Jini™ Technology Glossary

Version 1.1
October 2000

Jini™ Technology Glossary

activation

The process of transforming a passive object into an active object. Activation requires that an object be associated with a Java™ virtual machine (JVM)[1], which may entail loading the class for that object into a JVM and the object restoring its persistent state (if any). (*Java™ Remote Method Invocation Specification,* Section 7.1.1)

activatable service

A Jini service that has a back end implemented with the ability to activate and deactivate. (*Introduction to Helper Utilities and Services*, Section US.2.9, "Activation") (See also *Jini technology-enabled service (Jini service)*)

activation descriptor

A class instance that holds an activatable object's group identifier (specifies the JVM in which it is activated), the object's class name, a location from where to load the object's class code, and object-specific initialization data in marshalled form. (*Java™ Remote Method Invocation Specification,* Section 7.2)

activation group

The entity that receives a request to activate an object in the JVM and returns the activated object back to the activator. (*Java™ Remote Method Invocation Specification,* Section 7.2) A separate JVM is spawned for each activation group. (Section 7.4.7)

[1] The terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

activator

The entity that supervises activation by being both (1) a database of information that maps activation identifiers to the information necessary to activate an object and (2) a manager of JVMs, that starts up a JVM (when necessary) and forwards requests for object activation (along with the necessary information) to the correct activation group inside a remote JVM. There is usually only one activator per host, started by `rmid`. (*Java™ Remote Method Invocation Specification,* Section 7.2)

active discard

The process of an entity itself discarding a reference to lookup service. (*Jini Discovery Utilities Specification,* Section DU.2.5.1, "The Semantics") (See also *passive discard*)

active object

A remote object that is instantiated and exported in a JVM on some system.(*Java™ Remote Method Invocation Specification,* Section 7.1.1)

ancestor transaction

A transaction that is the parent of a specific nested transaction (a transaction in which all its operations are contained, or executed, from within another transaction), or the parent of such a parent, recursively (a grandparent, a great-grandparent, and so on). (*The Jini Technology Core Platform Specification,* Section TX.3.5 "Semantics")

attribute set

A strongly-typed set of fields in a service item (represented by a `net.jini.core.entry.Entry`) that describes the service or provide secondary interfaces to the service. A single attribute is a public field of an `Entry`. (*The Jini Technology Core Platform Specification,* Section LU.1.2 "Attributes")

back end, back-end server

See *remote object.*

bad invocation exception

An exception that implies that any retries of the *same* method with the *same* arguments that are expected to return the *same* value will also fail. For a list of exceptions that fall into this category, please see *Introduction to Helper Utilities and Services,* Section US.2.6, "What Exceptions Imply about Future Behavior".

bad object exception

An exception that implies that any further operations on that object will also fail. For a list of exceptions that fall into this category, please see *Introduction to Helper Utilities and Services*, Section US.2.6, "What Exceptions Imply about Future Behavior".

changed event

An event related to the discarded event that notifies the entity of changes in the contents of the member groups of one or more of the lookup services in the managed set. (*Jini Discovery Utilities Specification*, Section DU.2.6.2, "The DiscoveryChangeListener Interface") (See also *discarded event*)

channel

The abstraction for a conduit between two address spaces in the RMI transport layer. As such, it is responsible for managing connections between the local address space and the remote address space for which it is a channel. (*Java™ Remote Method Invocation Specification*, Section 3.5)

client-like entity

A term that may be used, in general, when referring to Jini clients and Jini services that act as clients. (*Introduction to Helper Utilities and Services*, Section US.2.2, "Jini Clients and Services") (See also *Jini technology-enabled client (Jini client)*)

connection

The stream-oriented (*Java™ Remote Method Invocation Specification*, Section 3.4) abstraction for transferring data (performing input/output) in the RMI transport layer. (Section 3.5)

deactivation

The process of transforming an active object into a passive object. (*Introduction to Helper Utilities and Services*, Section US.2.9, "Activation") (See also *activation*)

