novell, ODI, Quantum, Seagate, Toshiba, Computer Associates, Oki, and Salomon Brothers.
- It is expected that by using Jini technology, we can enable infinitely connected network of services into which anyone will be able to plug-and-participate anytime, from anywhere, using the simplest possible technology.

## 1.3) Goals

The main goal of Jini architecture is to provide a type-safe and reliable way of providing services in the network. It is a framework for designing and implementing OO distributed systems. The aim is to allow services to be easily available and removable on a network to anyone who can reach them.

Service deployment means plugging the service to the network, and it becomes visible and available, as easily as possible, to those who want to use it. The service can be pure software, hardware, or a combination of them. The idea of a flexible network is expanded, because all the services available are reachable to other services and clients, and removing

the service is not any harder. The network can reorganize itself to be able to continue working after changes.

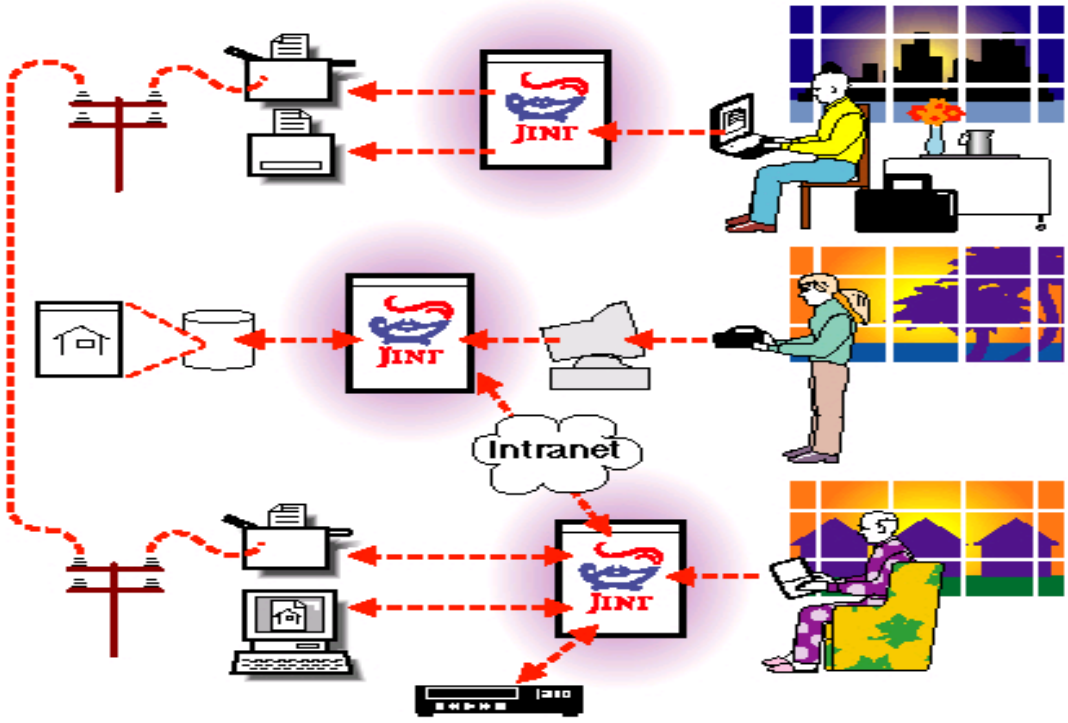
Jini technology changes the idea of network applications to the idea of a service-based network. Everything is a service. The goal is that this architecture allows any device with a processor, some memory and a network connection to offer and use services available. Jini contributes in implementing the services and making them available on the network. It is also possible to automate some implementation issues with Jini. Because of the Jini's new programming models, the developer is forced to acknowledge the difficulties of the distributed computing. Thus, this framework offers better capabilities to deal with the issues, which are cannot be handled bythe traditional distributed systems.

## 1.4) Where Jini Technology be Used

Almost in every devices that passes digital information in and out like

1. Traditional computer hardware and software
2. Consumer appliances such as personal digital assistants (PDAs), digital cameras, TVs, DVD players, cell phones, and CD players.

