Jini™ Architecture Specification

The Jini™ Architecture Specification defines the top-level view of the Jini architecture, its components, and the systems on which the Jini architecture is layered. This will give you a high-level view of the architecture that will be filled out in the following specifications.

Version 1.1
October 2000

Contents

AR

The Jini™ Architecture Specification

AR.1 Introduction

THIS document describes the high-level architecture of system of Jini™ technology-enabled services and/or devices (Jini system), defines the different components that make up the system, characterizes the use of those components, discusses some of the component interactions, and gives an example. This document identifies those parts of the system that are necessary infrastructure, those that are part of the programming model, and those that are optional services that can live within the system.

AR.1.1 Goals of the System

A Jini system is a distributed system based on the idea of federating groups of users and the resources required by those users. The overall goal is to turn the network into a flexible, easily administered tool with which resources can be found by human and computational clients. Resources can be implemented as either hardware devices, software programs, or a combination of the two. The focus of the system is to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.

A Jini system consists of the following parts:

- A set of components that provides an infrastructure for federating services in a distributed system

- A programming model that supports and encourages the production of reliable distributed services
- Services that can be made part of a federated Jini system and that offer functionality to any other member of the federation

Although these pieces are separable and distinct, they are interrelated, which can blur the distinction in practice. The components that make up the Jini technology infrastructure make use of the Jini technology programming model; services that reside within the infrastructure also use that model; and the programming model is well supported by components in the infrastructure.

The end goals of the system span a number of different audiences; these goals include the following:

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

The Jini system extends the Java™ application environment from a single virtual machine to a network of machines. The Java application environment provides a good computing platform for distributed computing because both code and data can move from machine to machine. The environment has built-in security that allows the confidence to run code downloaded from another machine. Strong typing in the Java application environment enables identifying the class of an object to be run on a virtual machine even when the object did not originate on that machine. The result is a system in which the network supports a fluid configuration of objects that can move from place to place as needed and can call any part of the network to perform operations.

The Jini architecture exploits these characteristics of the Java application environment to simplify the construction of a distributed system. The Jini architecture adds mechanisms that allow fluidity of all components in a distributed system, extending the easy movement of objects to the entire networked system.

The Jini technology infrastructure provides mechanisms for devices, services, and users to join and detach from a network. Joining and leaving a Jini system are easy and natural, often automatic, occurrences. Jini systems are far more dynamic than is currently possible in networked groups where configuring a network is a centralized function done by hand.

AR.1.2 Environmental Assumptions

The Jini system federates computers and computing devices into what appears to the user as a single system. It relies on the existence of a network of reasonable speed connecting those computers and devices. Some devices require much higher bandwidth and others can do with much less—displays and printers are examples of extreme points. We assume that the latency of the network is reasonable.

We assume that each Jini technology-enabled device has some memory and processing power. Devices without processing power or memory may be connected to a Jini system, but those devices are controlled by another piece of hardware and/or software that presents the device to the Jini system and itself contains both processing power and memory. Architectures for devices not equipped with a Java virtual machine[1] (JVM) are explored more fully in the *Jini™ Device Architecture Specification.*

The Jini technology infrastructure is Java technology centered. The Jini architecture gains much of its simplicity from assuming that the Java programming language is the implementation language for components. The ability to dynamically download and run code is central to a number of the features of the Jini architecture. However, the Java technology-centered nature of the Jini architecture depends on the Java application environment rather than on the Java programming language. Any programming language can be supported by a Jini system if it has a compiler that produces compliant bytecodes for the Java programming language.

[1] *As used in this document, the terms “Java virtual machine” or “JVM” mean a virtual machine for the Java platform.

AR.2 System Overview

AR.2.1 Key Concepts

THE purpose of the Jini architecture is to *federate* groups of devices and software components into a single, dynamic distributed system. The resulting federation provides the simplicity of access, ease of administration, and support for sharing that are provided by a large monolithic system while retaining the flexibility, uniform response, and control provided by a personal computer or workstation.

