Jini and Universal Plug and Play (UPnP) Notes

Abstract

Jini and UPnP are overlapping technologies. They both address the area of device connectivity and the ability to dynamically make use of new devices on the network. The emphasis in UPnP is on hardware devices offering services, whereas Jini emphasises services – both software and hardware. UPnP seems to focus on providing the ability of devices to join a network and have those devices provide an API (via HTTP and SOAP) by which other devices on the network can run their software using the offered API.
Jini is aimed at providing a broad framework for provisioning distributed computing which includes device collaboration on a network, but also includes additional, required components for robust service offerings across both a LAN and WAN. Probably the most important component to meet the goal of robust distributed computing on a network is the concept of providing code mobility between devices and application spaces.
The following document provides an outline of the basics features or functions required for distributed service offering, and tries to compare the methods of providing these services by UPnP and Jini.

Table of Contents

Common Concepts

Both Jini and UPnP have the concepts of Discovery of new devices/services on a network. Both have the concept of providing more details about the device/service. Both have the concept of how to handle a device or service leaving the network. The aspects of a distributed computing framework and how Jini and UPnP compare is shown in Figure 1.

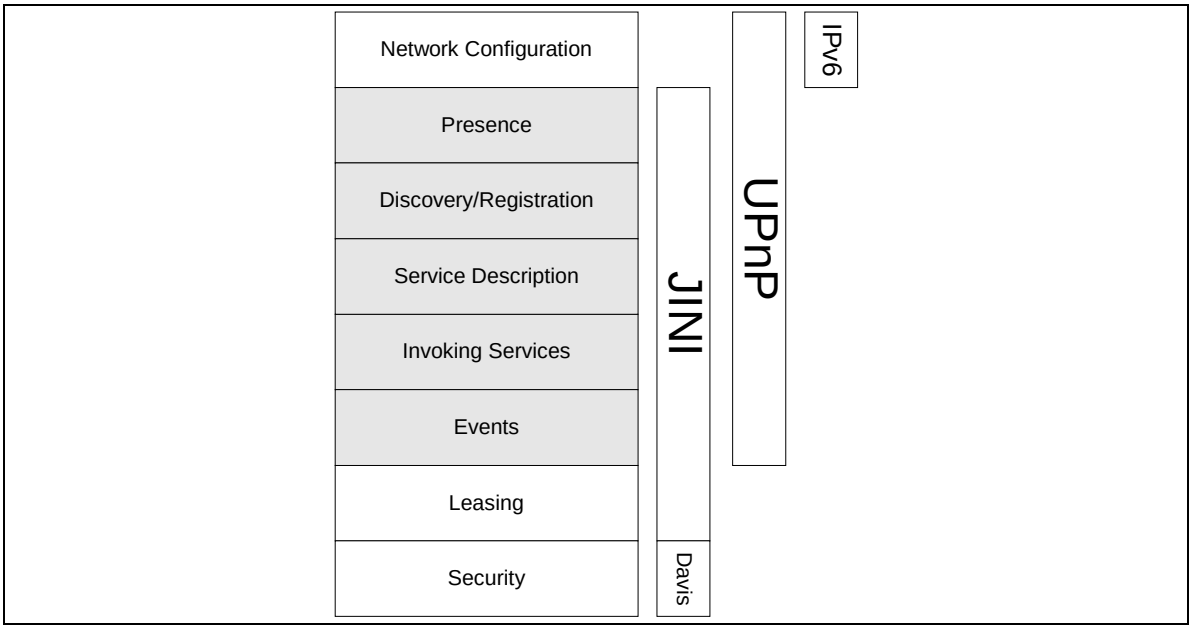

Figure 1 Common distributed computing concepts

The differences between UPnP and Jini have to do with HOW each has decided to handle the different aspects of distributed computing. This discussion is by necessity limited by UPnP to discussing how devices could offer services.

Network Configuration:

Comparison

Jini does not address the concept of a device getting an address upon joining a network. This configuration really has nothing to do with distributed computer *per se* except that one of the main preconditions that must exist before devices or service can use each other is that they must be able to communicate!

Many devices handle the allocation of an IP address to itself automatically, via Dynamic Host Configuration Protocol (DHCP). Some devices have a backup or default address that is used in the case where a DHCP server does not exist. UPnP goes further and uses 'Auto IP' to negotiate an IP address in the case where the network is not managed

(DHCP served). UPnP Auto IP defines how a device chooses an IP address from a set of reserved addresses.
IPv6, the next standard for Internet addresses handles this situation where a device joins a network without an IP address and without access to a DHCP server. For good or ill, Jini simply assumes that network connectivity exists at the TCP/IP level.

Presence

Presence or Announcing is attempting to let other devices/services/controllers on the network know that a new device or service is available. Both UPnP and JINI have similar mechanisms.