definite exception

An exception that is indicative of a permanent failure. (*Jini Lease Utilities Specification*, Section LM.4, "The Semantics")

desired expiration

A value (in milliseconds) that represents when the client would like the lease to expire. (*Jini Lease Utilities Specification,* Section LM.4, "The Semantics") (See also *remaining desired duration*)

desired expiration reached event

An event that is generated when a lease's desired expiration is reached, which signals that the lease renewal manager has removed a lease from the managed set without an explicit request by the client. (*Jini Lease Utilities Specification,* Section LM.4, "The Semantics") (See also *renewal failure event*)

discarded lookup service

A lookup service is said to be discarded when an already discovered lookup service is removed from the managed set of lookup services. (*Introduction to Helper Utilities and Services*, Section US.2.8, "Discarding a Lookup Service") (See also *discarded event*)

discarded event

An event sent to notify all of an entity's discovery listeners, whenever a lookup service is discarded by a utility employed by the entity. (*Introduction to Helper Utilities and Services,* Section US.2.8, "Discarding a Lookup Service") (See also *changed event*) (See also *active discard*) (See also *discovered event*)

discovered event

An event sent from a lookup discovery service to a registered listener upon discovery of a lookup service. (*Jini Lookup Discovery Service) (See also discarded event*)

discovering entity

One or more cooperating objects in the Java programming language on the same host that are about to start, or are in the process of, obtaining references to one or more Jini lookup services. (*The Jini Technology Core Platform Specification,* Section DJ.1.1 "Terminology") (See also *group discovery*)

discovery utility

An object that implements one or more of the discovery management interfaces to perform and manage the entity's discovery duties. (*Jini Discovery*

Utilities Specification, Section DU.2.6.1, "The DiscoveryListener Interface")

distributed event adapter

An event adapter in which the event generator and the event listener instances may exist in different virtual machines, possibly on different hosts. The distributed event adapter is at least a remote event listener, but may also be a remote event generator (see *local event, remote event*). (*The Jini Technology Core Platform Specification,* Section EV.3 "Third-Party Objects")

djinn *(pronounced "gin")*

The group of devices, resources, and users joined by the Jini technology infrastructure. (*The Jini Technology Core Platform Specification,* Section LU.1.1 "The Lookup Service Model") This group, controlled by the Jini technology infrastructure, agrees on basic notions of trust, administration, identification, and policy.

dynamic class loading

The capability of the Java application environment to download files (classes for the Java platform, audio, and images) from an HTTP server at runtime if they are not already available to the client JVM. Dynamic class loading may be used by the RMI runtime to download: stub classes; skeleton classes; classes that are passed as subtypes of declared method parameters; and classes that are passed as subtypes of declared method return types. (See *dynamic stub loading*)

dynamic stub loading

A subset of dynamic class loading, used to support client-side stubs that implement the same set of remote interfaces as a remote object itself. (*Java™ Remote Method Invocation Specification,* Section 3.1)

endpoint

The abstraction used to denote an address space or JVM in the RMI transport layer. In the implementation an endpoint can be mapped to its transport. That is, given an endpoint, a specific transport instance can be obtained. (*Java™ Remote Method Invocation Specification,* Section 3.5)

entity

A general term that may refer to a discovering entity, a joining entity, a client-like entity, a service, or some combination of these types of entities. It

should be clear from the context what type of entity is being discussed. (*Introduction to Helper Utilities and Services*, Section US.2.2, "Jini Clients and Services")

entry

A typed group of object references, expressed as a class for the Java platform that implements the `net.jini.core.entry.Entry` interface. Entry fields must all be references to `Serializable` objects. (*Jini™ Entry Specification,* Section EN.1) An entry is a class that contains a number of public fields of object type. Services provide concrete values for each of these fields; each value acts as an attribute. Entries thus provide aggregation of attributes into sets; a service may provide several entries when registering itself in the lookup service, which means that attributes on each service are provided in a set of sets. (*Jini Lookup Attribute Schema Specification*, Section LS.1, "Introduction" LS.1)

