The Jini architecture

Johan Petrini and Henning Sundvall

Distributed Systems Fall 2002
Abstract

A technology has been developed that exemplifies a new approach to the architecture of computing systems. The notion of peripherals and applications has been replaced with that of network-available services and clients thus making the network the central part of the system. Jini network technology is middleware, developed by Sun Microsystems, that provides a set of application programming interfaces (APIs) as well as network protocols. Jini basically enable anything with a processor, some memory and a network connection in offering services to entities on the network or using services provided by other entities on the network. A couple of protocols are used when you connect a device (software or hardware) to a Jini network and start using its services. The important ones are the discovery protocol and the lookup service protocol. The discovery protocol allows entities to join a Jini network and to find lookup services. The lookup service is a place where services advertise themselves for potential users. Jini is an example of a coordination based communication model. The main characteristic of such a model is the separation between computation and coordination. An example of where Jini connection technology is used in real life is the company Eko Systems (http://www.ekosystems.com). Eko Systems uses Jini in connecting medical equipment such as monitors and ventilators to its information system.
# Table of contents

Table of contents .............................................................................................................. 2

1 Introduction ................................................................................................................... 3

2 An overview of Jini ........................................................................................................ 3

3 Jini - a distributed coordination-based model .............................................................. 5

4 The Jini architecture ..................................................................................................... 5

  4.1 Discovery and Lookup service .................................................................................. 5

  4.2 Discovery protocols .................................................................................................. 6

  4.3 Leasing .................................................................................................................... 7

  4.4 Transactions .......................................................................................................... 7

5 Conclusion .................................................................................................................... 8

References ....................................................................................................................... 9

Appendix
1 Introduction

Services on the Internet have come a long way since basic communication between two homogenous computers not far from each other. Today, we can observe intelligent devices interacting with each other anytime and anywhere in the world. One major event that has contributed to this is the recent advances in network technology bringing forward alternative ways of communication between devices e.g Bluetooth. Another is the decreasing processor costs and size facilitating for engineers to develop devices with application specific processor power. Such progress has also opened up the field for new applications [1].

A popular scenario often referred to in media is the intelligent home. When you buy a new piece of office computing equipment such as a desk lamp, or a new home computer appliance such as an alarm clock, it will not only carry out its “traditional” functions but will also join into a network of other computer devices and services. The desk lamp will turn itself off when you leave your desk, informed by sensors in your chair; the alarm clock will tell your coffee maker to switch on a few minutes before it wakes you up [4].

In comparison with an enterprise network, a home network can characterised by a couple of features. For example, a home network must handle interference from household appliances such as microwave ovens and cordless phones. Also, the fact that a home network lacks a system administrator implies a great deal of self administration. The network must operate with little or no user intervention. Further, a home network is often very heterogenous with variances in structure and usage among in-home devices. Their purpose range from washing clothes to controlling room humidity [4].

Of course, this discussion can be generalized to most networks, not only local area ones. By connecting the home network to the Internet electrical devices are made accessible from all over the world. In order to bring some order to the chaos there is clearly need for a technology enabling and coordinating the communication between the devices in a, for the user, transparent way.

2 An overview of Jini

Jini network technology is middleware, developed by Sun Microsystems, that provides a set of application programming interfaces (APIs) as well as network protocols. Jini assumes a network environment where the components and the way they interact constantly changes and is designed to support the incremental upgrading of network components [1]. For example, a Jini-compliant digital camera connects directly to an IP-based network. Other nodes on the network will detect the camera and can download the drivers required directly from the camera. This ensures that the driver matches the target hardware. Another scenario to consider is adding more storage to your network. Today, that probably means adding a hard drive to your server. However, with Jini you'll simply plug a storage device in to the network. Everyone on the network will immediately see the storage device, load the appropriate drivers, and start saving and loading data to their hearts' content. This same process works for printers or other devices. Jini basically
enable anything with a processor, some memory and a network connection in offering services to entities on the network or using services provided by other entities on the network [4].

Jini’s lead architect Jim Waldo claims that Jini technology is not to be looked upon as a distributed system. It is simply a system defining a small, simple set of conventions allowing services and clients to form a flexible distributed system tolerant to changes in the environment. The best way to illustrate Jini as a system is to separate it’s various components into:

1. The infrastructure.
2. A programming model.
3. The clients and services.

The Java Remote Method Invocation system constitutes the foundation of the Jini infrastructure. This basic architecture for remote programming is extended with two components, the discovery protocol, which makes it possible for an entity to join the network, and a lookup service assisting services to advertise themselves network wide [1].

