Jini Network Technology PROJECT REPORT

MASTER OF COMPUTER APPLICATIONS BY AMIT KUMAR SINHA

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT
NIT CAMPUS P.O, KOZHIKODE 673601, KERALA

DEPARTMENT OF COMPUTER ENGINEERING

National Institute of Technology Calicut

Department of Computer Engineering CERTIFICATE This is to certify that the work reported in this project report entitled “Jini Newtwork Technology” is a bonafide re cord of the work done by AMIT KUMAR SINHA (Y2M001), a student in the Department of Computer Engineering, National Institute of Technology Calicut .

Co-ordinator:

Head of the Department:

Acknowledgement
I would like to put on records my sincere thanks to: Dr V.k. Govindan Head of Co mputer science and Engineering Department, my guide for this seminar, who helped me in preparing for this seminar. Mrs. Priya Chandran and Miss Nisha K.K, for a ll guidance I have received. My friends who’s wholehearted support helped me at al l stages of the preparation of this seminar.

ABSTRACT Jini is a network architecture for the construction of the distributed systems where scale, rate of change and the complexity of the interactions withi n and between networks are extremely important and cannot be satisfactorily addr essed by existing technology. This technology provides a flexible infrastructure for delivering services in a network and for creating spontaneous interactions between clients that use these services regardless of their hardware or software implementations. Concepts introduced by this technology such as leasing discove ry and the movement of objects over a network vastly simplifies interaction on a network embedding a great degree of resiliency and scalability within a distrib uted system. Companies are using this technology to build communities of service s out of simple piece of hardware and software that have been built according to it's specification. Examples ranges from business enterprises, medical systems, f inancial applications, and battlefield implementations, to a Jini enabled car or a Smart House. It's source code is available at no cost. There are no fees of any kind associated with the licensing and use of this technology. The Jini archite cture is the only technology in the marketplace that moves Objects around the ne twork and relegates wire protocols to mere implementation details. It's vision is "n etwork anything , anytime, anywhere !".Due to the simplicity of Jini architecture it clearly wins and will be.

1. Introduction:

Under the leadership of Bill Joy and Jim Waldo on January 25,1999 Sun Microsyste m launched Jini technology.”Jini isn’t an acronym though, it has been stated that it does function as an anti acronym “Jini Is Not Initials”. Over the last quarter cent ury, network technology has evolved rapidly and in some unexpected ways. Client/ server and multi-tier models operating within a single business enterprise have given way to an Internet/Web environment where services are provided by nodes sc attered over a far-flung network. Today, the next generation of network interact ion is emerging that will place unprecedented demands upon existing network tech nologies and architectures. For example, participants in one network will need t o directly access and use the services provided by participants in another netwo rk. It is in this network environment - one of mindnumbing complexity driven by geometric increases in scale, rate of change, and multiplicity of participant in teractions - that the simplicity of the Jini architecture clearly wins. Traditio nally, operating systems have been designed with the assumption that a computer will have a processor, some memory, and a disk. When we boot a computer, the fir st thing it does is look for a disk. If it doesn't find a disk, it can't functio n as a computer. Increasingly, however, computers are appearing in a different g uise: as embedded devices that have a processor, some memory, and a network conn ection -- but no disk. The first thing a cellphone does when we boot it up, for example, is look for the telephone network. If it doesn't find the network, it c an't function as a cellphone. This trend in the hardware environment, from disk- centric to network-centric, will affect how we organize our software -- and that 's where Jini comes in. Jini is an attempt to rethink computer architecture, giv en the rising importance of the network and the proliferation of processors in d evices that have no disk drive. These devices, which will come from many differe nt vendors, will need to interact over a network. The network itself will be ver y dynamic -- devices and services will be added and removed regularly. Jini prov ides mechanisms to enable smooth adding, removal, and findi ng of devices and se rvices on the

network. In addition, Jini provides a programming model that makes it easier for programmers to get their devices talking to each other. Building on top of Java, object serialization, and RMI, which enable objects to move around the network from virtual machine to virtual machine, Jini attempts to extend the benefits of object-oriented programming to the network. Instead of requiring device vendors to agree on the network protocols through which their devices can interact, Jini enables the devices to talk to each other through interfaces to objects.

1.1 What is Jini?

Jini is a set of APIs and network protocols that can help us 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.

