A day in the life of Jini: a peek at service-oriented architecture for internet appliances

Billy Lim*

School of Information Technology,
Illinois State University, Normal, IL 61790-5150
E-mail: bllim@ilstu.edu
*Corresponding author

Hao Zheng

Kosan Biosciences, 3832 Bay Center Place,
Hayward, CA 94545
E-mail: zheng@kosan.com

Abstract: With the advent of the web, mobiles, and service-oriented technologies, many are looking forward to seeing how these technologies can shape our lives on a daily basis. This paper describes a project that is set out to simulate the technologies in question through a day in the life of a knowledge worker in the 21st century. The project uses Jini, a distributed object computing infrastructure that allows services to announce their presence and what they can do when networked together. This paper briefly introduces the architecture of Jini, including its plug and play and self-healing features, describes the scenarios simulated in the day of the life of a worker, and reports on the challenges that lie ahead for the technology used and for the scenarios to be realised.

Keywords: Jini technology; mobile computing; distributed object computing; service-oriented architectures; ubiquitous computing.

Reference to this paper should be made as follows: Lim, B. and Zheng, H. (2004) ‘A day in the life of Jini: a peek at service-oriented architecture for internet appliances’, *Int. J. Mobile Communication*, Vol. 2, No. 2, pp.199–216.

Biographical notes: Dr. Billy Lim is a tenured Associate Professor in the School of Information Technology of Illinois State University. He received his BS degree in Computer Science from the University of Toronto, Canada and MS and PhD degrees in Computer Science from the University of Louisiana at Lafayette, Louisiana. His research and teaching interests are in the areas of web development technologies, object-oriented (OO) programming, and new generation database management systems. He has published numerous journal papers in these areas and also presented papers in many international, national, and regional conferences.

Hao Zheng received his BS degree (Chemistry) in 1995 and MS degree (Organic Chemistry) in 1998 both from Nanjing University, Nanjing, China. He also received his MS degree (Computer Science) from Illinois State University in 2002. He is currently working as a Research Associate in a biotech company at Silicon Valley, California.

1 Introduction

Over the past decade or so, the internet/web has dramatically transformed our societies and changed our lives in many different ways, from simple electronic news publishing to e-commerce in many different forms (e.g., business-to-consumer, business-to-business, peer-to-peer, etc.), all of which can be done anywhere, anytime, and by anyone.

One of the advances in this net/web arena is distributed object computing in the form of internet appliances. This is where a client device (e.g., a PC, PDA, cellular phone) can communicate with, say, a net-aware refrigerator, which in turn can make a maintenance call to a repair store and order groceries from an online grocer. (IBM had a TV commercial that describes this very scenario to showcase the future of internet appliance computing.) With the advent of mobile technologies [1] and location-based services [2], the ability to use a device such as a PDA to conduct business transactions (e.g., pay for a soda in a vending machine (another TV commercial on the future of computing), locate the nearest Starbuck coffee shop, etc.) is especially intriguing.

In the above example, all these devices are examples of internet appliances, and they communicate to each other via some common protocol. A protocol that has received a lot of attention is the architecture by Sun Microsystems known as Jini™ [3–7]. Jini, created by Bill Joy, who is often known as the 'Edison of Silicon Valley', leverages the unique distributed computing characteristics of Java technology, specifically its ability to move both code and data from machine to machine, to create a ubiquitous network.

One of the goals of Jini technology is to make adding an electronic device to a network as easy as plugging in the base unit of a new cordless phone – in other words, 'spontaneous network' or 'network plug and play'. Another goal of Jini is to make distributed computing a near-term reality, i.e., allow computing capabilities to be shared among many diverse machines across a common network. Jini technology lets users easily access the needed power and features of any appropriate device on the network by creating 'federations' or communities of shared data, storage, and computing power.

From the perspective of Jini, there is no real distinction between hardware such as a printer, scanner, or dishwasher and software such as MS Word – everything is considered a service. That is why Jini is considered as 'service-oriented architecture' (SOA) [8,9]. A Jini-enabled service, when started, can spontaneously connect to a network and broadcast its presence and what it can do. A client can search the network, find the service, and use it without configuration. There is no need to set up an IP address, gateway, or router, i.e., true plug and play that is 'administration free'. For instance, when plugged into a network, a Jini-enabled printer can broadcast its presence and capability. Then, a Jini-enabled digital camera can find the printer and send a picture to be printed.

With the popularity of wireless devices all around the world, one can imagine that a technology that supports spontaneous computing such as Jini will play a major role in today's IT evolution. This is because wireless devices, such as mobile phones, PDAs, and pagers, are ideal for joining dynamic, spontaneous networks.

Indeed, because of the promises of Jini, many major vendors have already announced support for the technology. These companies are actively architecting Jini technology into a new breed of consumer products and services. It has broadened beyond the device space to encompass service delivery to and from enterprises as well. Examples of use of the technology by well-known organisations include Cisco's scalable communication framework, Ford's and DaimlerChrysler's IT cruiser telematics systems, and the US

Army's reliable battlefield communication architecture. Also, the burgeoning growth of the Jini technology community has doubled its numbers to more than 50,000 developers in recent years [10].

Recognising that the internet appliances phenomenon (using Jini or other technologies) is probably one of the next big things for the internet/web, the project described in this paper extends a prior work [11] and sets out to simulate more comprehensively internet appliances by using a collection of devices and tasks on a Jini network. The internet working of these devices and tasks and the way they work together on the network are described here. These simulated services include a Jini-enabled calendar, refrigerator, grocery store, airline check in, printer, projector, gaming vendor, and a Jini client to interact with the services. The details of the simulation scenarios are described in the following sections.

The significance of this project is that, from a high level, it gives sufficient evidence to demonstrate what service-based computing is all about and specifically what Jini services are and how the services are consumed in a Jini network. Also, this project could be used as the starting points for developing more complicated and comprehensive services in order to simulate completely a society with internet appliances. Readers wishing to understand and to get started with the burgeoning world of internet appliances can use this project as a reference.

The remainder of the paper is organised as follows. Section 2 describes the simulation of various appliances. The technology under the hood is given in Section 3. Section 4 compares the technology used with some other alternatives. Lastly, the summary and future trends are given in Section 5.