event

Something that happens in an object, corresponding to some change in the abstract state of the object. Events are abstract occurrences that are not directly observed outside of an object, and may not correspond to a change in the actual state of the object that advertises the ability to register interest in the event. (*The Jini Technology Core Platform Specification,* Section EV.2.1 "Entities Involved")

event generator

An object that has some kinds of abstract state changes that might be of interest to other objects and allows other objects to register interest in those events. This is the object that will generate notifications when events of this kind occur, sending those notifications to the event listeners that were indicated as targets in the calls that registered interest in that kind of event. (*The Jini Technology Core Platform Specification*, Section EV.2.1 "Entities Involved")

event listener

An object that has an interest in being notified when a particular event type occurs. The event listener (1) implements the appropriate interface and (2) registers with an event generator. (See also *remote event listener*)

event mailbox service

A helper service that can be employed by entities to store event notifications on their behalf. When an entity registers with the event mailbox ser-

vice, that service will collect events intended for the registered entity until the entity initiates delivery of the events. (*Introduction to Helper Utilities and Services*, Section US.4.3, "The Event Mailbox Service") (See also *notification mailbox*)

export, -ed, -ing

The process of making a remote object available to accept incoming calls on a specific port. An object can be exported (1) if the object is a subclass of `java.rmi.server.UnicastRemoteObject`, through the constructor; (2) if the object is a subclass of `java.rmi.activation.Activatable`, through the constructor; (3) by passing the object to the static `exportObject` method of `UnicastRemoteObject` (*Java™ Remote Method Invocation Specification*, Section 5.3.1); or (4) by passing the object to the static `exportObject` method of `Activatable`. (Section 7.3)

faulting remote reference

A faulting remote reference to a remote object, sometimes referred to as a fault block, "faults in" the active object's reference upon the first method invocation to the object executed via the faulting reference. Each faulting reference, contained in the remote object's stub, holds both a persistent object handle (a `java.rmi.activation.ActivationID`) and a transient remote reference to the target remote object. (*Java™ Remote Method Invocation Specification*, Section 7.1.2)

filter

A non-remote object that defines additional matching criteria that will be applied when searching for the entity's services of interest. This filtering facility is particularly useful to entities that wish to extend the capabilities of the standard template-matching scheme. (*Introduction to Helper Utilities and Services*, Section US.3.4.1, "The ServiceDiscoveryManager Helper Utility")

front end, front-end proxy

See *proxy*.

group

An organization of lookup services into a logical set. A lookup service is said to be a member of (or to belong to) a group, while a service may advertise itself to one or more groups. (*Introduction to Helper Utilities and Services*, Section US.2.1, "Terms Related to Discovery and Join") (See also *locator*)

group discovery

The discovery process in which a discovering entity employs the multicast discovery protocol to discover lookup services that are members of one or more groups belonging to a set of groups. (*Introduction to Helper Utilities and Services*, Section US.2.1, "Terms Related to Discovery and Join") (See also *locator discovery*)

helper service

A Jini service that can be registered with any number of lookup services and whose methods can execute on remote hosts. In general, a helper service should be of use to more than one type of entity participating in the Jini application environment and should provide a significant reduction in development complexity for developers of such entities. A helper service consists of an interface or set of interfaces and associated implementation that encapsulates behavior that is either required or highly desirable in services that adhere to the Jini technology programming model. (*Introduction to Helper Utilities and Services*, Section US.2.3, "Helper Service")

helper utility

Helper utilities are programming components that can be used during the construction of Jini services and/or clients. Helper utilities are *not* remote and do not register with a lookup service. Helper utilities are instantiated locally by entities wishing to employ them. (*Introduction to Helper Utilities and Services*, Section US.2.4, "Helper Utility")

host

A hardware device that may be connected to one or more networks. An individual host may house one or more JVMs. (*The Jini Technology Core Platform Specification*, Section DJ.1.2 "Host Requirements")

idempotent

A method that is idempotent can be called multiple times and produce only the result as though it were called only a single time.