## Consider the Example Scenario

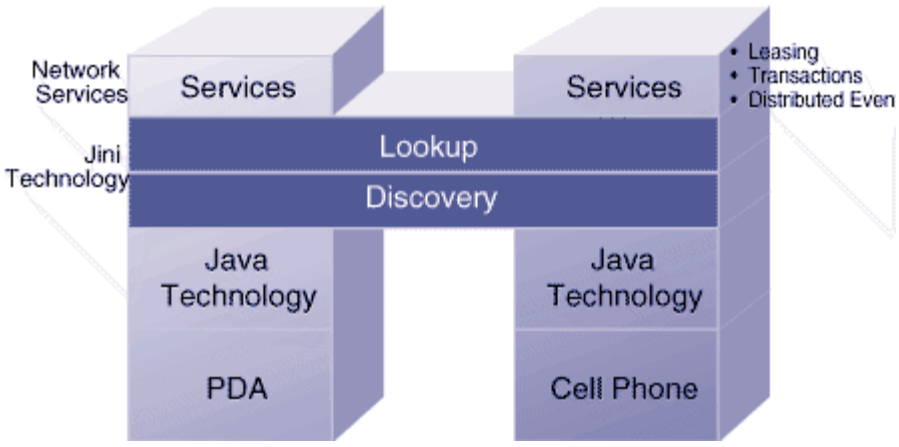



In this example , there are three professors say Bob, sue and Dave. They are at the different site with each site is having is on network. These networks are connected with each other's by using the jinni technology. What these professors are doing is, they are working over a same project with the devices on their own network uses the services provided by the devices on the another network.

# CHAPTER 2

# Architecture of Jini Technology

## 2.1) Architecture Overview




## Architecture

When you plug a new Jini-enabled device into a network, it broadcasts a message to any lookup service on the network saying, in effect, "Here I am. Is anyone else out there?" The lookup service registers the new machine, keeps a record of its attributes and sends a message back to the Jini device, letting it know where to reach the lookup service if it needs help. So when it comes time to print, for example, the device calls the lookup service, finds what it needs and sends the job to the appropriate machine. Jini actually consists of a very small piece of Java code that runs on your computer or device.

Jini is a set of APIs and network protocols that can help you build and deploy distributed systems that are organized as *federations of services.* A *service* can be anything that sits on the network and is ready to perform a useful function. Hardware devices, software, communications channels -- even human users themselves -- can be services. A Jini-enabled disk drive, for example, could offer a "storage" service. A Jini-enabled printer could offer a "printing" service. A *federation of services,* then, is a set of services, currently available on the network, that a client (meaning a program, service, or user) can bring together to help it accomplish some goal.

Jini defines a *runtime infrastructure* that resides on the network and provides mechanisms that enable you to add, remove, locate, and access services. The runtime infrastructure resides on the network in three places: in lookup services that sit on the network; in the service providers (such as Jini-enabled devices); and in clients. *Lookup services* are the central organizing mechanism for Jini-based systems. When new services become available

on the network, they register themselves with a lookup service. When clients wish to locate a service to assist with some task, they consult a lookup service.

The runtime infrastructure uses one network-level protocol, called *discovery*, and two object-level protocols, called *join* and *lookup.* Discovery, illustrated in the middle of Figure 1, enables clients and services to locate lookup services. Join enables a service to register itself in a lookup service. Lookup enables a client to query a lookup service for services that can help the client accomplish its goals.

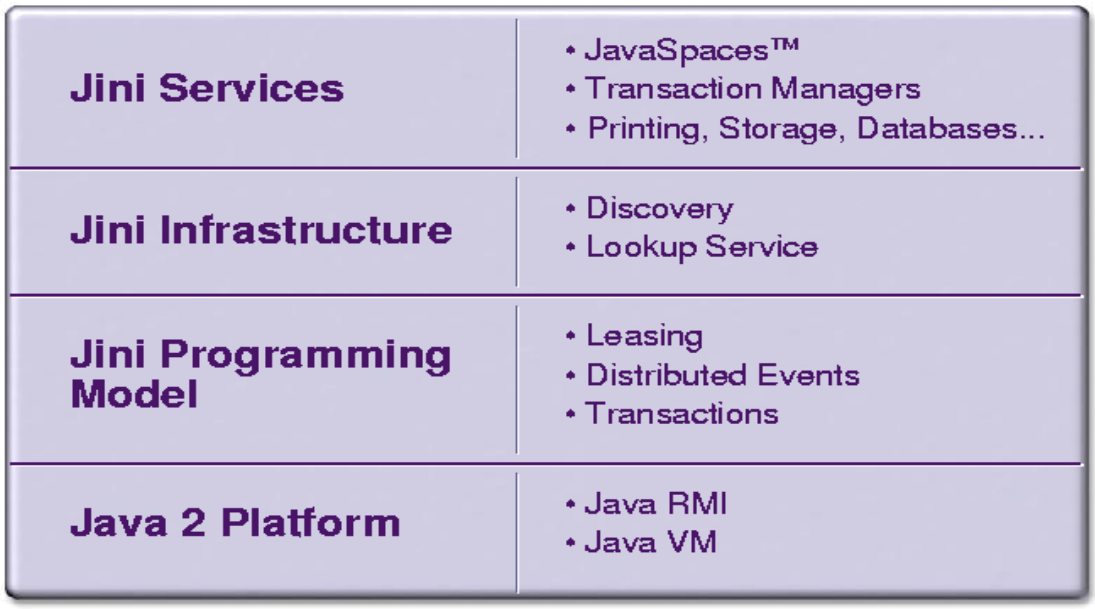
Discovery works like this: Imagine you have a Jini-enabled disk drive that offers a persistent storage service. As soon as you connect the drive to the network, it broadcasts a *presence announcement* by dropping a multicast packet onto a well-known port. Included in the presence announcement is an IP address and port number where the disk drives can be contacted by a lookup service.