The next part of the Jini infrastructure is the Jini programming model. It can be decomposed into three sets of interfaces. There is one set of interfaces that extends the standard Java event model in Java Beans and enables it to function in a distributed way. Another set of interfaces implements a simplified distribution version of the transaction model in the Java transaction service using a two-phase commit protocol [1].

Finally there are the services and the users of the services i.e. the clients. The environment in which these entities are operating is highly flexible. The services offered to clients depend totally on the current configuration of a specific Jini federation (informal group of clients and services complying with the same interaction patterns as defined by Jini). Examples of services in a network can be hardware implementations of Jini interfaces, software services in the form of distributed components and various combinations of them. A more comprehensive description of these three components is given later in the paper [1].

Each of the components is logically independent and can be seen as an extension of the Java language system. The functionality of Jini depends heavily upon the existing Java environment due to the fact that it requires features available only within the Java platform. One of the key aspects of Java is the ability to move code into a client that wants to use a particular service in an easy and safe way. With the Java virtual machine (JVM) offering a consistent environment in which Jini can exist the fact the fact that the underlying machines can vary considerably in structure and resources is of less importance. The possibility to first compile the code into object code and then move it to another machine enables the code to be loaded into a running process and therefore allowing new functions in a running program. Of course, another important part of Java
heavily drawn upon by Jini is its inherent safety e.g. referential integrity and type safety in conjunction with Java’s own security model [1].

3 Jini - a distributed coordination-based model

Often in the literature Jini is referred to as a coordination based model. The main characteristic of such a model is the separation between computation and coordination. A distributed system can be described as a collection of processes each doing its computations independently from the activity of any other process. In a coordination-based system focus lies on handling the coordination and communication between these processes instead of the actual computations of processes. There are different types of coordination-based systems and they can be distinguished from one another according to two different dimensions, temporal and referential [5].

The most widely known coordination model is a combination of referentially and uncoupled processes called generative communication. The main idea in generative communication is that a collection of independent processes makes use of a shared persistent database of tuples. A tuple is a data structure similar to a data record with a number of typed fields. It is up to the process to put a record of any type in the database. The only real identifier of the tuple is a tag indicating the kind of information it’s carrying. Hence, a process does not have to agree on the structure of a tuple in advance. Such generative communication can be found in Jini where it is provided by a coordination system called JavaSpaces. A JavaSpace is a shared dataspace that stores tuples representing typed set of references to Java objects. It’s a part of Jini in the form of a user defined service and contributes to alternative ways of communication between entities in a Jini federation. For example, a client can request to be notified when a specific tuple instance is written to JavaSpace be calling the latter’s notify operation. However, when a more detailed description of JavaSpaces is outside the scope of this paper we confine ourselves with explaining the meaning of the important notion of a distributed coordination-based model [5][6].

4 The Jini architecture

Here we provide the reader with a more detailed description of the Jini architecture. Core Jini concepts like discovery and lookup service are explained together with properties like leasing and transactions.

4.1 Discovery and Lookup service

A couple of protocols are used when you connect a device (software or hardware) to a Jini network and start using its services. The important ones are the discovery protocol and the lookup service protocol. The discovery protocol allows entities to join a Jini network and to find lookup services. The lookup service is a place where services advertise themselves for potential users [2].

5
The service that wants to join the Jini system multicasts a request containing its address and port number on which the lookup service should respond to (using the multicast request protocol) on the LAN wanting a lookup service to reply. When a lookup service receives such a request it makes a TCP connection to that address and port. The service now sends a unicast discovery request. The lookup service now uses the unicast protocol to respond with a proxy of itself. A service can now register itself by loading a proxy object of itself into the lookup service. This is done by the unicast discovery protocol. If a client that is joining the Jini system wants to find services it has to follow the same first step as the service wanting to register itself to lookup service, multicasting a discovery request receiving a proxy object of the lookup service. If the client finds a service it wants to use it sends a message through the unicast discovery protocol to obtain a proxy of that particular service. This is done by requesting a specific Java programming interface of the service wanted. When a service is found the client downloads a copy of the proxy of the service. The client can now use the proxy object to communicate directly to the service provider [3].

When entities start they should always wait for a random amount of time. This is done to avoid a packet storm that could occur if a network has been down and is coming up. When an entity has registered a service to a lookup service it has to periodically renew its registration. A lookup service can only cancel leases of services if the service requests such a thing [3].

4.2 Discovery protocols

Any one entity that wants to use a Jini system has to get a proxy to some lookup service. The discovery protocols are the protocols that handle this.

