Bulletin of Applied Computing and Information Technology

Lookup and discovery in a Jini architecture

03:02
2005, Jul

Zucker, R. (2005). Lookup and discovery in a Jini architecture. *Bulletin of Applied Computing and Information Technology*, 3(2). Retrieved June 2, 2015 from http://www.citrenz.ac.nz/bacit/0302/2005Zucker_Jini.htm

Abstract

Jini is a Java based software architecture to enable dynamic, adaptive systems. Because lookup and discovery is a critical portion of the Jini architecture, it is the subject of much ongoing study. The studies have resulted in alternative solutions both within and outside the Jini architecture. In this paper, we introduce the Jini Architecture in general including some examples. We explore the Lookup and Discovery Services provided by the Jini Environment in more depth and compare the Jini Lookup services to other implementations and architectures, incuding a discussion of the ways for Jini to work with these competing architectures.

Keywords

Adaptive architecture, service discovery, Jini

1. Introduction

Industry leaders have noted that PCs are so general purpose that less than 5% of their capabilities are being utilized. Market analysts predict that the PC market will top off at around 60% of the home market, based on the fact that after 10 to 15 years, the growth has leveled off at 50%. Bill Gates claims that the number of non-PCs will outnumber PCs connected to the Internet within the next ten years (Clark, 1999). The advances in software and chip technology will enable an increase in smart appliances with inexpensive memory, processors, and displays, specific to that particular appliance. The Swiss Army knife approach to computing may be replaced by inexpensive, specialized, tools, much the same way as earlier stereo systems that included a radio, turntable, speakers, etc. gave way to the more flexible specialized components in use today. Jini is a "Java-based architecture that can make networking your computer devices as easy as plugging your telephone into a wall-jack [that] could bring a dramatic shift to the existing ideas of distributed computing. Jini is technology that can make a network look like one large computer"(Clark, 1999).

It has been suggested that computer architecture will be concentrating on the adaptive architectures (Kubiatowicz, 1998) to enable the addition of these specialized devices and also allow for client services interaction. Jini represents a step in this direction by providing an architecture to support client services programming in a transparent manner without human intervention (Stang & Whinston, 2001). When someone connects to the World Wide Web the connection is seamless. The user may act and react to the web pages without affecting the web one way or the other. He/she may receive and send e-mail at will. If the user disconnects, voluntarily or involuntarily, the web continues without interruption and the e-mail that he/she has sent is still going to arrive and incoming email is held until the user can log on again. This concept is the basis for a new architecture based on network based computing (Waldo, 1999). It has been suggested that Jini may be the key to enabling these devices to be connected to the Internet (Clark, 1999). As the technology matures, new uses for the Jini technology include the use of enterprise programming.

Section 2 of this paper addresses the advantages of the Jini architecture in general and section 3 contains specific examples of Jini applications. Section 4 briefly gives an overview of the Jini process, while sections 5 and 6 specifically address lookup and discovery . Section 5 describes the lookup and discovery processes as proposed by Sun and section 6 introduces alternatives to these processes within the Jini framework. Section 7 provides an overview of competing technologies and section 8 discusses bridges between these competing

technologies and Jini. The paper concludes with Section 9.

2. Advantages of the Jini Architecture

Key advantages of the Jini architecture include:

2.1 Reliability

In a Jini environment, reliability is measured by how well the network or a device performs in the presence of disturbances. A Jini client relies on the health of the services provided (Stang & Whinston, 2001). Similar to the Internet, processes begin and end in a controlled or uncontrolled manner. In a controlled manner a process is started and eventually stopped. In an uncontrolled manner, processes are terminated accidentally by the user, or the processor or a portion of the communication network breaks down. "Jini technology can handle these changes because it expects devices to randomly move in and out of the network" (Clark, 1999). Here again, the Internet serves as a model. The Internet maintains multiple communication routes between machines. If a path breaks down, the Internet routing protocols uses another path. The comparison breaks down a bit when a server is no longer available. In a Jini-based system, the client will look for an alternate processor. If the client discovers an alternate processor it will then connect to that server or if the crashed server recovers it will reconnect to it. If neither an alternate server nor the existing server is able to provide a connection the Jini client will either wait or notify the user. The ability to find a server, wait, or notify is built into the Jini technology, eliminating the need for the application program to handle the task (Stang & Whinston, 2001).

2.2 Scalability

If the cost required to add more capability is less than the benefits of the additional capability, the system is said to be scalable (Stang & Whinston, 2001). Very much like the Internet, there is no centralized control required for a Jini network. In the process of discovery and lookup a Jini device inserts itself into the network, the entire Jini network is managed using lookup servers to manage entry and exit (Stang & Whinston, 2001), making Jini higly scalable.

2.3 Security

In an enterprise-computing environment, security is critically important (Stang & Whinston, 2001). Security must deal with both viruses and rogue services. Virus code must move from one computer to another in order to infect the computer. Since Java enforces a strong security policy, the client's security policy must be satisfied prior to accepting code. This is done via a principal and an access control list that is associated with the object. A problem that is still undergoing research is that of a rogue service. The rogue service is when a service registers itself as capable of one service yet provides another, tricking a client (Clark, 1999).