The architecture of a single Jini system is targeted to the workgroup. Members of the federation are assumed to agree on basic notions of trust, administration, identification, and policy. It is possible to federate Jini systems themselves for larger organizations.

AR.2.1.1 Services

The most important concept within the Jini architecture is that of a *service*. A service is an entity that can be used by a person, a program, or another service. A service may be a computation, storage, a communication channel to another user, a software filter, a hardware device, or another user. Two examples of services are printing a document and translating from one word-processor format to some other.

Members of a Jini system federate to share access to services. A Jini system should not be thought of as sets of clients and servers, users and programs, or even programs and files. Instead, a Jini system consists of services that can be collected together for the performance of a particular task. Services may make use of other services, and a client of one service may itself be a service with clients of its own. The dynamic nature of a Jini system enables services to be added or withdrawn from a federation at any time according to demand, need, or the changing requirements of the workgroup using the system.

Jini systems provide mechanisms for service construction, lookup, communication, and use in a distributed system. Examples of services include: devices such as printers, displays, or disks; software such as applications or utilities; information such as databases and files; and users of the system.

Services in a Jini system communicate with each other by using a *service protocol*, which is a set of interfaces written in the Java programming language. The set of such protocols is open ended. The base Jini system defines a small number of such protocols that define critical service interactions.

AR.2.1.2 Lookup Service

Services are found and resolved by a *lookup service*. The lookup service is the central bootstrapping mechanism for the system and provides the major point of contact between the system and users of the system. In precise terms, a lookup service maps interfaces indicating the functionality provided by a service to sets of objects that implement the service. In addition, descriptive entries associated with a service allow more fine-grained selection of services based on properties understandable to people.

Objects in a lookup service may include other lookup services; this provides hierarchical lookup. Further, a lookup service may contain objects that encapsulate other naming or directory services, providing a way for bridges to be built between a Jini lookup service and other forms of lookup service. Of course, references to a Jini lookup service may be placed in these other naming and directory services, providing a means for clients of those services to gain access to a Jini system.

A service is added to a lookup service by a pair of protocols called *discovery* and *join*—first the service locates an appropriate lookup service (by using the *discovery* protocol), and then it joins it (by using the *join* protocol).

AR.2.1.3 Java Remote Method Invocation (RMI)

Communication between services can be accomplished using *Java Remote Method Invocation* (RMI). The infrastructure to support communication between services is not itself a service that is discovered and used but is, rather, a part of the Jini technology infrastructure. RMI provides mechanisms to find, activate, and garbage collect object groups.

Fundamentally, RMI is a Java programming language-enabled extension to traditional remote procedure call mechanisms. RMI allows not only data to be passed from object to object around the network but full objects, including code. Much of the simplicity of the Jini system is enabled by this ability to move code around the network in a form that is encapsulated as an object.

AR.2.1.4 Security

The design of the security model for Jini technology is built on the twin notions of a *principal* and an *access control list*. Jini services are accessed on behalf of some entity—the principal—which generally traces back to a particular user of the system. Services themselves may request access to other services based on the identity of the object that implements the service. Whether access to a service is allowed depends on the contents of an access control list that is associated with the object.

AR.2.1.5 Leasing

Access to many of the services in the Jini system environment is *lease* based. A lease is a grant of guaranteed access over a time period. Each lease is negotiated between the user of the service and the provider of the service as part of the service protocol: A service is requested for some period; access is granted for some period, presumably taking the request period into account. If a lease is not renewed before it is freed—either because the resource is no longer needed, the client or network fails, or the lease is not permitted to be renewed—then both the user and the provider of the resource may conclude that the resource can be freed.

Leases are either exclusive or non-exclusive. Exclusive leases ensure that no one else may take a lease on the resource during the period of the lease; non-exclusive leases allow multiple users to share a resource.

AR.2.1.6 Transactions

A series of operations, either within a single service or spanning multiple services, can be wrapped in a *transaction*. The Jini transaction interfaces supply a service protocol needed to coordinate a *two-phase commit*. How transactions are implemented—and indeed, the very semantics of the notion of a transaction—is left up to the service using those interfaces.