2 Simulation of genie: a Jini community of tomorrow

Imagine a knowledge worker in the 21st century with all the cool technologies of tomorrow at his/her fingertip and now imagine that you can actually simulate this environment today! This is what this project, codename Genie, is set out to do.

To simulate some plausible scenarios that such a knowledge worker, named Jeannie (in this paper, Jeannie is the person who experiences the cool Jini technology whereas Genie refers to the simulation project), might be engaging in for a day, which includes a business trip, the following scenarios are captured. The assumption made here is that Jeannie has a wireless PDA/phone combo that can be connected to the internet. This may be done using Bluetooth, 802.11a (or b or g) Wi-Fi, or any other wireless technologies. In the actual simulation carried out, each of the Jini services was represented by a PC. Also, for simplicity, no authentication mechanism was incorporated. Here are the scenarios:

- Jeannie wishes to check her schedule for the day and thus she connects her PDA to a central calendar server that offers the Jini *calendar service* she needs.
- After checking her schedule, she leaves for the airport for a business trip to San Jose. On the way, she is notified on her PDA that the temperature of her refrigerator at home is not in the normal range (perhaps the kids at home are fooling around with the control panel!). Jeannie then connects to the *fridge service* and accesses the control panel remotely. She adjusts the temperature (and perhaps calls the kids up about their behaviours) and realises that milk is low in the fridge.

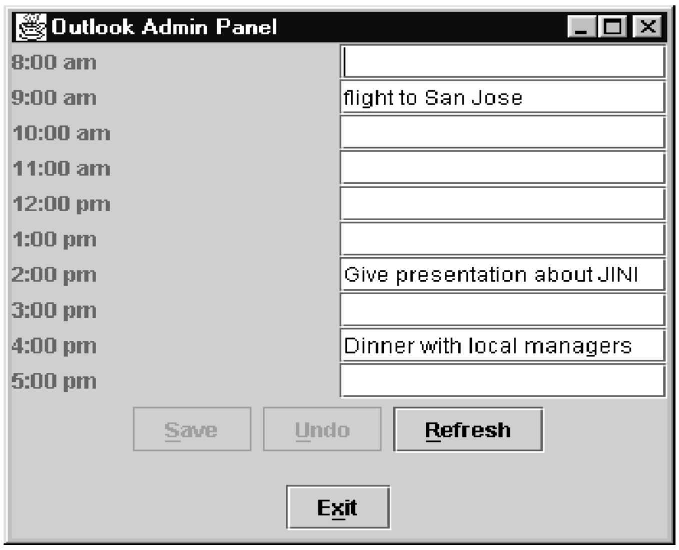
- With the milk level low, Jeannie orders milk from an *online grocer service* and completes one of her weekly chores via the PDA. (She usually does this from the control panel on the fridge. Note that there are already prototype refrigerators in the market place today that are equipped with cameras (e.g., one by Panasonic) to capture the contents of the fridge so that things like 'milk is low' can be detected.)
- When Jeannie gets to the airport, her PDA is recognised by the airline's *check-in service* and she is immediately prompted to check in. After pressing the 'check-in' button, she is offered a confirmation screen to check in. When she confirms, she is given terminal and gate information and the check-in process is echoed in the airline computer system.
- After handling Jeanne's check in, the check-in service goes down. A *second check-in service* automatically wakes up to take over the load of the first one.
- After a short flight, Jeanne gets to the destination. She suddenly realises she forgot to bring her presentation materials with her. So, she looks up a *printer service* and sends the materials remotely to the service for printing.
- After picking up the materials, Jeanne goes to the venue of the meeting. There is a *projector service* available in the presentation room and thus Jeanne simply uploads her presentation to the projector and gives a fantastic presentation on Jini distributed object technology.
- Before heading back home, Jeanne thinks about her kids and their love for video games. To celebrate her presentation job well done, she approaches a game vendor at the airport. The vendor supports *game service* and thus Jeanne simply selects a game of interest, pays for the game via credit card, and downloads the game onto her PDA.

Now, each of the scenarios above is detailed with respect to its user interface and functionality. Note that each of the scenarios, where a Jini service is showcased, has a client interface and a 'back-end' interface. The former is what a client such as a PDA downloads to its machine when interacting with the corresponding service, and the latter represents the service itself. Its purpose in this simulation is merely to echo or show the interactions behind the scene.

2.1 Calendar service

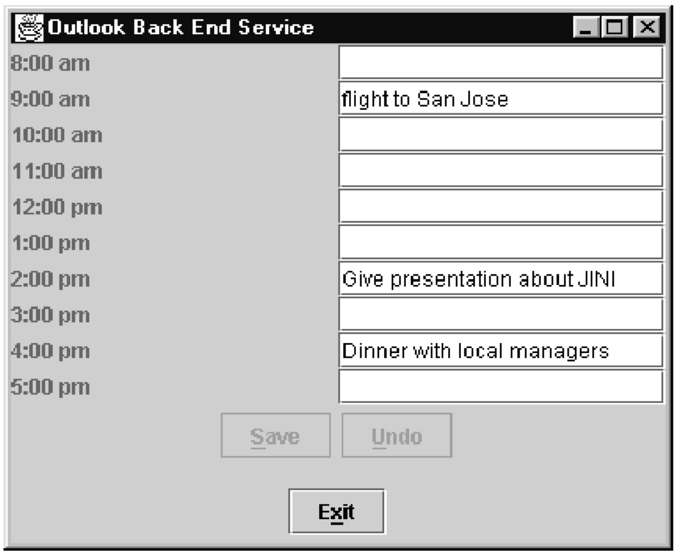
In Figures 1 and 2, the client and back-end interfaces to the calendar service are depicted, respectively. They both have a calendar-like look and feel to save and undo entries, and the client has an additional button to refresh the entries that may be entered in the back-end service. (Since the back-end represents the service itself, all changes made to the calendar are made in-place, and thus no refresh is needed.) Here, upon seeing the client panel, Jeannie sees that she has three items on her calendar for the day – fly to San Jose, make a presentation on Jini technology, and meet with the local managers.