3. Examples Of Jini Applications

The following examples help to illustrate the diversity of Jini applications:

3.1 Railway Systems

Mobile computing is used on trains in Switzerland for tracking the train make-up and also for determining maintenance issues with the train itself (Nieva, Fabri, & Benammour, 2000).

3.2 Robotics

Jini is used to simplify a robots design by removing sensing hardware from the robot and allowing external sensing data to help guide and coordinate the robots movements (Valavanis, Nelson, Doitsidis, Long, & Murphy, n.d.).

3.3 Network Aware Appliances

Network aware (NA) appliances may be found in the home, car, or business. Examples include thermostats, lights, consumer electronics, etc (Reilly & Taleb-Bendiab, 2002).

3.4 Enterprise Distributed Computing

Stark and Whinston have proposed a software license tracker which would use Jini and its leasing properties in particular for tracking software license compliance. Enterprise distributed computing is considered by some to be the direction for future computing architectures (Stang & Whinston, 2001).

4. How Does Jini Work?

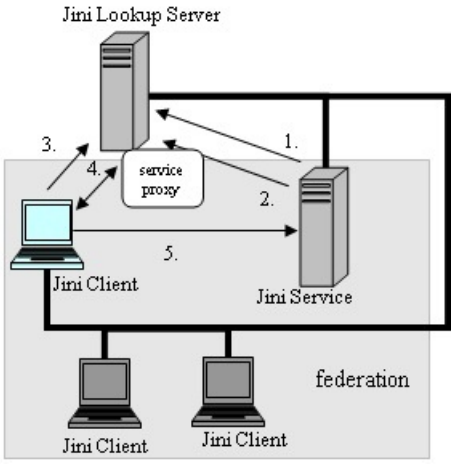
Jini provides clients and services with a unified communication process. It is not intended to do the actual processing of the client or the service. Jini functions much like a broker, accepting request from clients for services and connecting the service with the client. The broker also maintains a list of available services as they arrive and leave. In both cases, the broker is not required to know anything specifically about the client or how the service is provided. As long as the device has an operating system that supports the Java Virtual Machine (JVM), it can be plugged into the network using Jini (Clark, 1999). While this seems to be an unreasonable request, keep in mind that a specific inexpensive JVM could be placed on a chip for such a purpose. This paper also addresses devices that cannot support a JVM using drivers. Once the connection between client and service has been made, Remote Method Invocation (RMI) is used to simplify communication between components.

The three categories of Jini system components are: *infrastructure*, the *programming model*, and *services* (Sun Microsystems, 2003).

4.1 Infrastructure

The infrastructure is described in Figure 1; it is responsible for building a federation (shaded box) between services and clients. A federation is not limited to one client and one service, many to many relationships are also possible. It is also possible for a service to be a client of another service. In addition to lookup and discovery, Jini provides the distributed-system services for registration and leasing. The lookup server will create a registration object to be stored on the service processor to act as a proxy. Services communicate with each other by using a service protocol, which is a set of interfaces written in the Java programming language. The Jini system defines a small number of such protocols that define critical service interactions, however the set of protocols is not limited (Sun Microsystems, 2003). Leasing is a way of setting the duration for delivery of service to clients requesting the service. The lease can be extended or cancelled as the need arises. While Jini is developed in Java, since it is processing in the Java application environment, it will accept any bytecode that is Java compliant, even if not written in the Java language (Sun Microsystems, 2003).

1. Service discovers lookup server through multicast discovery.
2. Service registers service proxy with lookup server using join.
3. Client discovers lookup server through multicast discovery.



4. Client performs lookup to known lookup server and receives server location from .service proxy
5. Client directly accesses server and executes code on the server

Figure 1. Lookup and Discovery (based on Stang & Whinston, 2001)

4.2 Programming Model

The programming model consists of a set of Java interfaces that enables the construction of reliable services. There is some overlap between the categories, including those that are part of the infrastructure and those that join into the federation (Sun Microsystems, 2003).

4.3 Services

Services are simply entities within the federation that offer functionality to clients or other services (Sun Microsystems, 2003).

5. Exploring the Discovery Process

In order to provide discovery, a host is required. The host is a single hardware device connected to one or more networks. The host must have a functioning Java Virtual Machine (JVM) with access to all the packages required by Jini (for more details, see http://java.sun.com/products/jini/2.0/doc/api/) and a properly configured protocol stack to handle the protocols being used. The discovery process requires three related protocols: *multicast request protocol, multicast announcement protocol,* and the *unicast discovery protocol*, using a specific target server. As Jini has evolved two versions of these protocols have emerged. The difference in the two versions (1 for Jini 1.0 through 1.2 and 2 for Jini version 2.0) is primarily in the encoded representation of the data sent. It should be noted that in version two, with the exception of the unicast request packet (see Table II) which increased from a single integer value, the packets have simplified datatypes and/or decreased in size. For the sake of brevity only the version 2 packets are shown. Sun has recommended that variable size packets should be limited to less than 512 bytes to avoid fragmentation. However this is not a requirement (sun Microsystems, n.d.). The overall interaction process has remained virtually unchanged.