AR.2.1.7 Events

The Jini architecture supports distributed *events*. An object may allow other objects to register interest in events in the object and receive a notification of the occurrence of such an event. This enables distributed event-based programs to be written with a variety of reliability and scalability guarantees.

AR.2.2 Component Overview

The components of the Jini system can be segmented into three categories: *infrastructure*, *programming model*, and *services*. The infrastructure is the set of components that enables building a federated Jini system, while the services are the entities within the federation. The programming model is a set of interfaces that enables the construction of reliable services, including those that are part of the infrastructure and those that join into the federation.

These three categories, though distinct and separable, are entangled to such an extent that the distinction between them can seem blurred. Moreover, it is possible to build systems that have some of the functionality of the Jini system with variants on the categories or without all three of them. But a Jini system gains its full power because it is a *system* built with the particular infrastructure and programming models described, based on the notion of a service. Decoupling the segments within the architecture allows legacy code to be changed minimally to take part in a Jini system. Nevertheless, the full power of a Jini system will be available only to new services that are constructed using the integrated model.

A Jini system can be seen as a network extension of the infrastructure, programming model, and services that made Java technology successful in the single-machine case. These categories along with the corresponding components in the familiar Java application environment are shown in Figure AR.2.1:

	Infrastructure	**Programming Model**	**Services**
Base Java	Java VM RMI Java Security	Java APIs JavaBeans …	JNDI Enterprise Beans JTS …
Java + Jini	Discovery/Join Distributed Security Lookup	Leasing Transactions Events	Printing Transaction Manager JavaSpaces Service …

FIGURE AR.2.1: ***Jini Architecture Segmentation***

AR.2.2.1 Infrastructure

The Jini technology infrastructure defines the minimal Jini technology core. The infrastructure includes the following:

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

The discovery and join protocols define the way a service of any kind becomes part of a Jini system; RMI defines the base language within which the Jini technology-enabled services communicate; the distributed security model and its implementation define how entities are identified and how they get the rights to perform actions on their own behalf and on the behalf of others; and the lookup service reflects the current members of the federation and acts as the central marketplace for offering and finding services by members of the federation.

AR.2.2.2 Programming Model

The infrastructure both enables the programming model and makes use of it. Entries in the lookup service are leased, allowing the lookup service to reflect accurately the set of currently available services. When services join or leave a lookup service, events are signaled, and objects that have registered interest in such events get notifications when new services become available or old services cease to be active. The programming model rests on the ability to move code, which is supported by the base infrastructure.

Both the infrastructure and the services that use that infrastructure are computational entities that exist in the physical environment of the Jini system. However, services also constitute a set of interfaces which define communication protocols that can be used by the services and the infrastructure to communicate between themselves.

These interfaces, taken together, make up the distributed extension of the standard Java programming language model that constitutes the Jini programming model. Among the interfaces that make up the Jini programming model are the following:

- The leasing interface, which defines a way of allocating and freeing resources using a renewable, duration-based model
- The event and notification interfaces, which are an extension of the event model used by JavaBeans™ components to the distributed environment, enable event-based communication between Jini technology-enabled services
- The transaction interfaces, which enable entities to cooperate in such a way that either all of the changes made to the group occur atomically or none of them occur

The lease interface extends the Java programming language model by adding time to the notion of holding a reference to a resource, enabling references to be reclaimed safely in the face of network failures.

The event and notification interfaces extend the standard event models used by JavaBeans components and the Java application environment to the distributed case, enabling events to be handled by third-party objects while making various delivery and timeliness guarantees. The model also recognizes that the delivery of a distributed notification may be delayed.

The transaction interfaces introduce a lightweight, object-oriented protocol enabling applications using Jini technology to coordinate state changes. The transaction protocol provides two steps to coordinate the actions of a group of distributed objects. The first step is called the *voting phase,* in which each object "votes" whether it has completed its portion of the task and is ready to commit any changes it made. In the second step, a coordinator issues a "commit" request to each object.