indefinite exception

An exception that implies no new assertions can be made about the probability of success of any future invocation of that method, regardless of the arguments used or return value expected, nor can any new assertions be made about the probability of success of any *other* operation on the same object. For a list of exceptions that fall into this category, please see the

Introduction to Helper Utilities and Services, Section US.2.6, "What Exceptions Imply about Future Behavior".

inferior transaction

The inverse of the transactional ancestor relationship: Transaction T_i is an inferior of T_a if and only if T_a is an ancestor of T_i. (™*The Jini Technology Core Platform Specification*, Section TX.3.5 "Semantics")

Jini technology-enabled client (Jini client)

A discovering entity that can retrieve a service (or a remote reference to a service) registered with a discovered lookup service, and invoke the methods of the service to meet the entity's requirements. An entity that acts only as a client never registers with (requests residency in) a lookup service. (*Introduction to Helper Utilities and Services*, Section US.2.2, "Jini Clients and Services")

Jini technology-enabled service (Jini service)

Both a discovering and a joining entity containing methods that may be of use to some other Jini client or service, and that registers with discovered lookup services to provide access to those methods. Note that a Jini service can also act as a Jini client. (*Introduction to Helper Utilities and Services*, Section US.2.2, "Jini Clients and Services") (See also *back end, back-end server*)

joining entity

One or more cooperating objects in the Java programming language on the same host that have just received a reference to the Jini lookup service and are in the process of obtaining services from, and possibly exporting services to, a djinn. (*The Jini Technology Core Platform Specification*, Section DJ.1.1 "Terminology")

join protocol

The protocol that allows entities to start communicating usefully with services in a djinn, through the Jini lookup service. (*The Jini Technology Core Platform Specification*, Section DJ.1.3 "Protocol Overview")

JVM

A common abbreviation for "Java virtual machine."

lazy activation

The activation mechanism that the RMI system uses, which defers activating an object until a client's first use (that is, the first method invocation). Lazy activation of remote objects is implemented using a *faulting remote reference*. (*Java™ Remote Method Invocation Specification,* Section 7.1.1)

lease

A grant to use a resource, offered by one object in a distributed system, to another object in that system for a certain period of time. The duration of the lease is negotiated by the two objects when access to the resource is first requested and given. (*The Jini Technology Core Platform Specification,* Section LE.1 "Introduction") A lease ensures that the lease holder will have access to some resource for a period of time. During the period of a lease, a lease can be cancelled by the entity holding the lease. A lease holder can request that a lease be renewed, or a lease can expire. (*The Jini Technology Core Platform Specification,* Section LE.2.1 "Characteristics of a Lease") In the current implementation of RMI, a lease term is not negotiated, as described by *The Jini Technology Core Platform Specification, "Distributed Leasing";* the lease term is mandated by the implementation server. Another difference is that in RMI there is no notion of explicit lease cancellation; lease cancellation is implicit when a remote reference becomes unreferenced by a specific client. (*Java™ Remote Method Invocation Specification,* Section 9.1)

lease grantor

The object granting access to a resource for some period of time. (*The Jini Technology Core Platform Specification,* Section LE.2 "Basic Leasing Interfaces")

lease holder

The object asking for the leased resource. (*The Jini Technology Core Platform Specification,* Section LE.2 "Basic Leasing Interfaces")

lease renewal set

Clients of the renewal service organize the leases they wish to have renewed into lease renewal sets. A method is provided by the `LeaseRenewalService` interface to create these sets. These sets are then populated by methods on the sets themselves. (*Jini Lease Renewal Service Specification,* Section LR.2, "The Interface")

lease renewal service

A helper service that can be employed by both Jini clients and services to perform all lease renewal duties on their behalf. (*Introduction to Helper Utilities and Services*, Section US.4.2, "The Lease Renewal Service")

live reference

The concrete representation of a remote object reference (in the RMI transport layer), which consists of an endpoint and an object identifier. Given a live reference for a remote object, a transport can use the endpoint to set up a connection to the address space in which the remote object resides. On the server side, the transport uses the object identifier to look up the target of the remote call. (*Java™ Remote Method Invocation Specification*, Section 3.5)