Figure 1 Interface downloaded by client



8:00 am	
9:00 am	flight to San Jose
10:00 am	
11:00 am	
12:00 pm	
1:00 pm	
2:00 pm	Give presentation about JINI
3:00 pm	
4:00 pm	Dinner with local managers
5:00 pm	

Figure 2 Calendar service provider



8:00 am	
9:00 am	flight to San Jose
10:00 am	
11:00 am	
12:00 pm	
1:00 pm	
2:00 pm	Give presentation about JINI
3:00 pm	
4:00 pm	Dinner with local managers
5:00 pm	

2.2 Fridge service

In Figures 3 and 4, the client and back-end interfaces of the refrigerator service are depicted, respectively. Both interfaces have a slider to control the temperature of the fridge, an indicator to show the milk level, and a button to order (a preset level of) milk. The difference here is that the milk indicator on the client panel is not editable, whereas the one on the back-end is (this is to simulate the fact that as milk is added to the fridge, the level can be changed).

Figure 3 GUI control panel for the fridge client

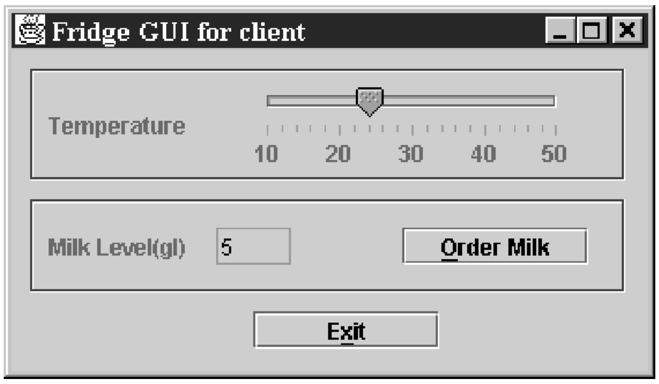
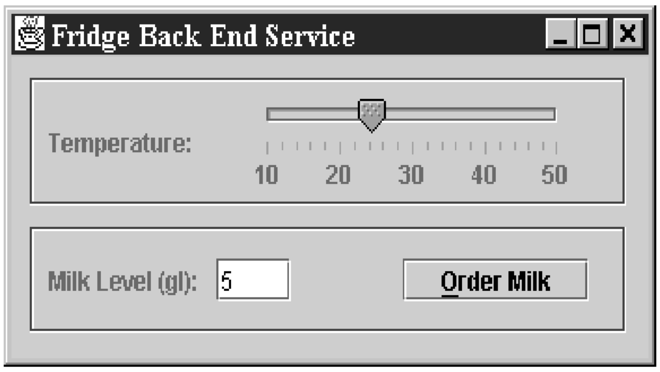


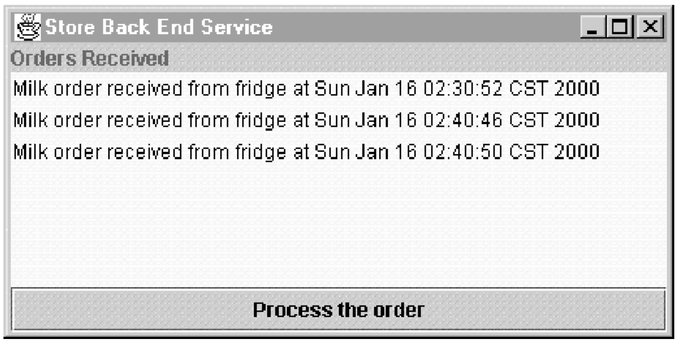
Figure 4 Fridge back-end (service provider)



2.3 *Online grocer service*

To simulate the online grocer store, a back-end service is created (Figure 5 below). Here, each time the Order Milk button is pressed (from either the GUI fridge client or the fridge back-end), the online grocer service echoes with a line on a terminal window to acknowledge receipt of the message.

Figure 5 Online store back-end (service provider)

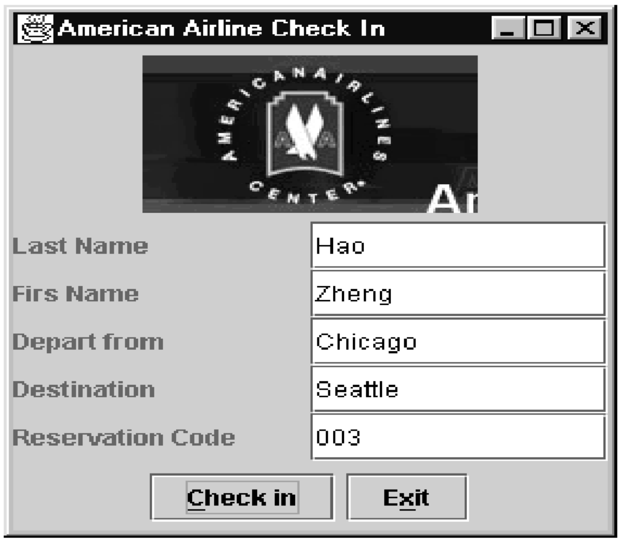


```
Milk order received from fridge at Sun Jan 16 02:30:52 CST 2000
Milk order received from fridge at Sun Jan 16 02:40:46 CST 2000
Milk order received from fridge at Sun Jan 16 02:40:50 CST 2000
```

2.4 Check-in service

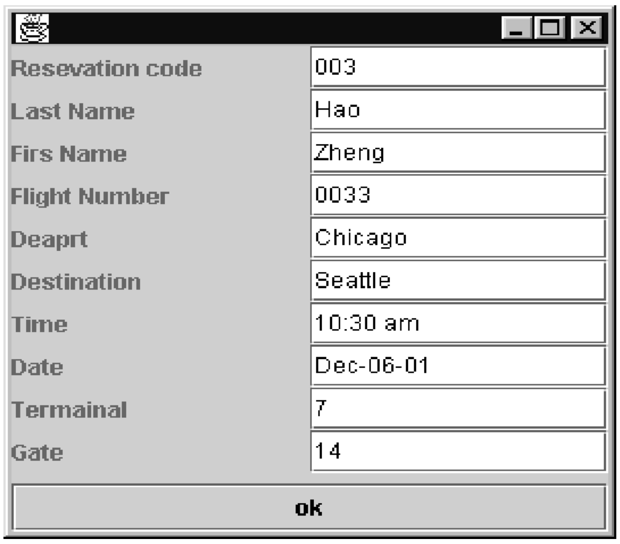
The most interesting and challenging service of this project is the check-in service. The client of the service is depicted in Figures 6 and 7. The former shows the passenger name, origin and destination of the flight, and the reservation code. Once the check-in button is pressed, the passenger is given a confirmation of the check in. The confirmation tells the passenger the terminal and gate information. The back-end check-in service is depicted in Figure 8. Like the online grocer backend, it merely logs and echoes the check-in transactions initiated by the clients.