The multicast request protocol deals with new processes wishing to locate a lookup service. The multicast request service is not an RMI based service. This is important to note since RMI is vulnerable to snooping (Chakraborty & Chen, 2000). It uses the multicast datagram facility which is part of the networking transport layer. Request datagrams are encoded using Java

serialization to provide platform independence.

The multicast announcement protocol allows the lookup services to advertise their existence either at startup or recovery from a failure. The format for the packet is the same as the request packet, shown in Table 1.

Table 1. Request Packet Format (based on Sun Microsystems, n.d.)

Count	Data Type	Description
1	int	protocol version
1	byte	multicast packet type
1	long	discovery format id
variable	byte	discovery format data

The unicast discovery protocol is used to provide a connection with a specific lookup service beyond local devices (WAN) or for using specific devices for a long period of time. The two packets for the unicast discovery protocol are shown in Tables 2 and 3.

Table 2. Unicast Request Packet (based on Sun Microsystems, n.d.)

Count	Data Type	Description
1	int	protocol version
1	unsigned short	proposed format ID count
variable	long	proposed format IDs

Table 3. Unicast Response Packet (based on Sun Microsystems, n.d.)

Count	Data Type	Description
1	int	protocol version
1	long	selected format ID
variable	byte	discovery format data

Lookup services maintain strings called groups which they advertise. Entities specify the groups they wish to communicate with via the multicast request protocol. A lookup service may have zero or multiple groups associated with it. The strings values are arbitrary, but to eliminate name conflicts Sun recommends using a Domain Name Service (DNS) scheme (Sun Microsystems, n.d). A special group called the *public* group is denoted by an empty string. The lookup services default to this public group and discovering entities should attempt to find lookup services from this group. There are several formats for discovery packets. Tables 4 and 5 are examples for the multicast discovery context using the plaintext format. Other formats include SHA1withDSA and SHA1with RSA (Sun Microsystems, n.d.).

Table 4. Multicast Requests, Version 2 (based on Sun Microsystems, n.d.)

Count	Data Type	Description
1	String	multicast response host
1	unsigned short	multicast response port
1	unsigned short	group count
variable	String	requested groups
1	unsigned short	service ID count
variable	ServiceID	heard lookup service IDs

When a service starts up, it tries to register using unicast discovery with a set of specific lookup services. If lookup services do not respond, it retries or gives up. If the service gives up, the service does not remove the non-responding lookup service from the set of lookup services. It is also interesting to note that the order of unicast or multicast steps is not important. What is important is that all of the lookup services must have the same registration from the service and must use the same ID for all registrations (Sun Microsystems, n.d.).

Sun recommend that the three components (infrastructure, programming model, and services) should be used as a system (for simplicity and compatibility) and not separated (Sun Microsystesms, 2003). Because of the importance of lookup and discovery, alternative approaches to the Jini discovery process have been proposed. Various groups are trying to improve on the performance, accuracy, and security of these components.

6. Alternatives to the Discovery Process

6.1 Service Location Protocol

The Service Location Protocol (SLP), developed by the Server Protocol Working Group (SVRLOC) of the Internet Engineering Task Force (IETF) is a language independent specification catering to both software and hardware services. The SLP infrastructure uses the agent metaphor and is conceptually handled in much the same way as the Jini lookup process. Clients are called user agents, services are called service agents and lookup servers are called directory agents. Discovery is based on service attributes which are descriptions of the services offered using configuration values of the properties for that particular service. The properties correspond to a network, configurable SLP, or the SLP agents. The format of the file containing the properties are new-line delimited. Security is handled by the API library and uses exceptions to notify clients. It is the responsibility of the user agent to catch an authentication exception at any time, since it may contact a Directory Agent with authenticated advertisements (Kempf & Guttman, 1999).

6.2 Ronin Agent Framework

The Ronin agent framework is a Jini based hybrid architecture combining agent and service oriented architectures (Chakraborty & Chen, 2000). The premise of Ronin is that developing in Jini is easy, developing smart in Jini is hard (Chen. n.d.). The approach taken is to provide autonomous agents with artificial intelligence that can adapt to their environment, and then model the Jini service to have the same properties. Rather than develop a Java based artificial intelligence, the developer chose to make a Java interface to communicate with Prolog. For building an inference engine and knowledge base, the Prolog engine is an excellent tool (Chen, n.d.). A quick way to provide knowledge based services was to provide Prolog with a Jini service interface, called Distributed Prolog KB (DPKB). The prolog engine runs on the server side to maintain the lightweight services. A client can have multiple knowledge bases (KB) from a single server since KBs are identified by ID and password. The developers of the Ronin Framework created a prototype called Agents2Go. Figure 2 illustrates the steps taken using the Ronin Framework. A client discovers the broker agent through the standard Jini lookup service. The client then submits a recommendation request to the broker with constraints and a personal profile attached. The broker preprocesses the request, constraints, and profile info, and then translates them into a number of queries in Prolog. It depends on the DPKB service for knowledge representation and reasoning. If the broker KB does not have sufficient knowledge to make a recommendation, then the broker will try to discover and lookup local service agents through the Jini lookup service. The broker will negotiate with these agents using a descriptive agent communication language, KQML. The message content is expressed in Prolog. After the broker has received enough information, it restarts the recommendation inference process and replies to the original request with the recommendation.