There are three discovery protocols:

1. The multicast request protocol.
2. The multicast announcement protocol.
3. The unicast discovery protocol.

The multicast request protocol is used when an entity is connected to the Jini system and wants to find lookup services on the LAN. An entity using the multicast request protocol to discover lookup services establishes a multicast request client that will send packets to the well-known multicast network port, and a multicast response server that listens for connection response from the lookup service. These two components always exist together. At the lookup service side resides a multicast request server that listens for incoming multicast requests to the well-known multicast port, and a multicast response client that responds [2].

The multicast announcement protocol is used when a lookup service wants to announce its presence on the LAN. A system that runs a lookup service establish an multicast
announcement client that sends datagram packets to a well-known multicast port on
which the multicast announcement service operates, and a server side of the unicast
discovery service. The announcements are often sent at intervals of 120 seconds. An
entity that listens for multicast announcements establishes a multicast announcement
server that listens for incoming multicast announcements. If the entity receives a
multicast announcement to its server it uses the unicast discovery protocol to get the
proxy object of the announced lookup service if it doesn’t have a proxy of that lookup
service [2].

The unicast discovery protocol is used when an entity wants to communicate with a
specific lookup service or a lookup service that is not located on the LAN [2].

4.3 Leasing

The concept of leasing is part of the Jini programming model. It basically means that an
object can gain access to some resources during some agreed upon amount of time. A
time period can be negotiated between a lease grantor and a lease holder or just given to
the lease holder by the lease grantor. Leases are time based. In Jini this means that the
time is based on duration, not absolute time. This is because of the difficulty of
synchronizing two entities in a distributed system. When a lease is about to expire it can
be renewed by the lease holder if the lease grantor accepts such a request. If the lease
grantor denies a renewal of the lease any resources associated with the lease are freed.
Further more, the lease grantor can at any time simply cancel the lease of a specific lease
holder and clean up resources used by the lease. When a lease expires or is cancelled,
both the holder of the lease and the grantor of the lease know that the service or resource
has been reclaimed. Using leases in a distributed system as Jini gives certain advantages
compared to traditional distributed systems. Leasing in Jini enables the system to cope
with system malfunctions due to the fact that resources are freed when a lease is expired
or cancelled. For example, if system A encounters a breakdown the number of possible
orphans that can occur in the system is dramatically reduced due to the properties of
leasing [2].

4.4 Transactions

As opposed to traditional transaction systems concentrating on that everyone
participating in the transaction provides the correct implementation of transactional
semantics, Jini handles transactions in a different way. Jini leaves the implementation of
the transactional semantics to the objects participating in the transaction. The system
offers a minimal set of protocols and interfaces that allows objects to implement
transactional semantics, hence separating the protocols from the semantics of a certain
transaction. One of the protocols, the two-phase commit protocol, outlines the
communication patterns between distributed object and resources and allows them to
wrap a set of operations into what appears to be a single one. To provide the protocol
with consistent resolution of operations, meaning that all participants are guaranteed to
know whether to commit an operation or to abort, some kind of manager is also required.
Any object implementing the proper interface is a possible participant in a transaction [2].
5 Conclusion

With the trends moving towards the vision of pervasive computing there is clearly a need for some kind of framework to address the problems caused by an inherent heterogeneous computing environment. Such framework must enable and coordinate the communication between computing entities let it be sophisticated workstations or embedded CPUs. Jini offers a small set of rules defining how different devices should communicate in a Java based environment. Jini’s strength comes from its object oriented roots with transportation of code and data together to perform tasks. The architecture provides a simple mechanism for devices to form instant network communities without planning, installation, or human intervention.

A major drawback within a Jini network is that each device must run a Java Virtual Machine (JVM). To run a JVM there is a need for substantial computing resources only found in stationary devices such as workstations or PC’s. So called surrogate hosts has been developed to take care of the problem but they are not very helpful in ad hoc networks and peer to peer communication with no fixed infrastructure. Another drawback with Jini lies in the scalability problem of a system based on generative communication. Using multicasting to either insert a tuple into JavaSpace or remove a tuple from JavaSpace would lead to serious networking problems, such as congested networks, if implemented in wide area networks.

There are not yet so many Jini enabled products but companies use Jini connection technology for application specific needs. For example, the company Eko Systems (http:www.ekosystems.com) uses Jini in connecting medical equipment such as monitors and ventilators to its information system. Recently, researchers at Rochester Institute of technology have developed a mobile edition of Jini called JiniME addressing the problems with limited size, computing power and memory in many wireless mobile devices. Further, JiniME provides independence from a fixed infrastructure. One of the more noticed application areas for Jini is the home network were the Jini surrogate architecture provides a strong platform and supports both Java and Bluetooth.