Lookup services monitor the well-known port for presence announcement packets. When a lookup service receives a presence announcement, it opens and inspects the packet. The packet contains information that enables the lookup service to determine whether or not it should contact the sender of the packet. If so, it contacts the sender directly by making a TCP connection to the IP address and port number extracted from the packet. Using RMI, the lookup service sends an object, called a *service registrar,* across the network to the originator of the packet. The purpose of the service registrar object is to facilitate further communication with the lookup service. By invoking methods on this object, the sender of the announcement packet can perform join and lookup on the lookup service. In the case of the disk drive, the lookup service would make a TCP connection to the disk drive and would send it a service registrar object, through which the disk drive would then register its persistent storage service via the join process.

## 2.2) Component of Jini Technolgy

- Jini Services
- Jini Infrastructure
- Jini Programming Model

- Java 2 Platform




### 2.2.1) Jini Infrastructure

- Infrastructure is the set of components that enables building a federated Jini system:
  - *The lookup service*, which serves as a repository of services. It reflects the current members of the federation and acts as the central marketplace for offering and finding services by members of the federation
  - *The discovery/join protocol*, a service protocol that allows services (both hardware and software) to discover, become part of, and advertise supplied services to the other members of the federation
  - A distributed security system, integrated into RMI, defines how entities are identified and how they get the rights to perform actions on their own behalf and on the behalf of others.

### 2.2.2) Jini Programming Model

A set of interfaces that enables the construction of reliable services, including those that are part of the infrastructure and those that join into the federation

- Leasing interface: defines a way of allocating and freeing resources using a renewable, duration-based model
- Event and notification interface: a notification mechanism between services and resources that stretches beyond machine boundaries
- Transaction interface: wrapping of multiple operations on multiple services into a single unit of work that either completely succeeds or is rolled back.

### 2.2.3) Java Services

Hand-coded using the programming model. Can be programmed to do any task suitable of the resource.

- Services appear programmatically as objects written in the Java programming language, perhaps made up of other objects.
- A service has an interface which defines the operations that can be requested of that service.
- The type of the service determines the interfaces that make up that service and also define the set of methods that can be used to access the service.
- A single service may be implemented by using other services.

### 2.2.4) Java 2 Platform

- **Java RMI( Java Remote Invocation)**

  Java RMI is the base infrastructure of the Jini technology, because only RMI provides the capabilities that enable the Jini software architecture. Java RMI allows the passing of objects, including their behavior, by their actual Java software type. This allows the passing of sub-types of an object where the base type is expected, and allows the code that enables different implementations of

the same service interface to be moved to the client of the service on demand. Without Java RMI, Jini would not be possible.

- **Java Virtual Machine (Java Virtual Machine)**

  A **Java Virtual Machine** (**JVM**) enables a set of computer software programs and data structures to use a virtual machine model for the execution of other computer programs and scripts. The model used by a JVM accepts a form of computer intermediate language commonly referred to as Java bytecode. This language conceptually represents the instruction set of a stack-oriented,capability. Sun Microsystems states there are over 4.5 billion JVM-enabled devices.

# Overview Of Jini Infrastructure

## Jini Infrastructure

The Jini™ technology infrastructure is built around the model of clients looking for services. The notion of a service encompasses access to information, computation, software that performs particular tasks, and in general any component that helps a user accomplish some goal. Services can themselves be clients of other services, and can be grouped together to provide higher-level functionality.

The Jini architecture requires a service to be defined in terms of a data type for the Java™ programming language that can then be implemented in different ways by

different instances of the service. A service can be a member of many different types, allowing a single service instance to provide a variety of functionality to clients. This is a standard practice in object-oriented software. However, the distributed nature of a system of Jini technology-enabled services and/or devices allows data types for the Java programming language to be implemented in a combination of software and hardware in a way that is unique.