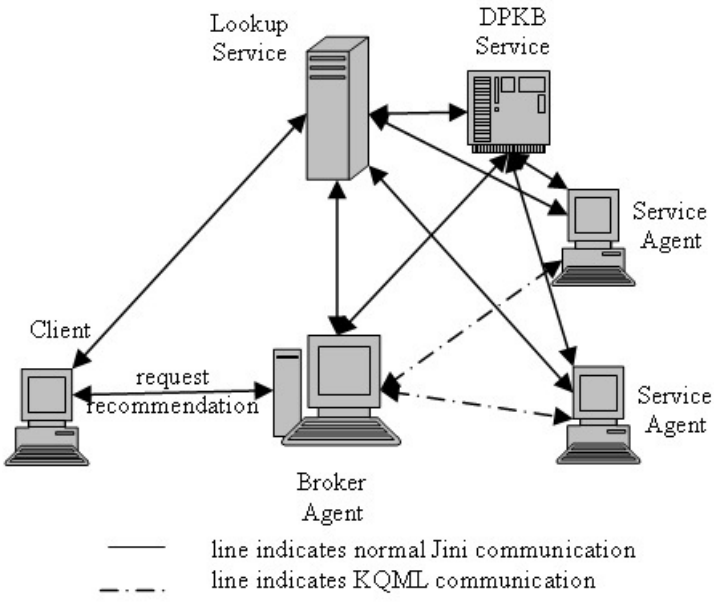


Figure 2. Agents2Go Processing

6.3 XReggie

The XReggie project focuses on the service matching aspect of discovery, using the eXtensible Markup Language (XML), while maintaining the Jini infrastructure. The use of XML offers both structure and flexibility in service descriptions for advertisement and discovery (Chakraborty & Chen, 2000). The Ninja Service Discovery Service, though not part of the XReggie project, uses XML as the format for service descriptions and is discussed next.

6.4 Ninja Service Discovery Service

The University of California, Berkeley has developed a Java based alternative called Ninja *Service Discovery Service* (SDS) (Hodes, Czerwinski, Zhao, Joseph, & Katz, 2002). Similar to Jini, SDS servers are responsible for the creation of a federation between clients and services. The service description is provided through XML which provides greater flexibility and use. XML also offers the advantage of validation through Document Type Definitions (DTD). Figure 3 shows a sample of a query, Figure 4 shows a match, and Figure 5 is a mismatch.

To provide security, all communication between the SDS servers, clients, and services is encrypted using a hybrid of symmetric and symmetric-key cryptography. To provide additional security, SDS authenticates endpoints. Every component of SDS has an associated principal name and public-key certificate that can be used to verify the client's identity to all other components. SDS also provides security through advertisement and location of private services. Private services specify capabilities, signed messages indicating that a particular user has access to the service that must be met to make use of the service. When a client wishes to use a private service it must supply the user's capabilities in order to have access.

```
<?xml version="1.0">
<printcap>
<color>yes</color>
<postscript>yes<</postscript>
</printcap>
```

Figure 3. XML Query (based on Hodes et al., 2002)

```
<?xml version="1.0">
<!doctype printcap system http://www.sample/printer.dtd >
<printcap>
<name>lc342</name>
<color>yes</color>
<postscript>yes</postscript>
<location>ENB 342</location>
</printcap>
```

Figure 4. Service Match (based on Hodes et al., 2002)

```
<?xml version="1.0">
<!doctype printcap system http://www.sample/printer.dtd >
<printcap>
<name>lw342</name>
<color>no</color>
<postscript>yes<</postscript>
<location>ENB 342</location>
</printcap>
```

Figure 5. Failed Match (based on Hodes et al., 2002)

For communication security SDS uses NinjaRMI which supports the following enhancements over the standard Java RMI: "Multiple communication protocols, including TCP, UDP, and multicast; Reliable, unreliable, one-way, and multicast communication semantics; API calls allowing both client and server objects to determine the hostname and socket port number of the connected peer; and the ability for server code to register callbacks which are invoked when certain events occur, such as socket creation and destruction" (Welsh, 1999).

For scalability the SDS Servers are organized into multiple shared hierarchies. The coverage of a particular server is called a *domain*, defined as a changeable list of Classless InterDomain Routing (CIDR) addresses (Hodes et al.,2002).