Fig.1 Federation of Service available on the Jini network. The idea behind the word federation is based on the Jini view of the network -- there isn't a central controlling authority. Because no one service is in charge, the set of all services available on the network form a federation -- a group composed of equal peers. Instead of a central authority, Jini's runtime infrastructure merely provides a way for clients and services to find each other (via a lookup service, which

stores a directory of currently available services). After services locate each other, they are on their own. The client and its enlisted services perform their task independently of the Jini runtime infrastructure. If the Jini lookup service crashes, any distributed systems brought together via the lookup service before it crashed can continue their work. Jini even includes a network protocol that clients can use to find services in the absence of a lookup service.

1.2 Goals of the System

The overall goal is to turn the network into a flexible, easily administered tool with which human and computational clients can find resources. Resources can be implemented as either hardware devices, software programs, or a 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 workgroup by enabling the ability to add and delete services flexibly. A Jini system consists of the following parts:

• • •

A set of components that provides an infrastructure for federating services in a distributed system A programming model that supports and encourages the production of reliable distributed services Services that can be made part of a federated Jini system and that offer functionality to any other member of the federation

Although these pieces are separable and distinct, they are interrelated, which can blur the distinction in practice. The components that make up the Jini technology infrastructure make use of the Jini technology-programming model; services that reside within the infrastructure also use that model; and the programming model is well supported by components in the infrastructure. The end goals of the system span a number of different audiences; these goals include the following:

•

Enabling users to share services and resources over a network

•

•

Providing users easy access to resources anywhere on the network while allowing the network location of the user to change Simplifying the task of building, maintaining, and altering a network of devices, software, and users

The Jini system extends the Java application environment from a single virtual machine to a network of machines. The Java application environment provides a good computing platform for distributed computing because both code and data can move from machine to machine. The environment has built-in security that allows the confidence to run code downloaded from another machine. Strong typing in the Java application environment enables identifying the class of an object to be run on a virtual machine even when the object did not originate on that machine. The result is a system in which the network supports a fluid configuration of objects that can move from place to place as needed and can call any part of the network to perform operations. The Jini architecture exploits these characteristics of the Java application environment to simplify the construction of a distributed system. The Jini architecture adds mechanisms that allow fluidity of all components in a distributed system, extending the easy movement of objects to the entire networked system. The Jini technology infrastructure provides mechanisms for devices, services, and users to join and detach from a network. Joining and leaving a Jini system are easy and natural, often automatic, occurrences. Jini systems are far more dynamic than is currently possible in networked groups where configuring a network is a centralized function done by hand.

1.3 How Jini works 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 s ervices 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 co nsult a lookup service. The runtime infrastructure uses one network-level protoc ol, called discovery, and two object-level protocols, called join and lookup. Di scovery enables clients and services to locate lookup services. Join enables a s ervice 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.

1.4 Environmental Assumptions The Jini system federates computers and computing devices into what appears to the user as a single system. It relies on the exist ence of a network of reasonable speed connecting those computers and devices. So me devices require much higher bandwidth and others can do with much less displa ys and printers are examples of extreme points. We assume that the latency of th e network is reasonable. We assume that each Jini technology-enabled device has some memory and processing power. Devices without processing power or memory may be connected to a Jini system, but those devices are controlled by another piec e of hardware and/or software that presents the device to the Jini system and it self contains both processing power and memory. Architectures for devices not eq uipped with a Java virtual machine 1 (JVM) are explored more fully in the Jini D evice Architecture Specification. The Jini architecture gains much of its simpli city from assuming that the Java programming language is the implementation lang uage for components. The ability to dynamically download and run code is central to a number of the features of the Jini architecture. However, the Java technol ogy-centered nature of the Jini architecture depends on the Java application env ironment rather than on the Java programming language. Any programming language can be supported by a Jini system if it has a compiler that produces compliant b ytecodes for the Java programming language.