Finally, one of the most important characteristics about Jini is the prerequisite of network-centric computing. The network takes on responsibilities former performed by the disk thus loosens the coupling between the code and the processor. In long-lived networks, the code to be run on a particular processor could be written before the processor even was designed. Processor independence makes it impossible to optimize the code for a processor but makes it easier to introduce new services in a network.
References


Appendix

An Example

This example shows how a Jini technology-enabled printing service might be used by a digital camera to print a high-resolution color image. It will start with the printer joining an existing Jini system, continue with its being configured, and end with printing the image [3].

1. Registering the Printer Service

A printer that is either freshly connected to a Jini system or is powered up once it has been connected to a Jini system grouping needs to discover the appropriate lookup service and register with it. This is the discovery and join phase.

2. Discovering the Lookup Service

The basic operations of discovering the lookup service are implemented by a Jini technology infrastructure software class. An instance of this class acts as a mediator between devices and services on one hand and the lookup service on the other. In this example the printer first registers itself with a local instance of this class. This instance then multicasts a request on the local network for any lookup services to identify themselves. The instance listens for replies and, if there are any, passes to the printer an array of objects that are proxies for the discovered lookup services.

3. Joining the Lookup Service

To register itself with the lookup service, the printer needs first to create a service object of the correct type for printing services. This object provides the methods that users and applications will invoke to print documents. Also needed is an array of LookupEntry instances to specify the attributes that describe the printer, such as that it can print in color or black and white, what document formats it can print, possible paper sizes, and printing resolution.

The printer then calls the register method of the lookup service object that it received during the discovery phase, passing it the printer service object and the array of attributes. The printing service is now registered with the lookup service.

4. Optional Configuration

At this point the printing service can be used, but the local system administrator might want to add additional information about the printer in the form of additional attributes, such as a local name for the service, information about its physical location, and a list of who may access the service. The system administrator might also want to register with the device to receive notifications for any errors that arise, such as when the printer is out of paper.
One way the system administrator could do this would be to use a special utility program to pass this additional information to the service. In fact this program might have received notification from the lookup service that a new service was being added and then alerted the system administrator.

5. Staying Alive

When the printer registers with the Jini lookup service it receives a lease. Periodically, the printer will need to renew this lease with the lookup service. If the printer fails to renew the lease, then when the lease expires, the lookup service will remove the entry for it, and the printer service will no longer be available.

6. Printing

Some services provide a user interface for interaction with them; others rely on an application to mediate such interaction. This example assumes that a person has a digital camera that has taken a picture they want to print on a high-resolution printer. The first thing that the camera needs to do after it is connected to the network is locate a Jini technology-enabled printing service. Once a printing service has been located and selected, the camera can invoke methods to print the image.

7. Locate the Lookup Service

Before the camera can use a Jini technology-enabled service, it must first locate the Jini lookup service, just as the print service needed to do to register itself. The camera registers itself with a local instance of the Jini technology infrastructure class LookupDiscovery, which will notify the camera of all discovered lookup services.

8. Search for Printing Services

Finding an appropriate service requires passing a template that is used to match and filter the set of existing services. The template specifies both the type of the required service, which is the first filter on possible services, and a set of attributes which is used to reduce the number of matching services if there are several of the right type. In this example, the camera supplies a template specifying the printer type and an array of attribute objects. The type of each object specifies the attribute type, and its fields specify values to be matched. For each attribute, fields that should be matched, such as color printing, are filled in; ones that don't matter are left null. The Jini lookup service is passed this template and returns an array of all of the printing services that match it. If there are several matching services, the camera may further filter them--in this case perhaps to ensure high print resolution--and present the user with the list of possible printers for choice. The final result is a single service object for the printing service.

At this point the printing service has been selected, and the camera and the printer service communicate directly with each other; the lookup service is no longer involved.
9. Configuring the Printer

Before printing the image, the user might wish to configure the printer. This might be done directly by the camera invoking the service object's `configure` method; this method may display a dialog box on the camera's display with which the user may specify printer settings. When the image is printed, the service object sends the configuration information to the printer service.

10. Requesting That the Image Be Printed

To print the image, the camera calls the print method of the service object, passing it the image as an argument. The service object performs any necessary preprocessing and sends the image to the printer service to be printed.

11. Registering for Notification

If the user wishes to be notified when the image has been printed, the camera needs to register itself with the printer service using the service object. The camera might also wish to register to be notified if the printer encounters any errors.

12. Receiving Notification

When the printer has finished printing the image or encounters an error, it signals an event to the camera. When the camera receives the event, it may notify the user that the image has been printed or that an error has occurred.