The Jini transaction protocol differs from most transaction interfaces in that it does not assume that the transactions occur in a transaction processing system. Such systems define mechanisms and programming requirements that guarantee the correct implementation of a particular transaction semantics. The Jini transaction protocol takes a more traditional object-oriented view, leaving the correct implementation of the desired transaction semantics up to the implementor of the particular objects that are involved in the transaction. The goal of the transaction protocol is to define the interactions that such objects must have to coordinate such groups of operations.

The interfaces that define the Jini programming model are used by the infrastructure components where appropriate and by the initial Jini technology-enabled services. For example, the lookup service makes use of the leasing and event interfaces. Leasing ensures that services registered continue to be available, and events help administrators discover problems and devices that need configuration. The JavaSpaces™ service, one example of a Jini technology-enabled service, utilizes leasing and events, and also supports the Jini transaction protocol. The transaction manager can be used to coordinate the voting phase of a transaction for those objects that support transaction protocol.

The implementation of a service is not required to use the Jini programming model, but such services need to use that model for their interaction with the Jini technology infrastructure. For example, every service interacts with the Jini lookup service by using the programming model; and whether a service offers resources on a leased basis or not, the service's registration with the lookup service will be leased and will need to be periodically renewed.

The binding of the programming model to the services and the infrastructure is what makes such a federation a Jini system not just a collection of services and protocols. The combination of infrastructure, service, and programming model, all designed to work together and constructed by using each other, simplifies the overall system and unifies it in a way that makes it easier to understand.

AR.2.2.3 Services

The Jini technology infrastructure and programming model are built to enable services to be offered and found in the network federation. These services make use of the infrastructure to make calls to each other, to discover each other, and to announce their presence to other services and users.

Services appear programmatically as objects written in the Java programming language, perhaps made up of other objects. A service has an interface that defines the operations that can be requested of that service. Some of these interfaces are intended to be used by programs, while others are intended to be run by the receiver so that the service can interact with a user. The type of the service determines the interfaces that make up that service and also define the set of methods that can be used to access the service. A single service may be implemented by using other services.

Example Jini technology-enabled services include the following:

- A printing service, which can print from applications written in the Java programming language as well as legacy applications
- A JavaSpaces service, which can be used for simple communication and for storage of related groups of objects written in the Java programming language

- A transaction manager, which enables groups of objects to participate in the Jini transaction protocol defined by the programming model

AR.2.3 Service Architecture

Services form the interactive basis for a Jini system, both at the programming and user interface levels. The details of the service architecture are best understood once the Jini discovery and Jini lookup protocols are presented.

AR.2.3.1 Discovery and Lookup Protocols

The heart of the Jini system is a trio of protocols called *discovery*, *join*, and *lookup*. A pair of these protocols—discovery and join—occur when a device is plugged in. Discovery occurs when a service is looking for a lookup service with which to register. Join occurs when a service has located a lookup service and wishes to join it. Lookup occurs when a client or user needs to locate and invoke a service described by its interface type (written in the Java programming language) and possibly other attributes. Figure AR.2.2 outlines the discovery process.