2 System Overview 2.1 Key Concepts The purpose of the Jini architecture is to federate groups of devices and software components into a single, dynamic distributed system. The resulting federation provides the simplicity of access, ease of administration, and support for sharing that are provided by a large monolithic system while retaining the flexibility, uniform response, and control provided by a personal computer or workstation. The architecture of a single Jini system is targeted to the workgroup. Members of the federation are assumed to agree on basic notions of trust, administration, identification, and policy. It is possible to federate Jini systems themselves for larger organizations. 2.1.1 Services The most important concept within the Jini architecture is that of a service as aforementioned. Members of a Jini system federate to share access to services. Jini systems provide mechanisms for service construction, lookup, communication, and use in a distributed system. Examples of services include: devices such as printers, displays, or disks; software such as applications or utilities; information such as databases and files; and users of the system. Services in a Jini system communicate with each other by using a service protocol, which is a set of interfaces written in the Java programming language. The set of such protocols is open ended. The base Jini system defines a small number of such protocols that define critical service interactions.

2.1.2 Java Remote Method Invocation (RMI) Communication between services can be accomplished using Java Remote Method Invocation (RMI). The infrastructure to support communication between services is not itself a service that is discovered and used but is, rather, a part of the Jini

technology infrastructure. RMI provides mechanisms to find, activate, and garbag e collect object groups. Fundamentally, RMI is a Java programming language-enabl ed extension to traditional remote procedure call mechanisms. RMI allows not onl y data to be passed from object to object around the network but full objects, i ncluding code. Much of the simplicity of the Jini system is enabled by this abil ity to move code around the network in a form that is encapsulated as an object. 2.1.3 Security The design of the security model for Jini technology is built on the twin notions of a principal and an access control list. Jini services are a ccessed on behalf of some entity the principal which generally traces back to a particular user of the system. Services themselves may request access to other s ervices based on the identity of the object that implements the service. Whether access to a service is allowed depends on the contents of an access control lis t that is associated with the object. 2.1.4 Leasing Access to many of the servic es in the Jini system environment is lease based. A lease is a grant of guarante ed access over a time period. Each lease is negotiated between the user of the s ervice and the provider of the service as part of the service protocol: A servic e is requested for some period; access is granted for some period, presumably ta king the request period into account. If a lease is not renewed before it is fre ed--either because the resource is no longer needed, the client or network fails , or the lease is not permitted to be renewed--then both the user and the provid er of the resource may conclude that the resource can be freed. Leases are eithe r exclusive or non-exclusive. Exclusive leases ensure that no one else may take a lease on the resource during the period of the lease; non-exclusive leases all ow multiple users to share a resource.

2.1.5 Transactions A series of operations, either within a single service or spanning multiple services, can be wrapped in a transaction. The Jini transaction interfaces supply a service protocol needed to coordinate a two-phase commit. How transactions are implemented--and indeed, the very semantics of the notion of a transaction--is left up to the service using those interfaces. 2.1.6 Events The Jini architecture supports distributed events. An object may allow other objects to register interest in events in the object and receive a notification of the occurrence of such an event. This enables distributed event -based programs to be written with a variety of reliability and scalability guarantees. 2.2. Component Overview The components of the Jini system can be segmented into three categories: infrastructure, programming model, and services. The infrastructure is the set of components that enables building a federated Jini system, while the services are the entities within the federation. The programming model is 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. These three categories, though distinct and separable, are entangled to such an extent that the distinction between them can seem blurred. Moreover, it is possible to build systems that have some of the functionality of the Jini system with variants on the categories or without all three of them. But a Jini system gains its full power because it is a system built with the particular infrastructure and programming models described, based on the notion of a service. Decoupling the segments within the architecture allows legacy code to be changed minimally to take part in a Jini system. Nevertheless, the full power of a Jini system will be available only to new services that are constructed using the integrated model. A Jini system can be seen as a network extension of the infrastructure, programming model, and services that made Java technology successful in the single-machine case. These

categories along with the corresponding components in the familiar Java applicat ion environment are shown in Figure 2.1:

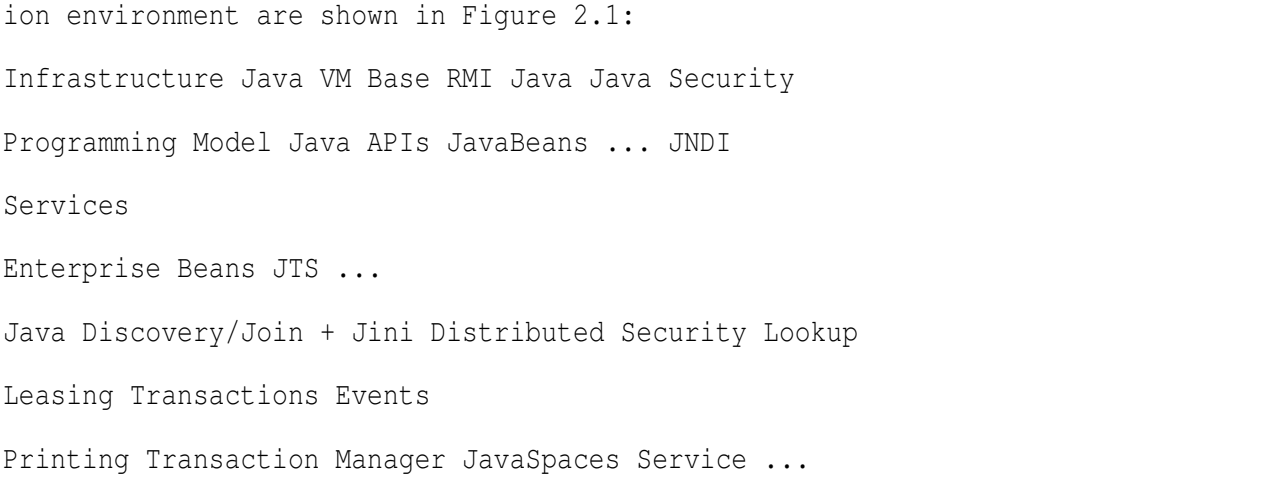

Figure 2.1: Jini Architecture Segmentation 2.2.1 Infrastructure The Jini technol ogy infrastructure defines the minimal Jini technology core. The infrastructure includes the following:

•

•

•

A distributed security system, integrated into RMI, that extends the Java platfo rm's security model to the world of distributed systems. The discovery and join protocols, service protocols that allow services (both hardware and software) to discover, become part of, and advertise supplied services to the other members of the federation. The lookup service, which serves as a repository of services. Entries in the lookup service are objects written in the Java programming langu age; these objects can be downloaded as part of a lookup operation and act as lo cal

proxies to the service that placed the code into the lookup service. The discove ry and join protocols define the way a service of any kind becomes part of a Jin i system; RMI defines the base language within which the Jini technology-enabled services communicate; the distributed security model and its implementation def ine how entities are identified and how they get the rights to perform actions o n their own behalf and on the behalf of others; and the lookup service reflects the current members of the federation and acts as the central marketplace for of fering and finding services by members of the federation. 2.2.2 Programming Mode l The infrastructure both enables the programming model and makes use of it. Ent ries in the lookup service are leased, allowing the lookup service to reflect ac curately the set of currently available services. When services join or leave a lookup service, events are signaled, and objects that have registered interest i n such events get notifications when new services become available or old servic es cease to be active. The programming model rests on the ability to move code, which is supported by the base infrastructure. Both the infrastructure and the s ervices that use that infrastructure are computational entities that exist in th e physical environment of the Jini system. However, services also constitute a s et of interfaces, which define communication protocols that can be used by the s ervices and the infrastructure to communicate between themselves. These interfac es, taken together, make up the distributed extension of the standard Java progr amming language model that constitutes the Jini programming model. Among the int erfaces that make up the Jini programming model are the following:

•

•

The leasing interface, which defines a way of allocating and freeing resources u sing a renewable, duration-based model The event and notification interfaces, wh ich are an extension of the event model used by JavaBeans components to the dist ributed environment, enable event -

.

based communication between Jini technology-enabled services The transaction int erfaces, which enable entities to cooperate in such a way that either all of the changes made to the group occur atomically or none of them occur

The transaction interfaces introduce a lightweight, objectoriented protocol enab ling applications using Jini technology to coordinate state changes. The transac tion protocol provides two steps to coordinate the actions of a group of distrib uted objects. The first step is called the voting phase, in which each object "v otes" whether it has completed its portion of the task and is ready to commit an y changes it made. In the second step, a coordinator issues a "commit" request t o each object. The Jini transaction protocol differs from most transaction inter faces in that it does not assume that the transactions occur in a transaction pr ocessing system. Such systems define mechanisms and programming requirements tha t guarantee the correct implementation of a particular transaction semantics. Th e Jini transaction protocol takes a more traditional object-oriented view, leavi ng the correct implementation of the desired transaction semantics up to the imp lementer of the particular objects that are involved in the transaction. The goa l of the transaction protocol is to define the interactions that such objects mu st have to coordinate such groups of operations. The interfaces that define the Jini programming model are used by the infrastructure components where appropria te and by the initial Jini technology-enabled services. For example, the lookup service makes use of the leasing and event interfaces. Leasing ensures that serv ices registered continue to be available, and events help administrators discove r problems and devices that need configuration. The JavaSpaces service, one exam ple of a Jini technology-enabled service, utilizes leasing and events, and also supports the Jini transaction protocol. The transaction manager can be used to c oordinate the voting phase of a transaction for those objects that support trans action protocol. The implementation of a service is not required to use the Jini programming model, but such services need to use that model for their interacti on with the Jini technology infrastructure. For example, every service interacts with the Jini lookup service by using the programming model; and whether a serv ice offers