As the number of lookup servers increase and the number of services become available, new problems arise due to the amount of services and queries entering the system. Adding to the problem is the introduction of multiple-criteria searches. SDS addresses both wide area distribution and complex queries, while Jini does not. SDS uses an approach called filtered query flooding or simply *query filtering* (Hodes et al., 2002). Query filtering is a hybrid technology, combining flooding, mapping, and centralization. Flooding sends messages to all nodes of the system. Mapping involves a hashing scheme which maps criteria to a particular node. The problem here is the namespace hashing scheme does not work well with multiple keys (Hodes et al., 2002). Finally, centralization requires a single source (or cluster) lookup. Problems with centralization include: a single source for failure or litigation, and data sharing. SDS uses a set of filtering schemes to remove false positives (declaring a match when a match is not made) and false negatives (not finding a match where a match exists). Jini uses the serialized object matching mechanism from JavaSpaces and does not take into account class versioning which may lead to false positives (Hodes et al., 2002). The SDS filtering schemes are: all-pass/null filtering (flooding); brokering; centroid-indexed terminals; and Bloom-filtered crossed terminals (mapping). While Jini does offer the ability to search by subtypes, it requires that a Java interface be used in the lookup process as a template, possibly introducing a storage and bandwidth problem. While no comparison figures were provided by the SDS developers, results showed performance to be as follows: cryptograph timing (encryption/decryption) ranged from 2.0/1.7 ms for Blowfish, to 15.5/142.5 ms for RSA; the secure query latency was 82.0 ms (Hodes et al., 2002). In order to make SDS available to Jini users the Ninja group has created a Jini proxy that listens for Jini services using the Jini discovery protocol. Once a new service is discovered, it transforms the descriptions to the SDS system. This allows SDS to discover Jini-enabled devices much like the SLP-Jini bridge (Hodes et al., 2002).

7. Competing and Related Service Discovery

The interest in dynamic client/server/services has been great and competing architectures have been evolving in addition to the Jini based architectures. These include UPnP, Salutation, Home Audio Visual Interface (HAVI), JetSend, TSpaces, Inferno, and Bluetooth (Chakraborty 7 Chen, 2000; Gupta, Talwar, & Agrawal, 2002). UPnP, Salutation and Bluetooth will be briefly explained. Table 6 summarizes some of the key concepts and terms used in Jini, SLP and UPnP.

Table 6. Mapping Concepts among Service-Discovery Systems (basd on Dabrowski et al., 2002)

Generic Model	SLP	Jini	UPnP
Service User (SU)	User Agent	Client	Control Point
Service Manager (SM)	Service Agent	Service or Device Proxy	Root Device
Service Provider (SP)	Service	Service	Device or Service
Service Description (SD)	Service Registration	Service Item	Device/Service Description
Identity	Service URL	Service ID	Universal Unique ID
Type	Service Type	Service Type	Device/Service Type
Attributes	Service Attributes	Attribute Set	Device/Service Schema
User Interface	Template URL	Service Applet	Presentation URL
Program Interface	Template URL	Service Proxy	Control/Event URL
Service Cache Manager (SCM)	Directory Service Agent (optional)	Lookup Service	not applicable

7.1 Universal Plug and Play

Universal Plug and Play (UPnP) is a set of service discovery protocols being developed by the Universal Plug and Play Forum (with over 470 companies) led by Microsoft (Allard, Chinta, Gundala, & Richard III, 2003; Chakraborty & Chen, 2000; DAbrowski, Mills, & Elder, 2002; Gupta, Talwar, & Agrawal, 2002). UPnP extends the hardware plug and play peripheral concept to a highly dynamic model with many devices from many vendors. It involves a six step process, numbered from 0-5. Figure 6 shows the steps taken during discovery.

0) Addressing - Addressing represents the foundation of UPnP. In this step devices obtain an IP address. If the device does not have an assigned IP address, assigned by a Dynamic Host Configuration Protocol (DHCP) Server, Auto-IP will create an address. Addressing enables steps 1 through 5.

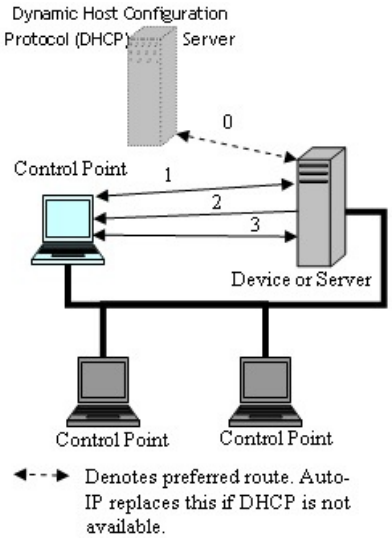
1) Discovery - When a device is added to the network, it advertises its services through multicast to *control points* (other services) on the network or if a control point is added, the discovery process searches for devices of interest. Discovery messages consisting of a few essential specifics (e.g. type, identifier, and a pointer to more specific information) are the means of exchanging data.

2) Description - Because the discovery message is brief the control point must learn more about the device and services. This is accomplished by retrieving a description from the device using the pointer (URL) from the discovery message.

3) Control - Once a control point is aware of a device and its services, it can then invoke actions and poll those services for the values of its properties. Invoking actions is similar to remote procedure calls.

4) Eventing -This step takes place when one or more of the properties changes from the source of the event, called a publisher, and the destination of the event (typically a control point), called a subscriber, has subscribed to receive this information.