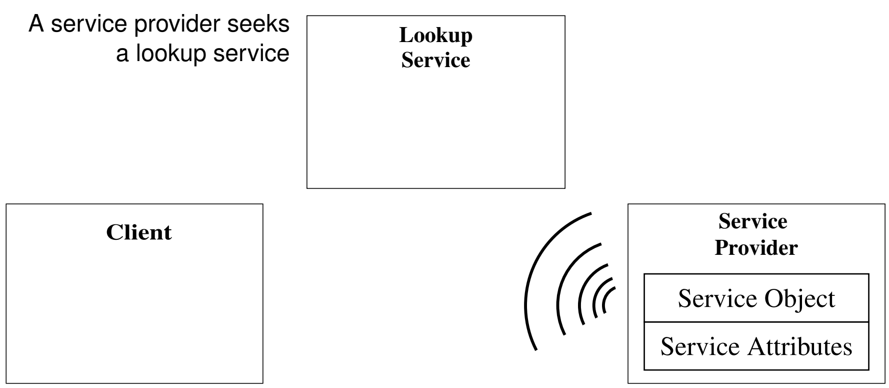

FIGURE AR.2.2: ***Discovery***

Jini discovery/join is the process of adding a service to a Jini system. A service provider is the originator of the service—a device or software, for example. First, the service provider locates a lookup service by multicasting a request on the local network for any lookup services to identify themselves (Figure AR.2.2).

Then, a service object for the service is loaded into the lookup service (Figure AR.2.3). This service object contains the Java programming language interface for the service, including the methods that users and applications will invoke to execute the service along with any other descriptive attributes.

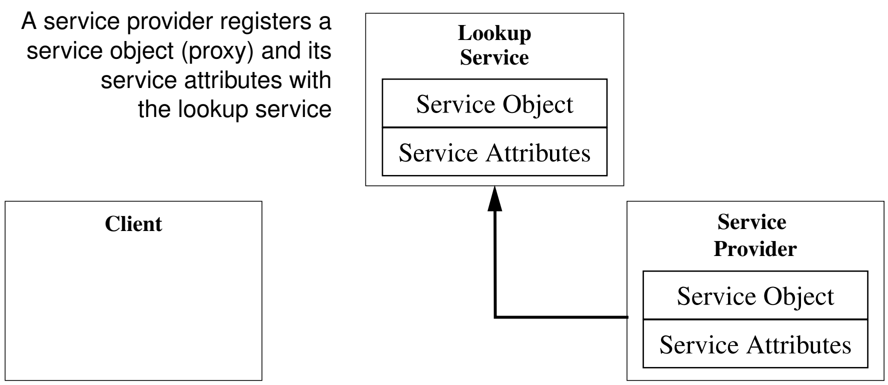

FIGURE AR.2.3: ***Join***

Services must be able to find a lookup service; however, a service may delegate the task of finding a lookup service to a third party. The service is now ready to be looked up and used, as shown in the following diagram (Figure AR.2.4).

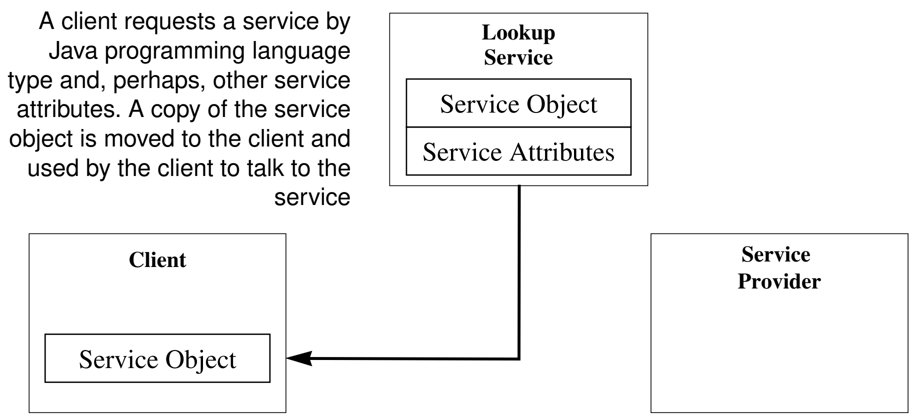

FIGURE AR.2.4: ***Lookup***

A client locates an appropriate service by its type—that is, by its interface written in the Java programming language—along with descriptive attributes that are used in a user interface for the lookup service. The service object is loaded into the client.

The final stage is to invoke the service, as shown in the following diagram (Figure AR.2.5).