resources on a leased basis or not, the service's registration with the lookup s ervice will be leased and will need to be periodically renewed. The binding of t he programming model to the services and the infrastructure is what makes such a federation a Jini system not just a collection of services and protocols. The c ombination of infrastructure, service, and programming model, all designed to wo rk together and constructed by using each other, simplifies the overall system a nd unifies it in a way that makes it easier to understand.

2.3 Service Architecture: Service from the interactive basis for a Jini system, both at the programming and user interface levels. The details of the service ar chitecture are best understood once the Jini lookup protocols are presented.

2.3.1 the discovery process
Discovery works like this: Imagine We have a Jini-enabled disk drive that offers a persistent storage service. As soon as We connect the drive to the network, i t 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 numb er where the disk drive can be contacted by a lookup service. Lookup services mo nitor the well-known port for presence announcement packets. When a lookup servi ce receives a presence announcement, it opens and inspects the packet. The packe t contains information that enables the lookup service to determine whether or n ot it should contact the sender of the packet. If so, it contacts the sender dir ectly by making a TCP connection to the IP address and port number extracted fro m the packet. Using RMI, the lookup service sends an object, called a service re gistrar, 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.

Fig.2.2 The Discovery Process

By invoking methods on this object, the sender of the announcement packet can pe rform 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.

2.3.2 The join process
Once a service provider has a service registrar object, the end product of disco very, it is ready to do a join -- to become part of the federation of services t hat are registered in the lookup service.

Fig.2.3 The Join Process 13

To do a join, 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. Once this has completed, the service provider has finished the join process: its service has become registered in the lookup service. The service item is a container for several objects, including an object called a service object, which clients can use to interact with the service. The service item can also include any number of attributes, which can be any object. Some potential attributes are icons, classes that provide GUIs for the service, and objects that give more information about the service. Service objects usually implement one or more interfaces through which clients interact with the service. For example, a lookup service is a Jini service, and its service object is the service registrar. The register() method invoked by service providers during join is declared in the ServiceRegistrar interface, which all service registrar objects implement. Clients and service providers talk to the lookup service through the service registrar object by invoking methods declared in the ServiceRegistrar interface. Likewise, a disk drive would provide a service object that implemented some well-known storage service interface. Clients would look up and interact with the disk drive by this storage service interface.

2.3.3 The lookup process

Once a service has registered with a lookup service via the join process, that service is available for use by clients who query that lookup service. To build a distributed system of services that will work together to perform some task, a client must locate and enlist the help of the individual services. To find a service, clients query lookup services via a process called lookup. To perform a lookup, a client invokes the lookup() method on a service registrar object. (A client, like a service provider, gets a service registrar through the process of discovery, as described earlier in this article.) 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. 14

These Class objects indicate to the lookup service the Java type (or types) of t
he service object desired by the client. The service template can also include a
 service ID, which uniquely identifies a service, and attributes, which must exa
ctly match the attributes uploaded by the service provider in the service item.
The service template can also contain wildcards for any of these fields. A wildc
ard in the service ID field, for example, will match any service ID. The lookup(
) method sends the service template to the lookup service, which performs the qu
ery and sends back zero to many matching service objects. The client gets a refe
rence to the matching service objects as the return value of the lookup() method
. In the general case, a client looks up a service by Java type, usually an inte
rface. For example, if a client needed to use a printer, it would compose a serv
ice template that included a Class object for a wellknown interface to printer s
ervices. All printer services would implement this well-known interface. The loo
kup service would return a service object (or objects) that implemented this int
erface. Attributes can be included in the service template to narrow the number
of matches for such a type-based search. The client would use the printer servic
e by invoking on the service object methods declared in the well-known printer s
ervice interface.