5) Presentation - If a control point has a URL for presentation, it can then present this information into a browser for the user to view and/or control the device. This is dependent on the specific capabilities of the page and the device.



0 - Addressing

1 - Discovery - advertisement and discovery using multicast

2 - Description

3 - Control

steps 4 and 5 are part the processing, not of the overall discovery.

Figure 6. UPnP Discovery (based on Dabrowski et al.,2002)

The UPnP Forum has created the UPnP Template Language, which has been derived from XML, as the language for communication within the architecture (UPnP, 2000).

7.2 Salutation

"Salutation is designed for pervasive, dynamic and heterogeneous networking, where the discovery solution must be separate from network protocols" as defined by the Salutation Consortium. It is offered royalty free. Salutation is based on capability descriptions, adapting standardized capability semantics. Salutation sits above the protocol layers which are used by Jini (RMI and JVM) and UPnP (HTTP and TCP/IP) so that it can leverage virtually any network protocol. Salutation uses the full spectrum of discovery from peer to peer to directory centric. Jini is directory centric, through its lookup service, and UPnP uses Simple Service Discovery Protocol (SSDP) which to some degree also relies on a directory. Salutation uses the DOC Storage Functional Unit to store proxy services (Jini), device drivers, etc. since it is architecturally independent. The location of the DOC Storage Unit is not fixed and can be loaded on a service device, the supporting network, or the Internet (Salutation Consortium, n.d.).

7.3 Bluetooth

Bluetooth is a computing and telecommunications specification describing how mobile phone, computers, and portable devices can interact using short-range radio link connections. The areas that Bluetooth provides short-range connectivity are: data and voice; cable

replacement; and ad hoc networking. The relationship between Jini and Bluetooth is much like software and hardware. Bluetooth enables communication, Jini establishes and defines the data flow in the communication pipe (Gupta, Talwar, & Agrawal, 2002) . The Bluetooth architecture fits the OSI seven-layer network reference model (Physical, Data Link, Network, Transport, Session, Presentation, and Application). Jini is located at the session and presentation layer (ibid).

8. Bridging Architectures

The alternative and competing technologies introduce a different problem for dynamic devices. Since each of the architectures offers different strengths and none are dominating the market, the question is whether they , can interoperate (Allard, Chinta, Gindala, & Richrad III, 2003). Fortunately a great deal of work has been done to allow the architectures to work together. The Jini Device Architecture Specification suggests three approaches to connect non-Jini devices to a Jini network: a *Device Bay* consisting of a number of devices sharing a co-located JVM (Figure 7); *A Network Surrogate Option*, which is similar to the device bay except the devices are not co-located with the JVM, but are accessible to the JVM through the network (Figure 8); and finally using the Internet Inter-Operability Protocol (IIOP) developed by the Object Management Group (OMG). There are now bridges to ease the use of non-Jini based architectures. Included in the list of bridges are the Jini-UPnP Interoperability Framework and the SLP-Jini Bridge (McDowell & Shankari, 2000; Sun Microsystems, 2004).

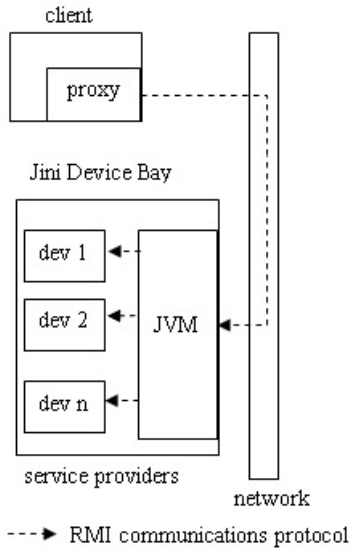


Figure 7. Device Bay Option (based on Sun Microsystems, 2004)

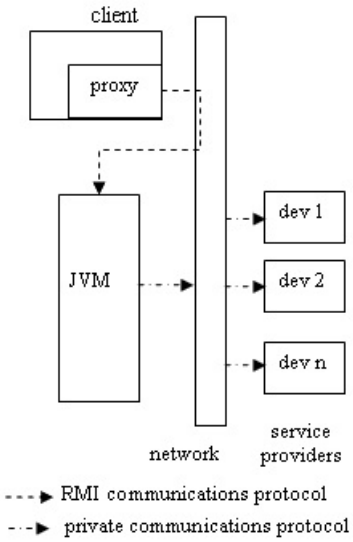


Figure 8. Network Surrogate Option (based on Sun Microsystems, 2004)

8.1 Jini-UPnP Interoperability Framework

The Jini-UPnP Interoperability Framework has been proposed as a way to allow UPnP and Jini

devices to intermingle. The goal of this project was to introduce a minimum amount of code into the service that would allow the device to interact with an unmodified Jini and UPnP architecture. The approach was to fool Jini Clients into believing they were communicating with Jini services that were indeed UPnP services and vice versa.

To accomplish this, the framework utilizes virtual services and clients, that is a UPnP client converses with either a true UPnP service or a virtual UPnP service (from a Jini Device) and a Jini Client converses with either a true Jini service or a virtual Jini service (from a UPnP device). The framework uses six major components, highlighted in Figure 9.