Figure 6 Check-in service client



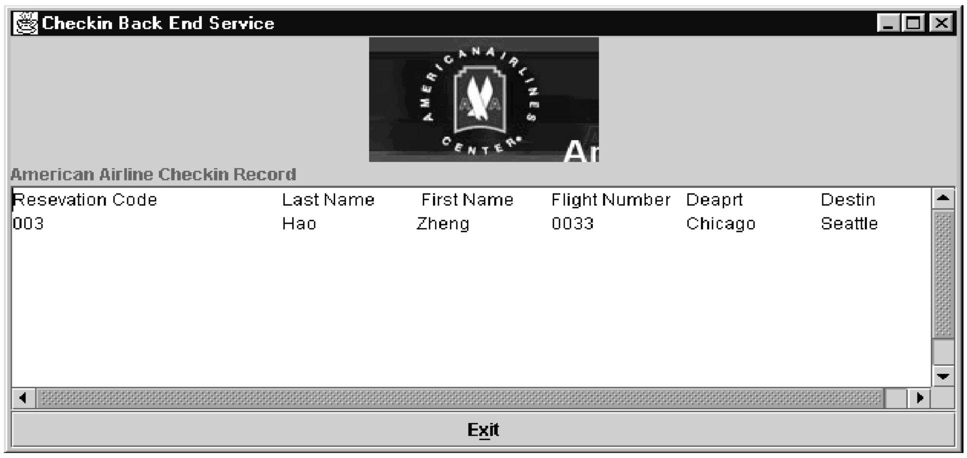
Last Name	Hao
Firs Name	Zheng
Depart from	Chicago
Destination	Seattle
Reservation Code	003

Figure 7 Check-in client – confirmation screen



Resevation code	003
Last Name	Hao
Firs Name	Zheng
Flight Number	0033
Deaprt	Chicago
Destination	Seattle
Time	10:30 am
Date	Dec-06-01
Termainal	7
Gate	14

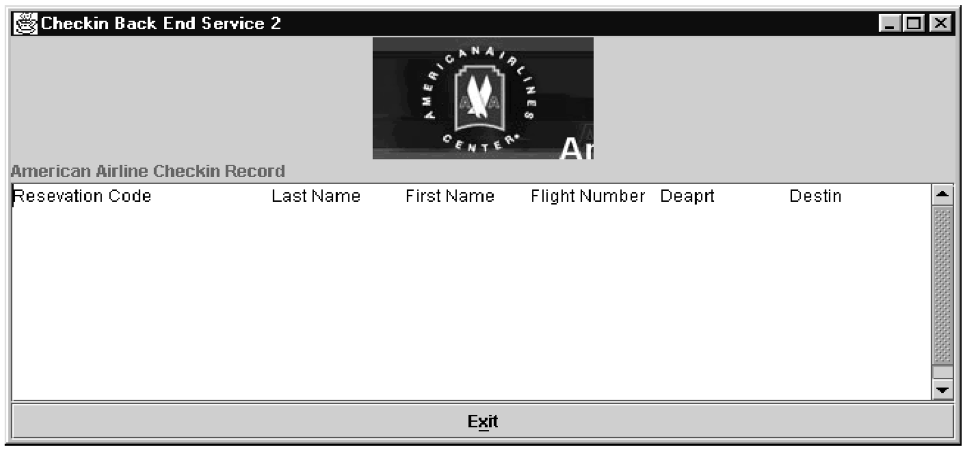
Figure 8 Check-in back-end service



Resevation Code	Last Name	First Name	Flight Number	Deaprt	Destin
003	Hao	Zheng	0033	Chicago	Seattle

When the first check-in back-end service is down (here, this is simulated by explicitly shutting the service down with the Exit button), the second check-in service is immediately started up automatically (Figure 9). It looks and behaves exactly the same as the first one, with the exception that it has the frame title 'Check-in Back-End Service 2'.

Figure 9 Backupcheck-in service

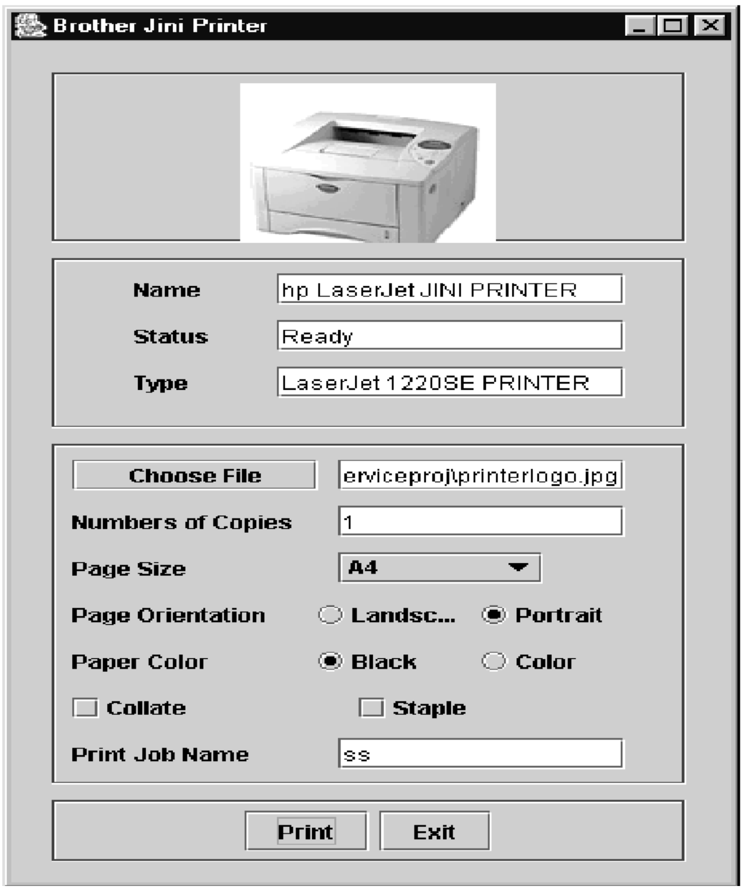


Resevation Code	Last Name	First Name	Flight Number	Deaprt	Destin

2.5 Printer service

Once a printer service is located (and its services and location are checked out), its client can be downloaded, as depicted in Figure 10 below. It offers the typical interface found in many printers. The back-end printer service is depicted in Figure 11. Like the online grocer back-end, it merely logs and echoes the printer jobs initiated by the clients.