3. Separation of interface and implementation
Jini's architecture brings object-oriented programming to the network by enablin
g network services to take advantage of one of the fundamentals of object-orient
ed programming: the separation of interface and implementation. For example, a s
ervice object can grant clients access to the service in many ways. The object c
an actually represent the entire service, which is downloaded to the client duri
ng 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 se
rvice 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. One important consequence of Jini's architecture is that the
network protocol used to communicate between a proxy service object and a remote
 server does not need to be known to the client. As illustrated in the figure be
low, the network protocol is part of the service's implementation. This protocol
 is a private matter decided upon by the developer of the service. The client ca
n communicate with the service

via this private protocol because the service injects some of its own code (the service object) into the client's address space. The injected service object could communicate with the service via RMI, CORBA, DCOM, some home-brewed protocol built on top of sockets and streams, or anything else. The client simply doesn't need to care about network protocols, because it can talk to the well-known interface that the service object implements. The service object takes care of any necessary communication on the network.

The client talks to the service through a well-known interface

Fig 3.Interface Implementation Different implementations of the same service interface can use completely different implementation approaches and completely different network protocols. A service can use specialized hardware to fulfill client requests, or it can do all its work in software. In fact, the implementation approach taken by a single service can evolve over time. The client can be sure it has a service object that understands the current implementation of the service, because the client receives the service object (by way of the lookup service) from the service provider itself. To the client, a service looks like the well-known interface, regardless of how the service is implemented.

4. Performance Analysis:
16 Sun Microsystems provides a development platform for E-business suit over 5,0 00 out of 13,000+ Companies worldwide, including Sun Corporate IT.Jini is one of the market leader among Solaris OE Servers,Storage,Java,J2EE,JME.Some of the fa cts and achievements associated with Jini are mentioned below :

Company 4D Networks, Inc. eko systems, Inc.

Industry

Product/ Category

Abstract

Telecom/Service Provider
Healthcare/Insuranc e

Java and Technologies
Operating Systems, Java and Technologies

Based on Jini network technology, 4D Pocket System uses the Jini lookup service to register services to its directory. Jini technology provided the system, call ed Frontiers, with the ability to connect to any type of medical equipment–inclu ding physiologic monitors, ventilators, and anesthesia machineswithout requiring clinicians to configure data or software drivers. Jini, a new paradigm for the development and management of distributed systems, provides mechanisms that enab le systems to plug together to form an impromptu community. This practice report demonstrates how Jini can be applied in an industrial environment, or more exac tly, how it can be used in the integration of embedded devices on-board trains i n the backoffice IT infrastructure of railway operators. The results are encoura ging and prove that Jini is the appropriate technology to link application serve rs and service gateways in embedded servers. This paper describes the architectu re of a Jini based global computing environment, whose aim is to support the exe cution of sequential and parallel programs on a large scale. We describe the arc hitecture, supported programming models of the system, as well as our current, o ngoing work and future development plans.

Txomin Nieva, Swiss Federal Institute of Technology and ABB Corporate Research L td. Andreas Fabri, ABB Corporate Research Ltd. Abdenbi Benammour , Swiss Federal Institute of Technology

IEEE

Jini Technology Applied to Railway Systems

Zoltan Juhasz, Arpad Andics, Szabolcs Pota

2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'0 2) May 21 – 24, 2002 Berlin, Germany

JM: A Jini Framework for Global Computing

17

6. Conclusion

As we have seen in this report, Jini attempts to raise the level of abstraction for distributed systems programming, from the network protocol level to the object interface level. In the emerging proliferation of embedded devices connected to networks, many pieces of a distributed system may come from different vendors. Jini makes it unnecessary for vendors of devices to agree on network level protocols that allow their devices to interact. Instead, vendors must agree on Java interfaces through which their devices can interact. The processes of discovery, join, and lookup, provided by the Jini runtime infrastructure, will enable devices to locate each other on the network. Once they locate each other, devices will be able to communicate with each other through Java interfaces.

7. References:

http://www.jini.org http://www.sun.com/jini/licensing. http:// wwws.sun.com/software/jini/specs/
http://www.java.sun.com/developer/products/jini/index.jsp http://csdl.computer.org/comp/proceedings/doa/2000/0819/00/08190251abs.htm