The core of the idea that enables this implementation flexibility is quite simple. Services are defined via an interface, and the implementation of a proxy supporting the interface that will be seen by the service client will be uploaded into the lookup service by the service provider. This implementation is then downloaded into the client as part of that client finding the service. This service-specific implementation needs to be code written in the Java programming language (to ensure portability). However, since this code comes from the actual instance of the service being used, it can know in great detail the specifics of the particular service implementation for which it is the proxy. Not only can the code that is downloaded know about the software used to implement the service, the code can know specifics about the hardware on which the service resides. In the limit case of this, the hardware could be all that there is to the service, and the downloaded software could act as a network-level device driver, taking method calls in the Java programming language from the client and generating specific, hard-coded requests to the hardware on the other end of the network wire.

This approach to services requires that there be a piece of code written in the Java programming language that can be downloaded by the client of the service and some hardware that ultimately runs the service. Between these two points, however, there are a number of options concerning the software structure, hardware structure, and location of components that can be chosen by the service provider. These options allow trade-offs to be made in the functionality provided and the cost of the underlying hardware.

## 3.1) LookUp Service

The Jini Lookup service is the heart of a Jini community. The lookup service is similar in principle to the naming service used in other distributed computing paradigms like CORBA's COS Naming Service. It holds the registrations of all other services available in the Jini community. An application that wants to use a Jini service finds the desired service by looking for the service's registration within the lookup service. A Jini service must register itself with the Jini lookup service in order to be used. The Jini lookup service is just another service. The service interface of the lookup service defines all the operations that are possible on the lookup service. It defines the way in which clients in a Jini community locate services. There may be more than one instance of the lookup service running in the community. This is to provide a certain level of redundancy. Jini lookup services are organized into groups. There is a default group called the public group. One can create groups with any name. When he start a lookup service, he can specify which group it should hold registrations for. When he search for lookup services, he can specify which groups he is interested in.

As part of its responsibility, the Jini Lookup Service is always listening for the discovery-request datagram packets. Once it hears one, the discovery process begins. If the Lookup Service supports any one of the groups requested, the requesting service may be permitted to join. The Jini Lookup Service notifies the requesting service, which responds with the new service's programming interface - a serialized Java object, that the Lookup Service stores. Since a service may join or leave the network at any time, the loose term of federation is given to the Lookup Service and its current set of registered services. The newly registered service is granted a lease for its registration for a duration determined by those who initially configure the system. If the lease is not renewed, the Lookup Service will drop the service from the federation.

## 3.2) Discovery

A client that wants to use a particular service finds the appropriate server by looking in the Jini lookup service. But before that the client has to locate the Jini lookup service. This can be done by a process called discovery. Jini specification for this defines at the network

level the protocol by which clients can find the lookup service. There are two ways in which a client can discover the Jini lookup service:

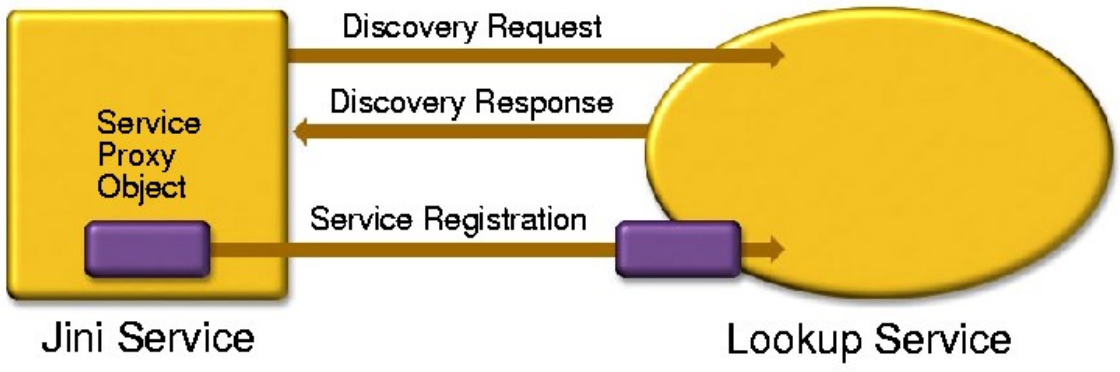
Multicast discovery - the client sends out a multicast request of a specified format. All Jini lookup services that come across the request will respond to it, and the client is said to have discovered the lookup services.

Unicast discovery - In this, the client should attempt to connect to a lookup server having known the existence and location of the lookup server. This is in a sense not discovery. The discovery protocol uses a combination of both unicast and multicast discovery in order to find lookup servers.