local event

An event object that is fired from an event generator to an event listener, where both the generator and the listener instances exist in the same virtual machine. (See *event*, *remote event*) (*The Jini Technology Core Platform Specification*, Section EV.1.1 "Distributed Events and Notifications")

locator

An instance of the class `net.jini.core.discovery.LookupLocator`, as defined in the *Jini™ Discovery and Join Specification.* (*Introduction to Helper Utilities and Services*, Section US.2.1, "Terms Related to Discovery and Join") (See also *group*)

locator discovery

The discovery process in which a discovering entity employs the unicast discovery protocol to discover specific lookup services, each corresponding to an element in a set of locators. (*Introduction to Helper Utilities and Services*, Section US.2.1, "Terms Related to Discovery and Join") (See also *group discovery*)

lookup discovery protocol

The protocol that governs the acquisition of a reference to one (or more) instances of the Jini lookup service. (*The Jini Technology Core Platform Specification*, Section DJ.1.3 "Protocol Overview")

lookup discovery service

A helper service that employs the Jini discovery protocols to find lookup services in which an entity has expressed interest and to notify the entity

when a previously unavailable lookup service becomes available. (*Introduction to Helper Utilities and Services*, Section US.4.1, “The Lookup Discovery Service”)

lookup service
The Jini lookup service provides a central registry of service items, representing services, available within the djinn. This Jini lookup service is a primary means for programs to find services within the djinn, and is the foundation for providing user interfaces through which users and administrators can discover and interact with services in the djinn. (*The Jini Technology Core Platform Specification*, Section LU.1 “Introduction”)

lookup service group
See *group*.

manged set
When the general term *managed set* is used, it should be clear from the context whether groups, locators, lookup services, or leases are being discussed. (*Introduction to Helper Utilities and Services*, Section US.2.5, “Managed Sets”)

managed set of groups
Each element of the managed set of groups is a name of a group whose members are lookup services that the entity wishes to be discovered through group discovery. The managed set of groups is typically represented as a `String` array or a `Collection` of `String` elements. (*Introduction to Helper Utilities and Services*, Section US.2.5, “Managed Sets”)

managed set of locators
Each element of the managed set of locators corresponds to a specific lookup service that the entity wishes to be discovered via locator discovery. Typically, this set is represented as an array of locators or some other `Collection` type whose elements are locators. (*Introduction to Helper Utilities and Services*, Section US.2.5, “Managed Sets”)

managed set of lookup services
References to discovered lookup services. (*Introduction to Helper Utilities and Services*, Section US.2.5, “Managed Sets”)

marshal streams

Input/output streams, used by the RMI remote reference layer, that employ *object serialization* to enable objects in the Java programming language to be transmitted between address spaces. (*Java™ Remote Method Invocation Specification,* Section 3.3)

marshalled object

A container for an object that allows that object to be passed as a parameter in an RMI call, but postpones deserializing the object at the receiver until the application explicitly requests the object (via a call to the container object). The *serializable* object contained in the `MarshalledObject` is serialized and deserialized (when requested) with the same semantics as parameters passed in RMI calls (*Java™ Remote Method Invocation Specification,* Section 7.4.8), which means that any remote object in the `MarshalledObject` is represented by a serialized instance of its stub. The object contained by the `MarshalledObject` may be a remote object, a non-remote object, or an entire graph of remote and non-remote objects.

member group

See *group*.

multicast radius

Roughly the number of hops beyond which neither multicast requests from the entity, nor multicast announcements from the lookup service, will propagate. (*Jini Discovery Utilities Specification,* Section DU.2.6.2, “The DiscoveryChangeListener Interface”)

notification filter

A distributed event adapter that can be used by either the generator of a notification or the recipient to intercept notification calls, do processing on those calls, and act in accord with that processing (perhaps forwarding the notification, or even generating new notifications). (*The Jini Technology Core Platform Specification,* Section EV.3.2 “Notification Filters”) This filter may be used as an event multiplexer or demultiplexer.