Figure 10 Printing service client

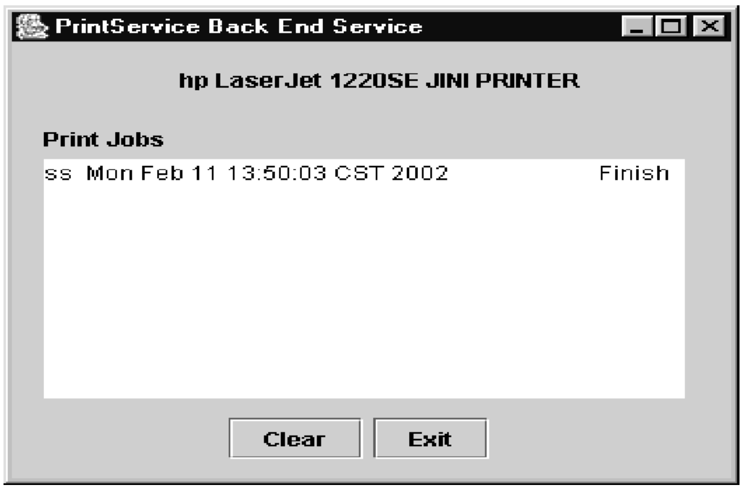


Brother Jini Printer

Name: hp LaserJet JINI PRINTER
Status: Ready
Type: LaserJet 1220SE PRINTER

Choose File: erviceproj\printerlogo.jpg
Numbers of Copies: 1
Page Size: A4
Page Orientation: ○ Landsc... ● Portrait
Paper Color: ● Black ○ Color
☐ Collate ☐ Staple
Print Job Name: ss

Figure 11 Back-end panel of the printing service



PrintService Back End Service

hp LaserJet 1220SE JINI PRINTER

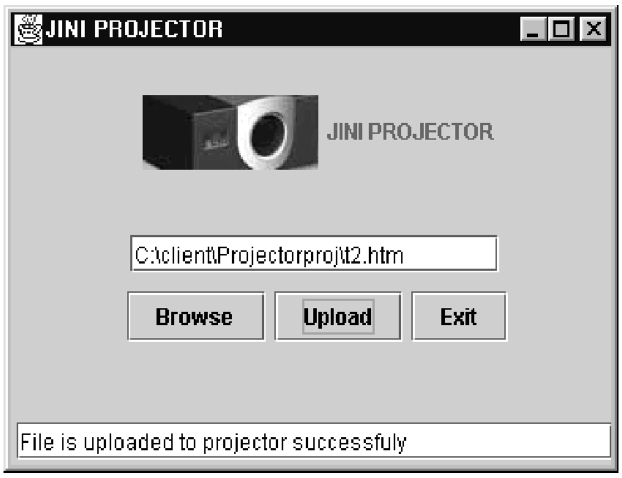
Print Jobs

```
ss  Mon Feb 11 13:50:03 CST 2002            Finish
```

2.6 Projector service

In Figures 12 and 13, the client and back-end interfaces of the projector service are depicted, respectively. The client allows the user to browse the local device to pick a file for upload to the projector and the back-end logs the file transfer.

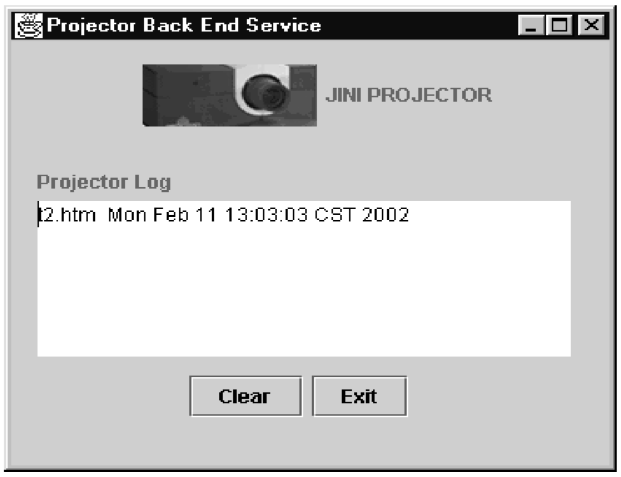
Figure 12 Projector service client



C:\client\Projectorproj\t2.htm

File is uploaded to projector successfuly

Figure 13 Projector back-end service

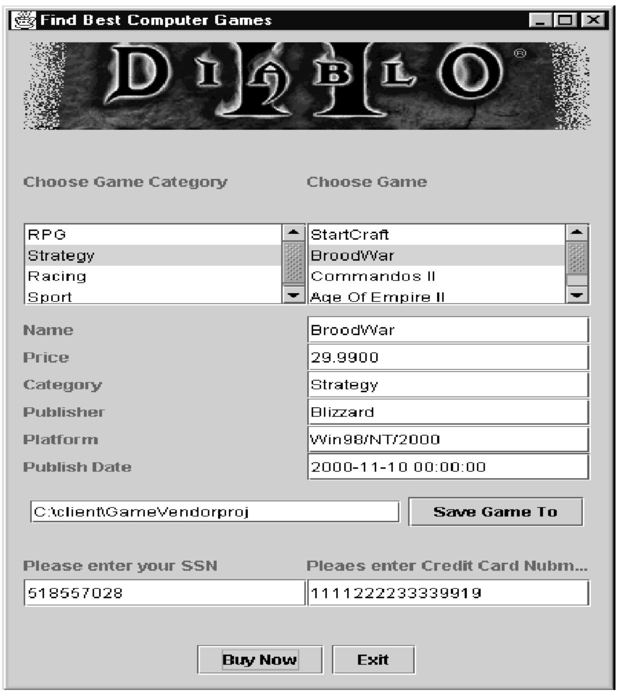


```
t2.htm  Mon Feb 11 13:03:03 CST 2002
```