Clients will discover all lookup servers that are within the multicast radius of the network. The multicast discovery packet will be broadcast on the local network. Routers that join two networks may or may not route the multicast packet between the two networks depending upon the rules of the router. Many routers will not pass any multicast packets between networks and for these networks; the multicast radius is equal to the local network. Otherwise, the router will forward the multicast packet based on its time to live (TTL) value, which is decremented each time the packet passes through a router. When the TTL value reaches zero, the packet is no longer forwarded onto new networks.

## 3.3) How Jini Works

### 3.3.1) Registering A Service



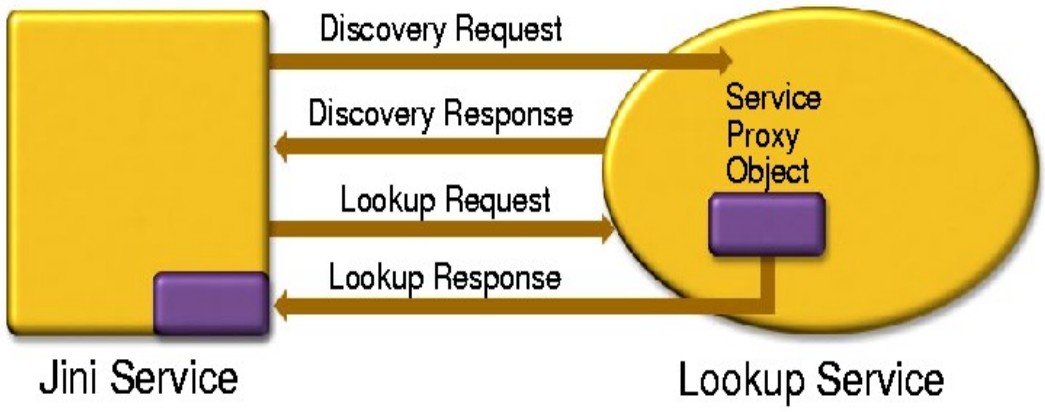
1. Service provider broadcasts a *presence announcement* by dropping a multicast packet onto a well-known port

2. Lookup services monitor the well-known port for presence announcement packets. It opens and inspects the packet when receives a presence announcement

3. Using RMI, the lookup service sends an object, called a *service registrar,* across the network to the originator of the packet

4. The service provider invokes the *register()* method on the service registrar object, passing as a parameter an object called a *service item,* a bundle of objects that describe the service. The *register()* method sends a copy of the service item up to the lookup service, where the service item is stored.

### 3.3.2) Finding A Service



1. A client invokes the lookup () method on a service registrar object.

2. The client passes as an argument to lookup () a *service template,* an object that serves as search criteria for the query. The service template can include a reference to an array of Class objects. These Class objects indicate to the lookup service the Java type (or types) of the service object desired by the client.

3. The lookup service performs the query and sends back zero to many matching service objects. The client gets a reference to the matching service objects as the return value of the lookup () method.

4. After receiving a service object now its time for client to use that service.

### 3.3.3) Using A Service

After receiving the service object from the server ,now its time for clien to use that service object to perform its tasks.Service object can be received by many ways.


Service object can grant clients access to the service in many ways. The object can actually represent the entire service, which is downloaded to the client during lookup and then executed locally. Alternatively, the service object can serve merely as a proxy to a remote server. When the client invokes methods on the service object, it sends the requests across

the network to the server, which does the real work. The local service object and a remote server can each do part of the work as well.