notification mailbox

A distributed event adapter that can be used to store the notifications sent to an object until such time as the object for which the notifications were intended desires delivery. Such delivery can be in a single batch, with the mailbox storing any notifications received after the request for delivery until the next request is given. Alternatively, a notification mailbox can be

viewed as a faucet, with notifications turned on (delivering any that have arrived since the notifications were last turned off) and then delivering any subsequent notifications to an object immediately, until told to hold the notifications. (*The Jini Technology Core Platform Specification,* Section EV.3.3 “Notification Mailboxes”) (See also *event mailbox service*)

object serialization

The system that allows a bytestream to be produced from a graph of objects, sent out of the Java application environment (either saved to disk or sent over the network) and then used to re-create an equivalent set of objects with the same state. (*Java™ Object Serialization Specification,* Section A.1) In RMI, objects transmitted using the object serialization system are passed by copy to the remote address space, unless they are remote objects, in which case they are passed by reference. (*Java™ Remote Method Invocation Specification,* Section 3.3)

passive discard

The process of a utility discarding a lookup service on behalf of an entity. (See also *discarded event*)

passive object

A remote object that is not yet instantiated (or exported) in a JVM but that can be brought into an active state (See also *active object*). (*Java™ Remote Method Invocation Specification,* Section 7.1.1)

proxy

An intermediary object through which one entity (the client) may request the invocation of the methods provided by another entity (the remote object or the service). A proxy can be one of a number of different forms: a smart proxy, the stub of a remote object, or a strictly local proxy. (*Introduction to Helper Utilities and Services*, Section US.2.8.6, “Remote Objects, Stubs, and Proxies”)

pure transaction

A transaction in which all access to shared mutable state is performed under transactional control. (*The Jini Technology Core Platform Specification,* Section TX.3.5 “Semantics”)

reference list

A reference list for a remote object is a list of client JVMs that hold references to that remote object. A client JVM is removed from the object’s ref-

erence list when that client no longer references that object. (*Java™ Remote Method Invocation Specification,* Section 9.1)

registrar
See *lookup service*.

registry
A remote object that maps names to remote objects. The `java.rmi.Naming` class provides methods for lookup, binding, rebinding, unbinding, and listing the contents of a registry. A registry can be used in a virtual machine shared with other server classes or in a standalone JVM. The methods of `java.rmi.registry.LocateRegistry` may be used to get a registry operating on a particular host or host and port. (*Java™ Remote Method Invocation Specification,* Section 6)

remaining desired duration
The desired expiration (of a lease) less the current time. (*Jini Lease Utilities Specification,* Section LM.4, "The Semantics") (See also *desired expiration*)

remote event
An object that is passed from an event generator to a remote event listener to indicate that an event of a particular kind has occurred. The remote event generator and the remote event listener instances may exist in different virtual machines, possibly on different hosts. (*The Jini Technology Core Platform Specification,* Section EV.2.1 "Entities Involved")

remote event generator
An object that is the source of remote events.

remote event listener
An object implementing the `net.jini.core.event.RemoteEventListener` interface, which is interested in the occurrence of remote events in some other object. The major function of a remote event listener is to receive notifications of the occurrence of a remote event in some other object (or set of objects). (*The Jini Technology Core Platform Specification,* Section EV.2.1 "Entities Involved")

remote interface
An interface written in the Java programming language that extends `java.rmi.Remote`, either directly or indirectly, which declares the methods

of a remote object. (*Java™ Remote Method Invocation Specification,* Section 2.1)

remote method invocation (RMI)
The action of invoking a method of a remote interface on a remote object. (*Java™ Remote Method Invocation Specification,* Section 2.1)

remote object
An object whose methods can be invoked from another JVM, potentially on a different host. An object of this type is described by one or more *remote interfaces.* (*Java™ Remote Method Invocation Specification,* Section 2.1)