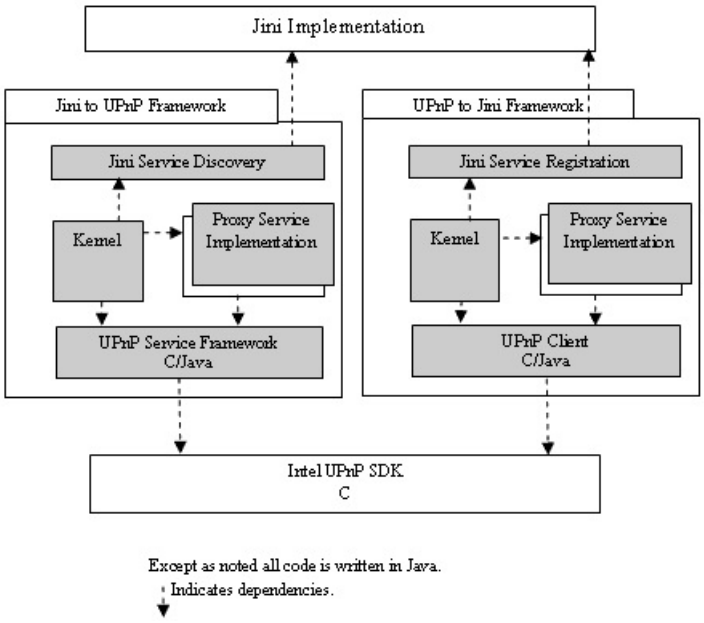


Figure 9. Jini/UPnP Framework Design (based on Allard et al., 2003)

1. UPnP Service Framework which is responsible for providing basic UPnP Service using a Java Native Interface (JNI) interface. This component is responsible for supporting the development proxies for new service types.
2. The UPnP Client provides basic UPnP client function in order to transmit commands to a UPnP service for a Jini Client.
3. The Jini Service Discovery is responsible for the discovery of known Jini service types.
4. Jini Service Registration is responsible for registering virtual Jini services with available Lookup services.
5. Proxy Service Implementation. This is the only portion of the framework that must be developed for each particular service. A pair of proxy service implementations must be constructed, one to support Jini to UPnP interoperability and the other to support UPnP to Jini interoperability.
6. The Kernel is the controller for this architecture responsible for: controlling the other modules; instantiating the appropriate proxies; advertisement and registration of the virtual services and finally garbage collection.

The Proxy Service Implementation is a source of concern since it requires a pair of proxies to be created for each new service. The amount of effort to design and program a proxy service implementation consisted of one programmer programming in Java for one half day. The resulting code was approximately 100 lines of Java code. The design team is currently investigating methods to automate the Proxy Service Implementation (Alard, Chinta, Gundala, & Richard III, 2003). There are a number of steps involved using these components to make the framework work. Below is an example of a Jini client discovery and registering with a UPnP service (refer to the right side of Figure 9).

1. The kernel uses the UPnP Client to discover services which have proxy service implementations.
2. The UPnP client forwards the request to the UPnP SDK.
3. The UPnP transmits service request messages to the UPnP services.
4. The UPnP services return UPnP service descriptions to UPnP client via the UPnP SDK.
5. The UPnP client parses the description documents.
6. The UPnP client creates a UPnPService interface to interact with the UPnP service,

sending it to the kernel.
7. The kernel locates the appropriate proxy service class
8. The found proxy service class is returned to the kernel.
9. The kernel creates a Jini service proxy instance.
10. The kernel creates a Jini Service RMI stub and sends it to Jini service registration.
11. The Jini service registration creates a Jini service description and forwards it to the Jini implementation.
12. The Jini implementation registers the service description with available Jini lookup services.
13. A Jini client requests a service and downloads the RMI service stub.
14. Methods invoked in the stub go through the kernel to the UPnP SDK to the UPnP service and back.

8.2 SLP-Jini Bridge

The SLP-Jini Bridge is useful for thin clients that are too small to support a JVM. Instead of service agents advertising services, they advertise the presence of a Jini driver factory. The SLP-Jini Bridge, posing as an SLP user agent is responsible for lookup and discovery of all services advertised with SLP that offer a Jini driver factory. The service advertisement is used to create a Jini service registration which in turn is registered with all the appropriate Jini Lookup services. While this may explain how lookup and discovery works, it does not explain how the SLP-Jini bridge makes the client service work. The SLP thin service contains a Java ARchive (JAR) file, which will be requested from the service. The JAR file contains Java code to be run on the client system, which has a JVM. The Java code has a factory method for instantiating the driver to be used to drive the service. The interesting thing here is that the service stores the necessary code to interact with itself and sends the code to a processor that has the JVM to actually produce the driver that the service needs (Guttman & Kempf, 1999).

9. Conclusion

Jini is an existing and evolving technology that is providing a new hardware/software paradigm in much the same way that object-oriented programming has created a paradigm shift in software development. It may redefine the way computers are thought of and constructed in the future. As with any new technology, getting the technology to work only opens the door. As an area of interest for future development, clearly adaptive architecture ranks high. Thus the search for improvements within the Jini architecture and its alternative approaches appears to be an ongoing activity. The discovery process is vital to this technology and is an important area to study for optimization in terms of security, accuracy, and speed.