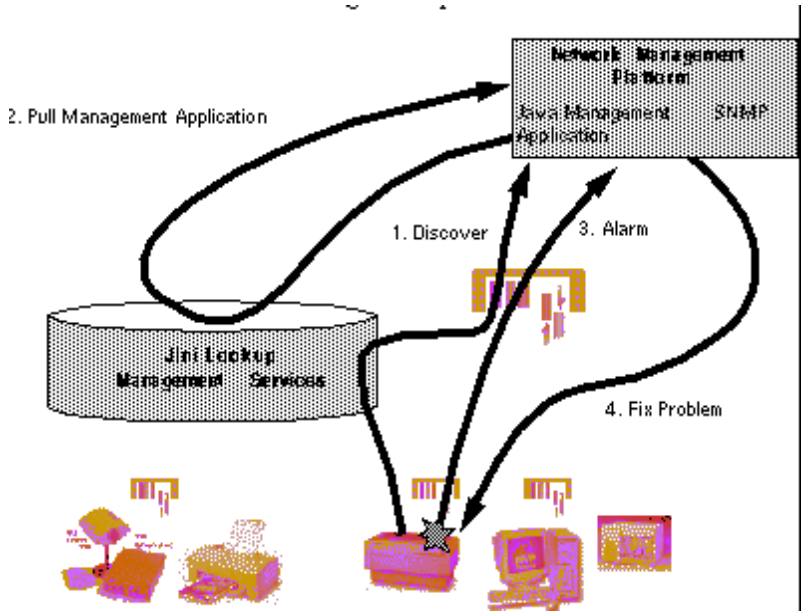
# CHAPTER 4

# Applications

## Applications of Jini Technology

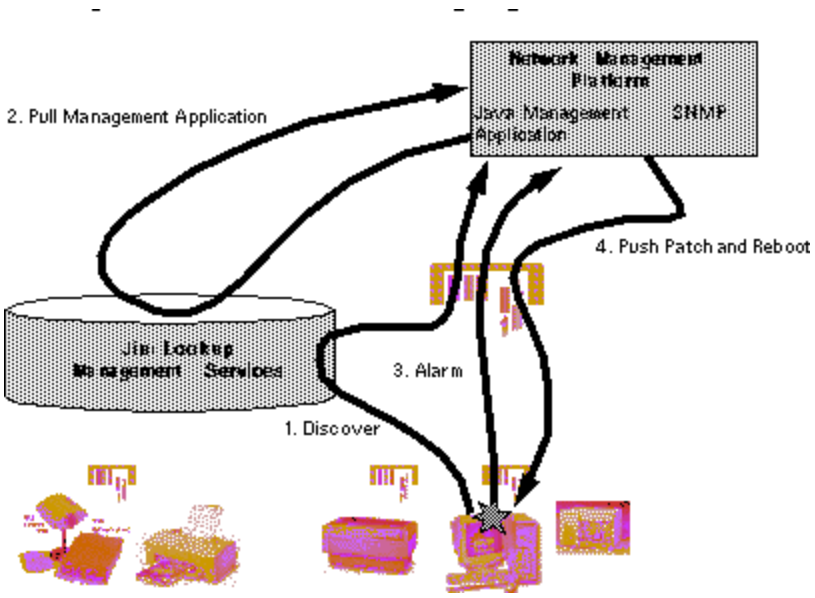
### 4.1) Managing a Printer

**Consider a Scenario**




The particular example shows how alarms are forwarded from the Jini technology-enabled resource to the network management platform, and what actions are taken. First, the printer is connected to the network. Through the Jini technology and the Java Dynamic Management kit, the printer is discovered and automatically appears on the map on the network management platform. Second, the system administrator, via a pop-up window on the network management platform, accesses the Jini technology-based lookup server directory to recover the management application from the printer. Finally, when a problem arises, and a corresponding alarm is generated. One example might be "job too big." When this occurs, the job is cancelled from the network management platform, via the pop-up printer management application, and the owner warned of the cancellation. In addition, statistics on printer usage.

## 4.2) **Managing a NT Server**




A second example often sited is managing an NT server. As shown in Figure 4, using the Jini technology-based lookup service, the system administrator discovers the NT server BIOS as the NT server is booted, and adds it to the map on the network management platform. When the inevitable server crash occurs, a corresponding alarm is generated to the network management platform. Then the system administrator again uses the lookup service directory, this time to recover the management application from the NT server. Finally, a patch is downloaded to the NT server, using the management application. The NT server is rebooted, using the management application. When the NT server is finally up and running, a cancellation alarm appears on the network management platform acknowledging things have returned to normal.

# CHAPTER 5

# Summerization

## Summarization

To sum up, Jini provides an extensive framework for developing flexible and robust distributed systems. It seems to fulfill the expectations related to network administration work reduction in a time, when the complexity of the distributed

systems keeps growing. It also deals with network failures better than the traditional object-based distribution solutions, because of its powerful and careful interface design.

A number of different distributed system frameworks exist as introduced in addition to the ones that where shortly mentioned in this thesis. The interoperability between these techniques is a key question, when considering the global development work in the future. It seems that Jini is quite adaptable to these other solutions because of its flexible nature, but there is a lot of competition going on between vendors like, for instance, Sun Microsystems and Microsoft Corporation. The worst scenario is that the techniques of these competing vendors try to drop each other out from the market instead of trying to complement each other. In this kind of competition, the best technology cannot always survive. More discussion about this rigid competition can be found in. Some research is already ongoing related to the complementing issues, and the results seem to be very promising , especially with Jini and SLP .In addition to the solutions for the challenges of distributed computing, Jini offers capabilities, which are required to fulfil the needs of modern end users of the network services. Flexibility and spontaneity are the most important issues, which are possible to deploy, if a Jini based system is designed in a sophisticated way. As the degree of distribution of the services grows all the time, security issues are also very important. In this area, the developers of Jini still have a lot of work to do, because Jini does not provide any security mechanisms in addition to the basic Java environment properties.