remote reference layer (RRL)
The layer of the RMI system that supports remote reference behavior (such as invocation to a single object or to a replicated object) and carries out the semantics of method invocation. This layer sits between the RMI stub/skeleton layer and the RMI transport layer. Also handled by the remote reference layer are the reference semantics for the server. (*Java™ Remote Method Invocation Specification,* Section 3.2)

remote server
See *remote object.*

renewal failure event
An event that is generated when the renewal manager finds that it can't renew a lease, which signals that the lease renewal manager has removed a lease from the managed set without an explicit request by the client. (*Jini Lease Utilities Specification,* Section LM.4, "The Semantics") (See also *desired expiration reached event*)

rmic
The stub and skeleton compiler used to generate the appropriate stubs and skeletons for a specific remote object implementation. The compiler is invoked with the package-qualified class name of the remote object class. The class must previously have been compiled successfully. (*Java™ Remote Method Invocation Specification,* Section 5.11)

rmid
The activation system daemon which provides an implementation of the activation system interfaces. To use activation, you must first run `rmid`. This

is the JVM with which activation descriptions get registered. (*Java™ Remote Method Invocation Specification,* Section 7.2)

rmiregistry

The RMI system command that provides an implementation of the `java.rmi.registry.Registry` interface. The rmiregistry, run on a remote host, can be accessed by calling methods of the `java.rmi.Naming` class.

semantic transaction

A *transaction* with specific, associated semantics, as opposed to the protocol specified by the `TransactionManager` interface, which does not specify transaction semantics. A semantic transaction is contractual in nature and implies a particular usage pattern, so if a program operates within the constraints of the contract, assumptions can be safely made about the transaction's behavior or state. (*The Jini Technology Core Platform Specification,* Section TX.1.1 "Model and Terms")

serializable

Any data type that may be read from `java.io.ObjectInputStreams` and written to `java.io.ObjectOutputStreams`. This includes primitive data types in the Java programming language, remote objects in the Java programming language, and non-remote objects in the Java programming language that implement the `java.io.Serializable` interface. (*Java™ Remote Method Invocation Specification,* Section 2.6)

service

Something that can be used by a person, a program, or another service. It can be computational, storage, a communication channel to another user, or another service. Examples of services include devices such as printers, displays, disks, software (such as applications or utilities), information (such as databases and files), and users of the system. Services will appear programmatically as objects in the Java programming language, perhaps made up of other objects in the Java programming language. A service will have an interface, which defines the operations that can be requested of that service. The type of the service determines the interfaces that make up that service. (*The Jini™ Architecture Specification,* Section AR.2.1.1, "Services")

service items

Each service item represents an instance of a service available within the djinn. The item contains the stub (if the service is implemented as a remote

object) or serialized object (if the service makes use of a local proxy) that programs use to access the service, and an extensible collection of attribute sets that describe the service or provide secondary interfaces to the service. A new service item is created in the Jini lookup service when a new service is added to the djinn. (*The Jini Technology Core Platform Specification*, Section LU.1.1 “The Lookup Service Model”)

service registrar

A synonym for Jini lookup service. (See *lookup service*) (*The Jini Technology Core Platform Specification*, Section LU.2.5 “ServiceRegistrar”)

skeleton

The server-side entity that reads parameters from incoming method requests and dispatches calls to the actual remote object implementation. Note that in the Java 2 Software Development Kit, Standard Edition, v1.2, skeleton functionality is now handled by the remote object stub, but skeletons may still be used for compatibility with earlier releases of the 1.1 Java Development Kit (JDK). (*Java™ Remote Method Invocation Specification*, Section 3.3)

smart proxy

A proxy that typically consists of a set of local methods and a set of one or more remote object references (stubs). Clients invoke one or more of the local methods to access the methods of the remote objects referenced in the smart proxy. (*Introduction to Helper Utilities and Services*, Section US.2.8.6, “Remote Objects, Stubs, and Proxies”) (See also *proxy*)

store-and-forward agent

A distributed event adapter that enables the object generating a notification to hand the actual notification of those who have registered interest off to a separate object. This agent can implement various policies for reliability. (*The Jini Technology Core Platform Specification*, Section EV.3.1 “Store-and-Forward Agents”)