2.7 Gaming service

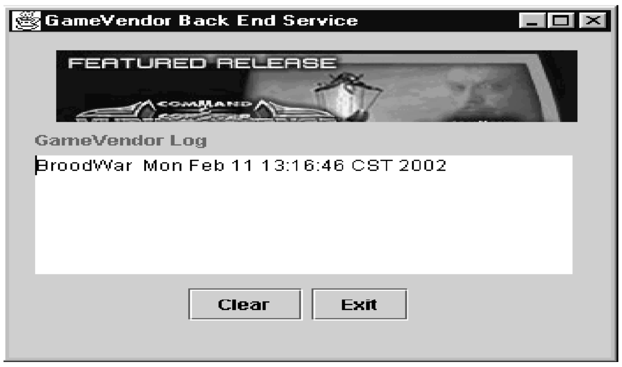
In Figures 14 and 15, the client and back-end interfaces of the gaming service are depicted, respectively. As can be seen, the client allows the user to browse the categories of the games to narrow down the search. Once a game title is selected, the details of the game are shown. Once picked, the game can be downloaded to a chosen directory. Of course, credit card information must also be provided. Like the other back-ends, this one simply logs the game purchase.

Figure 14 Game service client



Choose Game Category	Choose Game
RPG	StartCraft
Strategy	BroodWar
Racing	Commandos II
Sport	Age Of Empire II
Name	BroodWar
Price	29.9900
Category	Strategy
Publisher	Blizzard
Platform	Win98/NT/2000
Publish Date	2000-11-10 00:00:00
C:\client\GameVendorproj	
Please enter your SSN	Pleaes enter Credit Card Nubm...
518557028	1111222233339919

Figure 15 Game service back-end



```
BroodWar  Mon Feb 11 13:16:46 CST 2002
```

3 What's under the hood

As given earlier, this Genie project was implemented in Jini, an ambitious project at Sun Microsystems to build a distributed computing infrastructure or framework that is reliable, maintainable, scalable, evolvable, and spontaneous. This framework is built atop the Java distributed computing mechanisms of sockets and Remote Method Invocation (RMI), which in turn uses whatever network protocol is supported by the operating system. Jini technology does not require any particular operating system or network transport: a printer may use infrared or radio or be physically plugged into a local network. Many of the emerging network technology specifications are designed for specific types of networks such as TCP/IP for Ethernet, IrDA for infrared, IEEE 1394 (Firewire), or wireless; all of these are supported by Jini.

The architecture and components of Jini are briefly described below.

3.1 Architecture and components of Jini

Jini architecture is made up of three major components:

- one or more services that implement a well-known interface
- one or more clients that use the service(s)
- one or more lookup services that act as an agent between the service(s) and client(s).

To join a Jini community, a service needs to register itself with a lookup service. Usually, the service will publish its service proxy object, which is referred to as a *ServiceItem,* on the lookup service chosen. It is not uncommon that a service also publishes several *Entry* attributes, which describe the service itself. For instance, a printer may tell its location, brand, and model, whether it supports colour printing, etc. The user interface provided by the service provider may also be published as an Entry attribute.

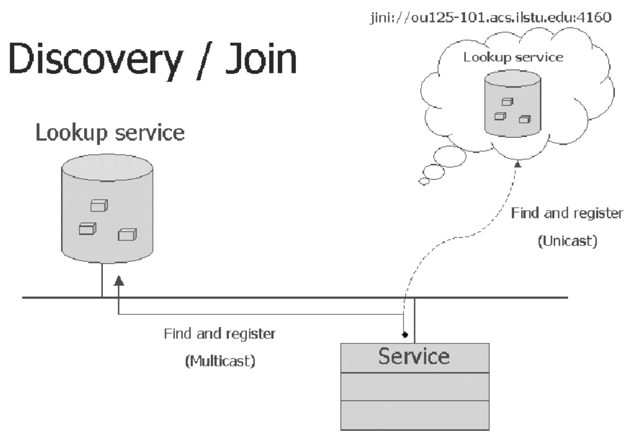
To find a service, a client such as a PDA, PC, or digital camera needs to discover a lookup service first. Once a lookup service is found, a client can form *ServiceTemplates*, which describe the interface that a target service implements and Entry attributes that the target service has, and search the lookup service using the ServiceTemplates. The lookup service will return the service proxy objects (ServiceItem) that match the ServiceTemplates defined by the client. After receiving the service proxy objects, the client is then able to use them to make a remote service call (e.g., a digital camera can send a picture to a colour printer for printing). The protocols under the hood are TCP/IP and RMI.

The discovery and lookup processes, together with several other important concepts, namely Leasing, Remote Event and Transaction, are further described below.

3.2 Discovery/Join

Discovery is the process by which Jini applications find the lookup services that serve their communities. There are two basic forms of discovery. One form is called *multicast discovery*. It is used to support serendipitous interaction between services and lookup services. Serendipitous interaction means that the lookup services and Jini services find each other without any previous configuration or advance knowledge. This form of discovery needs the lookup service and the services it serves to be on the same subnet. When a new lookup service joins the Jini community, it uses multicast announcement to broadcast its presence. The second form is called *unicast discovery*. It is to ‘hard wire’ a Jini service to a lookup service. This direct form of discovery allows services to contact a particular lookup service that they may know about ahead of time. This form of discovery has a URL-based naming scheme for specifying lookup services, e.g., jini://ou125-101.acs.ilstu.edu:4160 (Figure 16). Once a service discovers the lookup service, it registers itself by storing a proxy object and the attributes in the lookup service. This project uses the multicast discovery protocol; thus, all the services need to be on the same subnet.