## 5.1) Drawbacks

At any rate, some drawbacks for Jini exist. One of the most restricting ones is that Jini is not compatible with the *Kilobyte Virtual Machine* (KVM) which is a very limited VM. It is targeted to embedded devices with approximately 128 kilobytes of available memory. There exists a lack of interoperability, because Jini requires some J2SE based features like RMI to work, and these kinds of features are too heavyweight for the current KVM implementations. The role of the KVM is to be a part of the new running environment of the larger developing solution J2ME for limited-resource mobile devices. This development environment is constructed of *configurations* and *profiles* (Figure 5-1). Configurations include a VM, optimized

APIs and core classes for a certain type of a device, such as a PDA or a mobile phone, which is not capable of running a full-blown JVM. Current basic configuration for the KVM is called *Connected, Limited Device Configuration*,which has been already approved by the *Java Developer Community* (JDC) as a *Java Specification Request* (JSR).

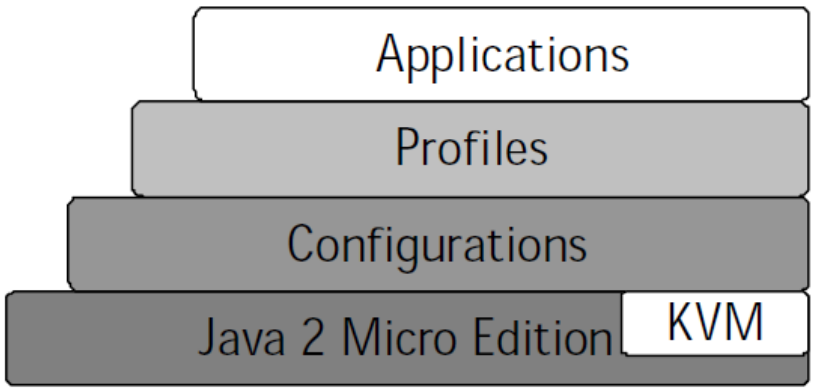



**Figure 5-1.** Configurations and profiles atop J2ME.

Profiles extend configurations by adding APIs that provide more capabilities for a specific market segment and a device type. The most interesting profile at the moment is the *Mobile Information Device Profile* (MIDP), which is targeted to mobile, wireless, bandwidth-limited, battery operated devices, as for instance, mobile phones. Jini support for these kinds of platforms, like J2ME, is still not available. This has reduced the success and adoption of the Jini technology, because wireless devices are the ones, which are now being integrated as a part of existing, IP based distributed systems.As a solution for capacity problems, there exists a new *surrogate* technology . The principle of this technology is that some other Jini networkconnected device, which has better capabilities to execute code, acts on behalf of the limited devices, which are connected to these *surrogate hosts* through interoperable

*adapters* using a *surrogate protocol*. The model has much equivalence with the third-party remote event model and the leasing model used in Jini. In addition, the surrogate technology seems to be the only solution at this moment for *Bluetooth* (BT) [Blu99] and *Wireless LAN.* enabled devices, which will soon exist on the mass market. It seems natural that BT and WLAN *access points* (APs) will work as surrogate hosts, since they can be static and have more processor power to run the full JRE in support of the wireless devices itself. The combination of these wireless technologies and Jini is definitely a worth of investigating in the future. The hardware and software requirements discussed in the thesis can already be met to some extend within the limited devices, but the market seems not to be ready for Jini products yet, at least not the mass market of cost-critical consumer products. Nevertheless, some attempts to break into the market have already been made, for instance , and the evelopment of the new concepts is ongoing in the laboratories of various companies. One general example for embedded devices is reviewed in. In addition to the relatively high computing capacity requirements, another drawback seems to be the security issues of Jini systems. The highly mobile nature of objects requires new security paradigms to be used to protect the data and the privacy of end users. As described by , traditional security models are class-based, but they should be object-based, because the objects are the ones that are transferred over the network. Even if the JVM with security policies offers the built-in security for Java objects, it does not seem to be enough for the Jini environment, because Jini expands Java's distribution and code downloading capabilities, but offers no extra security. Moreover, in Jini there is no default authentication mechanism in the lookup service specification, and furthermore, the discovery process is not protected, even if the use of services is controlled. Some solutions to security hazards in Jini systems are presented in, and presents a Jini capable architecture, which is based on a secure *Service Discovery Service* (SDS) with the use of *eXtensible Markup Language* (XML).