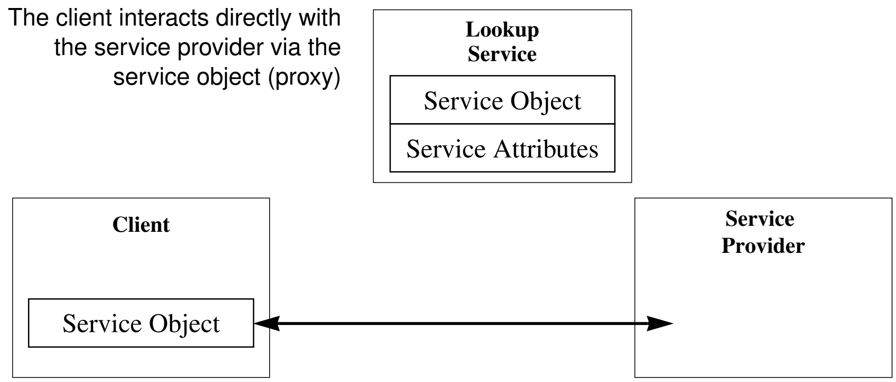

FIGURE AR.2.5: ***Client Uses Service***

The service object's methods may implement a private protocol between itself and the original service provider. Different implementations of the same service interface can use completely different interaction protocols.

The ability to move objects and code from the service provider to the lookup service and from there to the client of the service gives the service provider great freedom in the communication patterns between the service and its clients. This code movement also ensures that the service object held by the client and the service for which it is a proxy are always synchronized because the service object is supplied by the service itself. The client knows only that it is dealing with an implementation of an interface written in the Java programming language, so the code that implements the interface can do whatever is needed to provide the service. Because this code came originally from the service itself, the code can take advantage of implementation details of the service that are known only to the code.

The client interacts with a service via a set of interfaces written in the Java programming language. These interfaces define the set of methods that can be used to interact with the service. Programmatic interfaces are identified by the type system of the Java programming language, and services can be found in a lookup service by asking for those that support a particular interface. Finding a service this way ensures that the program looking for the service will know how to use that service, because that use is defined by the set of methods that are defined by the type.

Programmatic interfaces may be implemented either as RMI references to the remote object that implements the service, as a local computation that provides all of the service locally, or as some combination. Such combinations, called *smart proxies*, implement some of the functions of a service locally and the remainder through remote calls to a centralized implementation of the service.

A user interface can also be stored in the lookup service as an attribute of a registered service. A user interface stored in the lookup service by a Jini technology-enabled service is an implementation that allows the service to be directly manipulated by a user of the system.

In effect, a user interface for a service is a specialized form of the service interface that enables a program, such as a browser, to step out of the way and let the human user interact directly with a service.

In situations in which no lookup service can be found, a client could use a technique called *peer lookup* instead. In such situations, the client can send out the same identification packet that is used by a lookup service to request service providers to register. Service providers will then attempt to register with the client as though it were a lookup service. The client can select the services it needs from the registration requests it receives in response and drop or refuse the rest.

AR.2.3.2 Service Implementation

Objects that implement a service may be designed to run in a single address space with other, helper, objects especially when there are certain location or security-based requirements. Such objects make up an *object group*. An object group is guaranteed to always reside in a single address space or virtual machine when those objects are running. Objects that are not in the same object group are isolated from each other, typically by running them in a different virtual machine or address space.

A service may be implemented directly or indirectly by specialized hardware. Such devices can be contacted by the code associated with the interface for the service.

From the service client's point of view, there is no distinction between services that are implemented by objects on a different machine, services that are downloaded into the local address space, and services that are implemented in hardware. All of these services will appear to be available on the network, will appear to be objects written in the Java programming language, and, only as far as correct functioning is concerned, one kind of implementation could be replaced by another kind of implementation without change or knowledge by the client. (Note that security permissions must be properly granted.)

AR.3 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.

AR.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.

AR.3.1.1 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.

AR.3.1.2 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.

AR.3.1.3 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.

AR.3.1.4 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.

AR.3.2 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.

AR.3.2.1 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.

AR.3.2.2 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.

AR.3.2.3 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.

AR.3.2.4 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.

AR.3.2.5 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.

AR.3.2.6 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.