Figure 16 Discovery/Join Protocol

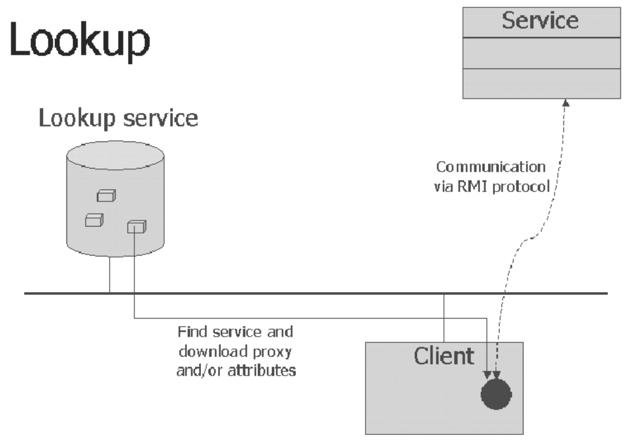


3.3 *Lookup*

Lookup provides the directory service for the Jini community, handles how services join the community, and manages how clients search, find, and use the services. As mentioned, after finding a lookup service, a service publishes a downloadable proxy object called ServiceItem that implements a well-known interface on the lookup service. This proxy object is copied to a client's machine when a client wants to use the service. With the proxy, the client could then use it to communicate with the service.

The lookup service is the key to scalability of the Jini community. Different Jini communities can be 'linked' together through lookup services to form a federated community. This process is called *federation*. Since a lookup service itself is a Jini service, it can register itself with the lookup services in other Jini communities. With such a registration, a lookup service can keep track of all the services for a Jini community and through federation, the services for a given community can also be searched and used by other Jini communities.

Figure 17 Lookup Protocol



3.4 Leasing (self-healing)

In a distributed environment, components of the environment could cause many failures, especially partial failure. Unlike local computing, once a component in a distributed system fails, there is no central unit that can inform other components of the system about the failure. Other components may continue to operate without knowing part of the system has already failed. For instance, if a service crashes for some reason, it has no way of telling the lookup service and deregistering itself. The clients can still find the service proxy on the lookup service, but they simply cannot use the service because of its failure. Another problem is that the lookup service will still have to waste resources to keep the service proxy active because it does not know that the service is already dead. Eventually, human administration has to be involved to clean up the inactive service registrations, which obviously violates the 'configuration-free' goal of Jini.

Jini uses a mechanism known as *leasing* to promote self-healing networks and provide maintainability in a long-running distributed system. It recognises that in a distributed system, the challenge is to design a system that realises the inevitability of failure and can clean up from failure, i.e., a distributed system that can heal itself. Through leasing, a resource is leased to consumers (clients or services) rather than granted to them in perpetuity. When the consumers want a resource, they need to request a lease, the grantor (usually a lookup service) may deny or grant lease, and the grantee needs to renew the lease before expiration. For example, in the aforementioned check-in service, when the first service registers itself, it gets a lease from the lookup service. When the service was terminated, it failed to renew the lease before the lease expired. The lookup service eventually detects this, cleans up the resource, and signals the event, so that the second check-in can be started (see Remote Event below).

3.5 Remote event

Jini's remote event is used by services to notify asynchronously the interested parties of changes in their states. For example, a digital camera wishing to print a picture but finding no printing service available can register a remote listener on the lookup service. It will be notified by the lookup service when a printer becomes available. Furthermore, the printer can use a remote event to notify the client when a print job is finished or has failed or when the state of the printer changes. The remote event can also provide the fail-over property of service, which is especially important to some critical services such as call centres, airline check in, etc. When a particular service is down, a duplicate service can be notified to take over the job.

3.6 Transaction

To guarantee that all Jini components are in a safe state, the concept of transaction is supported. This technique, found in most database systems, guarantees that a computation either finishes completely or no partial result is registered. Jini uses a two-phase commit mechanism to guarantee the transaction feature.

3.7 Other required services

To make a Jini service run in the Jini community, the following services are also required besides a lookup service and the service provider:

- The RMI activation daemon, which manages the transition between active and inactive states of the objects, must be run on each host that is running a lookup service.
- A web server that (a) provides the system reggie.jar file for download by the hosts that run lookup services, (b) supplies the service proxies and other required objects or stubs for download by the lookup services or clients, and (c) offers the required code for a lookup service to monitor a service of interest by the clients. For example, a client wishing to monitor a specific service can provide a listener to the lookup service so that it can be notified whenever the status of the service alters.

This section gives a broad overview of the development of the Genie project. The reader is referred to [12] for more details on the design and implementation of the system.

4 Jini and related technologies

As a distributed infrastructure, Jini addresses many of the problems faced by various distributed system architectures such as Microsoft's Distributed Component Object Model (DCOM), OMG's Common Object Request Broker Architecture (CORBA) and Java's RMI. The common issues and how the issues are addressed by Jini are presented below.

Table 1 How Jini handles some common problems in distributed systems

Common issues	*How Jini addresses the issues*
Network latency	Differentiate local and remote objects/components through Java remote interface
Partial failure	Objects need to express interest regularly through leasing mechanism
Concurrency	Provides transaction service through transaction manager interface
Tight coupling	Dynamic downloadable objects
Notification	Provides a distributed event mechanism through the EJB model
Address space	Differentiate local and remote address space
Dependence on specific protocol	Service can be implemented by any protocol: JRMP, IIOP, ORPC, SOAP, or proprietary protocol

Besides Jini, several other technologies are trying to address the same (or closely related) problem domain. They include Microsoft's Universal Plug and Play (UPnP) [13] and Hewlett-Packard's JetSend, Chai, e-Speak, and CoolTown [14], with CoolTown serving as the umbrella for the other HP technologies. Among them, UPnP and ChaiPnP are addressing the distributed device computing space, and JetSend and e-Speak are focusing on distributed service computing. Only CoolTown, like Jini, is trying to address both device and service computing, meaning that both allow hardware, software, and appliance services to be modelled. Thus, CoolTown is the only true competing technology of Jini listed above. But given its recent entry into the space, it would be a while before the ‘cool town’ will accept residents.