## 5.2) Future

In general, Jini specifications are relatively loose, which is an advantage,because it gives developers room for being creative and innovative. The specifications define mostly behavior related issues, which has to be taken care of when implementing the services. All

well-behaving Jini services should be capable of fulfilling the requirements defined by the specifications. These requirements have been discussed along with the discovery and join protocol issues.

It can also be a disadvantage to have so many system specific issues in an implementation process. Without well-guided general development paths, Jini based services can vary a lot in the future. Of course this is also desirable from the content point of view, but the difficulties are on the service interface side. In theory, an infinite number of possibly different services must be able to be implemented, but still all of them should be reachable by all potential clients through their known interfaces. Standardization process for these interfaces is an enormous task, which requires very careful designing. Jini's integration with the intelligent agent software technology is an interesting development path. As the Java programming language has become common, the portability problem of agent components has been solved for some extent. Also a VM based code execution provides better security solutions against hostile agents. Firstly, some guidelines are needed to model a Jini service as a software agent to enable even more intelligent and adaptable solutions.

Secondly, Jini does not really define a standard way for clients and services to communicate. However, to be interoperable with general software agents, the interaction of components must be clearer as it is now. One solution is to integrate Jini services to be able to communicate with agents through *Agent Communication Language* (ACL). Some ways to make this happen are presented. Because many agents are *autonomous*, which means that they do not need any human intervention to work, the Jini technology provides a good development framework for the mobile agent technologies because of spontaneity and a lightweight service discovery. The advantages mentioned here are obvious, when compared to the former work on this area.

We have a lot of electronic devices around to help us out scheduling meetings, locating restaurants and sending messages etc. These devices are usually focused on carrying out a particular task e.g. playing music or warming up the food. The realization of the concept of ubiquitous computing is only partly accomplished, because the technology is not invisible to the end users yet. The trend of the future seems to be that the end users are not required to command the devices to do some task, but the devices perform the task independently according to the current needs of the users. This is a goal, because the large number of

alternative services is impossible to handle manually, even at this moment. Some very interesting visions of this kind of technology behavior . Agent technologies have a ignificant role in the development of the ubiquitous computing. If considering Jini as a framework for agents, it is noticed that, in all probability, this way Jini has a role in ubiquitous computing too. As predicted data and services will be more important than location, which means that the end users must be able to connect to the network for the service in spite of the location of it and users. This, again, fits nicely to the Jini's idea of a service-centric network. This location independent approach has in recent years evolved fast, since the popularity and service development of mobile phones have increased. On the other hand, local improvements are still far away from the target of ubiquitous computing vision, because only the first steps has already been taken with BT and WLAN. It is clear at the moment that Jini is only suitable for a LAN environment, which has to be adaptable enough to handle clients, which temporarily connect to the LAN for some service and then leave the network. Furthermore, a Jini system is not scalable in the full meaning of the term, because it can only grow from a LAN to a WAN by using the ability to federate the lookup services. This will definitely chance in the future after the new versions of Jini are introduced as discussed along this thesis. At any rate, one problem related to WANs is that the service discovery by using the multicast messaging model is obviously not an option.

Despite of the drawbacks, Jini is evolving all the time. Very important thing for Jini technology has been Sun Microsystems' new model for developing technical concepts further [Wal00]. Jini is so called *open source technology*, which means that anybody can participate to the development process by e.g. reviewing the specifications, proving comments and suggestions. Sun Microsystems has succeeded well with this model, and it seem that they have been able to create a very active community of developers with more than 40000 members . All service developers use the *Sun Community Source License* (SCSL) model, which is a way to provide fast access to the technology and retain developers' intellectual properties related to Jini code, but it still provides some income to the original developers of Jini, when community members make any commercial products.

Even if Jini still does not fulfill all the wild ideas of the ability to connect every kind of devices, like toasters, refrigerators and wristwatches, to each other and to the Internet, it certainly offers more down-to-earth advantages to the distributed computing as seen. The

key features of Jini, e.g. simplicity and spontaneity, will be even more important in the future as the number of various systems, networks, clients, products and services are being developed.

## 5.3) References


1) www.**jini**.org

2) www.devx.com/assets/download/4508.pdf

3)Java.sun.com/developer/technical Articles/**jini**/.../Javatanks.html

4) en.**wikipedia**.org/wiki/**Jini**