Since adaptive architectures are an evolving technology, new efficiencies are being proposed not only for Jini but for the competing architectures. The discovery process is currently one of the critical points to making client services architecture successful. As adaptive systems become more prevalent, performance issues with respect to speed, accuracy, and resources will need to be addressed. Metrics should be developed to help to evaluate these systems. Future research could also address scalability, and reliability issues.

References

Allard, J., Chinta, V., Gundala, S., & Richard III, G. (2003). Jini meets upnp: an architecture for jini/upnp interoperability, *Proceedings of the 2003 Symposium on Applications and the Internet SAINT 2003,* pp. 268-275.

Arnold, K. (1999). The Jini architecture: dynamic services in a flexible network, *Proceedings of the 36th ACM/IEEE Conference on Design Automation*, pp. 157-162.

Chakraborty, D.& Chen, H. (2000). Service discovery in the future for mobile commerce, *ACM Crossroads, 7*(2), pp.18-24.

Chen, H. (1999). Developing agent-oriented Jini services. Retrieved October 15, 2004 from http://www.jini.org/meetings/second/jcm2_agents+jini_chenbof-bw.pdf

Clark, D. (1999). Network nirvana and the intelligent device. *IEEE Concurrency, 7*(2), pp. 16-19.

Dabrowski, C., Mills, K., & Elder, J. (2002). Perfroamce evaluation of software architecture: Understanding consistency maintenance in service discovery architectures during communication failure, *Proceedings of the Third International Workshop on Software and*

Performance, pp. 168-178..
Gupta, R., Talwar, S., & Agrawal, D. (2002) Jini home networking: a step toward pervasive computing, *Computer, 35*(8), pp. 34-40.
Guttman, E. & Kempf, J. (1999). Automatic discovery of thin servers: SLP, Jini and the SLP-Jini bridge, *Proceedings of the 25th Annual Conference of the IEEE Industrial Electronics Society IECON'99, 2*, pp. 722-727.
Hode,s T., Czerwinski, S., Zhao, B., Joseph, A., & Katz, R. H. (2002). An architecture for secure wide-area service discovery, *ACM Wireless Networks, 8*(2/3), pp. 213-230.
Kempf, J. & Guttman, E. (1999). Network working request for comments: 2614, Sun Microsystems. Retrieved October 10, 2004 from http://www.faqs.org/ftp/rfc/pdf/rfc2614.txt.pdf
Kubiatowicz, J. (1998). Lectures 1: Review of technology trends and cost/performance (slide show), University of California, Berkeley
McDowell, C. & Shankari, K, (2000). Connecting non-Java devices to a Jini network, *Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages TOOLS 33*, pp. 45-56.
Nieva, T., Fabri, A., & Benammour, A. (2000). Jini technology applied to railway systems, *Proceedings of the International Symposium on Distributed Objects and Applications DOA '00*, pp. 251-259.
Reilly, D. & Taleb-Bendiab, A . (2002). A Jini-based infrastructure for networked appliance management and adaptation, *Proceedings of the 5th IEEE International Workshop on Networked Appliances*, pp. 161-167.
Salutation Consortium (n.d.). *Discover quality service discovery*. Retrieved October 10, 2004 from http://www.salutation.org/techtalk/tc18.htm, Liverpool.
Stang, M. & Whinston, S, (2001). Enterprise computing with Jini technology, *IT Professional, 3*(1).
Sun Microsystems (2003). Jini™ architecture specification version 2.0. Retrieved November 3, 2004 from http://wwws.sun.com/software/jini/specs/jini2_0.pdf
Sun Microsystems (2004). Jini device architecture specification. Retrieved November 3, 2004 from http://wwws.sun.com/software/jini/specs/jini1.1html/devicearch-spec.html#996905
Sun Microsystems (n.d.). DJ - discovery and join, jini technology core platform specification Retrieved November 3, 2004 from http://wwws.sun.com/software/jini/specs/jini1.2html/discovery-spec.html
UPnP (2000). Contributing Members of the UPnP™ Forum: Upnp™ device architecture version 1.0. Retrieved October 6, 2004 from http://www.upnp.org/download/UPnPDA10_20000613.htm
Valavanis, K., Nelson, A., Doitsidis, L., Long, M., & Murphy, R. (n.d.). Validation of a distributed field robot architecture integrated with a MATLAB based control theoretic environment: a case study of fuzzy logic based robot navigation. Retrieved October 19, 2004 from http://crasar.csee.usf.edu/research/Publications/CRASAR-TR2004-25.pdf
Waldo J. (1999). The jini architecture for network-centric computing, *Communications of the ACM, 42*(7), pp. 76-82.
Welsh, M. (1999). Ninjarmi: a free java rmi, ninjarmi v1.2. Retrieved September 28, 2004 from http://www.eecs.harvard.edu/~mdw/proj/old/ninja/ninjarmi.html