strictly local proxy

A form of a proxy that consists of only local methods, each executing in the client’s JVM. Unlike smart proxies, no remote invocations result when any method of a strictly local proxy is invoked. (See also *smart proxy*)

stub

The proxy for a remote object, which implements all the interfaces that are supported by the remote object implementation and forwards method invocations to the actual remote object instance. (*Java™ Remote Method Invocation Specification,* Section 3.3) The stub is an object local to the client that acts as the "representative" of the remote object. From the point of view of the client, the stub *is* the remote object. When the client invokes a method on the local stub, communication with the remote object occurs, resulting in the execution of the corresponding method in the remote object's JVM.

stub/skeleton layer

The layer of the RMI system that aids in carrying out method invocation. The stub/skeleton layer is the interface between the application layer and the rest of the RMI system. (*Java™ Remote Method Invocation Specification,* Section 3.3) This layer does not deal with specifics of any transport, but transmits data to the remote reference layer via the abstraction of *marshal streams*. This layer contains client-side stubs (proxies) and server-side skeletons. (Section 3.2)

template

An entry object that has some or all of its fields set to specified *values.* Templates may be used to find matching entries. A template will match an entry if and only if the template's non-`null` public fields match the entry's non-`null` public fields exactly. Remaining fields (those set to `null`) are not used in the matching process but are left as *wildcards*. (*The Jini Technology Core Platform Specification,* Section EN.1.5 "Templates and Matching")

transaction

In general, a transaction is a tool that allows a set of operations to be grouped in such a way as to make them all appear to either all succeed or all fail; further, the operations in the set appear from outside the transaction to occur simultaneously. In the Jini architecture, the concrete representation of a transaction is encapsulated in an object. (*The Jini Technology Core Platform Specification,* Section TX.1.1 "Model and Terms")

transaction client

An object that does either or both of the following: (1) requests that a transaction manager create a transaction, (2) invokes the `commit` or `abort` method to complete a transaction. A single transaction may have more than one client, since the object that completes a transaction may be different

from the object that requested its creation. An object that is a transaction client may also be a transaction manager or participant. (*The Jini Technology Core Platform Specification,* Section TX.1.1 “Model and Terms”)

transaction manager

An object that (1) services requests from transaction clients to create transactions and (2) tracks and manages the completion state of those transactions by implementing the `TransactionManager` interface. An object that is a transaction manager may also be a transaction client or participant. (*The Jini Technology Core Platform Specification,* Section TX.1.1 “Model and Terms”)

transaction participant

An object that executes operations of a transaction and is able to interact with the manager to complete transactions properly. An object providing this service may implement the `TransactionParticipant` interface. An object that is a transaction participant may also be a transaction manager or client. (*The Jini Technology Core Platform Specification,* Section TX.1.1 “Model and Terms”)

transport

The abstraction that manages channels in the RMI transport layer. Each channel is a virtual connection between two address spaces. Within a transport, only one channel exists per pair of address spaces (the local address space and a remote address space). Given an endpoint to a remote address space, a transport sets up a channel to that address space. The transport abstraction is also responsible for accepting calls on incoming connections to the address space, setting up a connection object for the call, and dispatching to higher layers in the system. (*Java™ Remote Method Invocation Specification,* Section 3.5)

transport layer

The layer of the RMI system that is responsible for connection set up, connection management, and remote object tracking. (*Java™ Remote Method Invocation Specification,* Section 3.2) The transport layer sits below the *remote reference layer.*

weak reference

When a remote object is not referenced by any client, the RMI runtime refers to it using a weak reference. The weak reference allows the JVM’s garbage collector to discard the object if no other strong references to the

object exist. The distributed garbage collection algorithm interacts with the local JVM's garbage collector in the usual ways by holding normal or weak references to objects; thus, a weak reference allows the RMI runtime to reference a remote object, but not prevent the object from being garbage collected. (*Java™ Remote Method Invocation Specification,* Section 3.7)