Another (and certainly more burgeoning) incarnation of SOA is *web services*. Web services have been touted by many as the next big thing in information technology [15–18]. Like Jini, web services operate by having a service provider register its service, using Web Service Description Language (WSDL), with a directory service called UDDI (Universal Description, Discovery, and Integration), and a client of the service searching for the service through UDDI and invoking the service via the standard Simple Object Access Protocol (SOAP). However, the technologies that provide the implementation of web services are by their very core statically based. For example, imagine a web server or a UDDI server going down when a mission-critical message is being processed. A statically based technology does not account for such failure. In contrast, a dynamic-based technology such as Jini has the ability to self-heal and recover from failure. This is why application servers such as Macromedia’s JRun use Jini for their server technology.

It should be noted that one important difference between web services and Jini is that web services make use of HTTP as the transport protocol, whereas Jini uses RMI as the underlying mechanism for code moving. Using a firewall-friendly HTTP protocol that is also standard based is the main reason why web services have received so much attention lately. As elegant and innovative as Jini technology is, it will not gain wide acceptance and will only serve a niche market until it changes its model to accommodate standards. Also, web services cater more to B2B needs and will serve B2B markets in the future, whereas Jini focuses on the issues of ubiquitous computing and dynamic distributed community formation of services.

On the consumer electronics space, HAVi (Home Audio–Video interoperability) [19] is a specification for home networks of consumer electronics devices such as CD players, TVs, VCRs, and digital cameras, using IEEE 1394. The network configuration is automatically updated as devices are plugged in or removed. Applications using Jini connection software can gain access to HAVi devices such as VCRs. Likewise, home devices on the HAVi network, like a television, could connect to remote services enabled by Jini technology such as video-on-demand.

5 Summary and future challenges

Sun’s Jini technology provides end-to-end solutions for creating dynamically networked products, services, and applications that scale from devices to the enterprise. In this paper, we have briefly reviewed the Jini technology and described how it can be used to provide several potential network services such as remote printing and airline check in the Genie project. This project could be used as a pilot to explore more functionality and usage with Jini and SOA in general. Expanding the service and the functionality of Genie will be the subject of future work.

Based on the experience of this project, it is fair to state that there is tremendous potential for distributed object computing technology, especially in the form of internet appliances. It has been said that with Jini, the phrase ‘the network is the computer’ is now understood by Sun (as they did not know what it meant before). Many vendors such as GE, Sony, and Maytag are already experimenting with the notion of internet appliances, and it is reasonable to expect that consumers at large will be bombarded with smart internet appliances from these vendors and others in the near future, leading to ubiquitous computing [20]. The only question is, Will the current internet infrastructure hold up when millions of these devices are hooked up to the net? Imagine the number of IP addresses that would be needed to serve these devices. If one thinks WWW is World Wide '*Wait*', wait till one sees the explosion of internet appliances on the web if the internet infrastructure is not fundamentally changed!

The infrastructure issue is just one of the challenges ahead for a technology like Jini. Even Sun acknowledges that Jini has been held back by technologies like Bluetooth and several crucial ingredients. Another problem is that small devices may not have enough resources to run Jini since its needs a Java virtual machine and Jini runtime. This will be a non-issue when more powerful devices are available. Also, as a result of recent technology breakthrough, a microversion of Jini network technology can now be used to address a number of current challenges and gaps faced by the mobile industry.

Lastly, like many other internet technologies, security is another issue needed to be addressed. For instance, if an application is configured to discover the lookup service in the group 'sun', how does one guarantee that the lookup services discovered belongs to the 'sun' group? Part of this can be addressed using access control list in Java 2 security model, which Jini uses, but more needs to be done in this area (e.g., technologies such as code signing and codebase location specification).

References

1 Siau, K. and Shen, Z. (2003) 'Mobile communications and mobile services', *International Journal of Mobile Communications*, Vol. 1, Nos. 1–2, pp.3–14.

2 Varshney, U. (2003) 'Location management for wireless networks: issues and directions', *International Journal of Mobile Communications*, Vol. 1, Nos. 1–2, pp.91–118.

3 Sun's Jini official website (2003) *Jini Network Technology*, http://www.sun.com/jini.

4 Waldo, J. (1999) 'The Jini architecture for network-centric computing', *Communications of the ACM*, Vol. 42, No. 7, July.

5 Waldo, J. (2000) 'Alive and well: Jini technology today', *IEEE Computer*, Vol. 33, No. 6, June.

6 Edwards, W.K. (2001) *Core Jini*, 2nd ed., Prentice Hall.

7 Kumaran, S.I. (2002) *Jini Technology Overview*, Prentice Hall.

8 Pierce, B. (2001) 'Build service based architectures with Jini', *Dr. Dobb's Journal*, June.

9 Bloomberg, J. (2003) 'Principles of SOA', *Application Development Trends*, Vol. 10, No. 3, March, pp.22–26.

10 Jini success stories (2003) *Jini Network Technology*, http://wwws.sun.com/software/jini/news/success.html.

11 Lim, B. and Duan, G. (2001) 'On simulating internet appliances using the Jini object architecture', *WebNet Journal: Internet Technologies, Applications, and Issues*, Vol. 3, No. 1.

12 Zheng, H. (2002) 'The design and implementation of genie: a self-healing system of internet appliances', *M.S. Project*, Applied Computer Science Department.

13 *Universal Plug-N-Play Forum*, http://upnp.com/

14 *CoolTown Research Homepage* (2003) http://cooltown.hp.com/research/

15 Dyck, T. (2001) 'Web services wave' (the cover story: web services wake-up call), *eWeek*, Vol. 18, No. 35, September.

16 Booch, G. (2001) 'Web services: the economic argument', *Software Development*, Vol. 9, No. 11, November.

17 Curbera, F. *et al*. (2002) 'Unraveling the web services web', *IEEE Internet Computing*, March/April.

18 Ferris, C. and Farrell, J. (2003) 'What are web services?', *Communications of the ACM*, Vol. 46, No. 6, June, pp.31–34.

19 *Home Audio–Video Interoperability Homepage* (2003) http://www.havi.org/

20 Banavar, G. and Bernstein, A. (2002) 'Issues and challenges in ubiquitous computing: software infrastructure and design challenges for ubiquitous computing applications', *Communications of the ACM, Volume 45*, Vol. 12, December.