UPnP

UPnP has the concepts of devices and control points. A control point is the application software that will make use of the UPnP device services via the device's Simple Object Access Protocol (SOAP) interface. A 'control point' can be a stand-alone application or could be included as part of a device.
When a device is added to the network it is allowed to advertise it's services *to control points* using the Simple SOAP Discovery Protocol (SSDP). When a new control point is added to the network it is allowed to search for services or devices of interest on the network. In either case a simple discovery message is sent to a well-known multicast address. *Interested* control points listen to this address for notification that new services/devices are available.
All devices *must* listen to the well-known multicast address for these messages and *must* respond if any of their embedded devices or service matches the search criteria in the controller discovery message.
The advertisement and responses contain information of interest to the device or control point such as the unicast address to respond to, the version of UPnP being used, etc. The more detailed 'discovery' and information exchange is described below.
It should be noted that there is a provision to announce an UPnP device is leaving a network. This message is also multicast, but does not address the problem of a device leaving the network unannounced.

JINI

Jini uses a concept of Lookup Service(s) (LUS) that will 'register' a device or service. Jini uses multicast messaging to find these LUSs. When a device or service is added to a network, that device will advertise its existence by sending to a multicast address. If the LUS has not seen this device before, it will respond (including a specific address) and a more detailed 'discovery' or information exchange takes place.
If an LUS is added to the network, it will announce it's presence by sending a User Datagram Protocol (UDP) multicast announcement. This announcement includes (among other things) the address to use for unicast discovery (explained below) as well as the LUSs service ID. This allows the other entities on the net to ignore announcements from already announced LUSs as identified by the service ID.
Jini also allows the discovering entity to only look for 'groups' of LUS. For example, the discovering entity could specify to find only 'Engineering' LUS. This would allow 'Marketing' LUS to be ignored. (☺)

Comparison

The Jini notion of multiple LUSs that allow registration or distribution of services, allows a more flexible structure to dynamically add and use services in a network. For example, based on what other services are available, an entity in Jini may choose to withhold advertising (registering) services until other services or devices join the network. This option does not exist in UPnP.

Discovery

UPnP

To advertise the full extent of its services, a device multicasts a number of discovery messages corresponding to each of it's embedded devices and services. Each message contains information specific to the devices (or service) as well as information about its enclosing devices. Messages *should* include duration until the advertisement expire; if the device remains available, the advertisements *should* be re-sent with a new duration. If the device becomes unavailable the device should explicitly cancel its advertisements, else if unable to do so, the advertisement will expire on its own. Figure 2 presents a simplified view of this protocol.

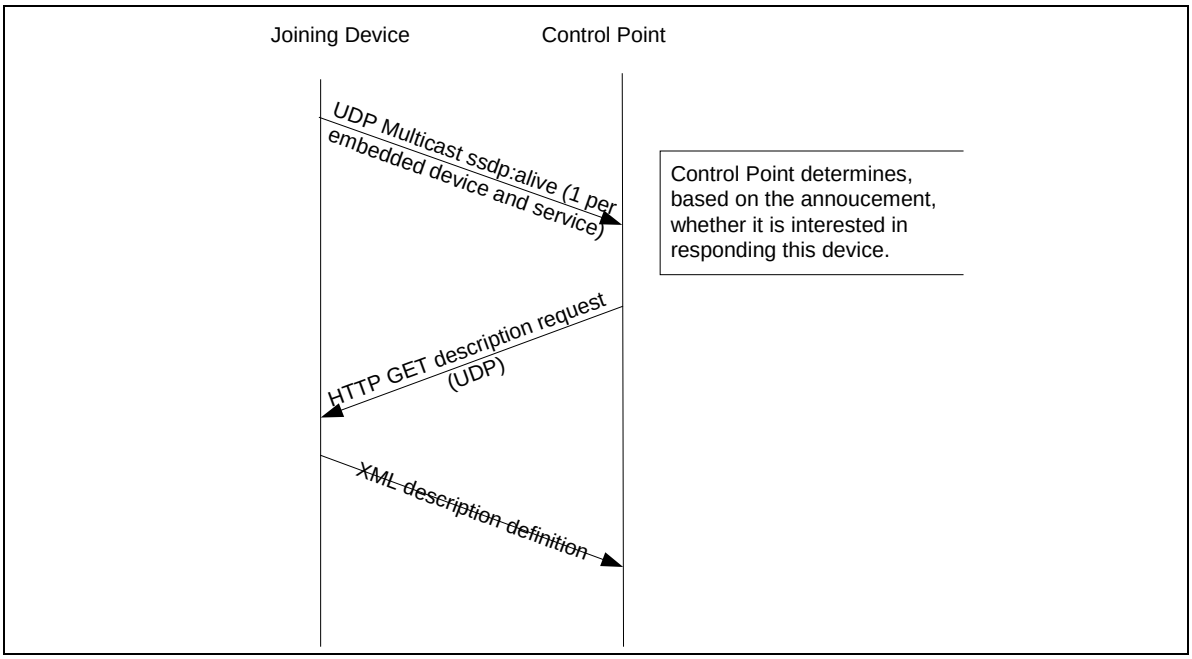

Figure 2 UPnP device join protocol

When a control point is added to the network, the UPnP discovery protocol allows that control point to search for devices of interest. The control point will multicast a message containing a pattern, or target that is equal to the type or identifier of the device or service the control point is interested in. Responses from the devices contain multicast discovery messages the same as those advertised by newly connected devices. Figure 3 presents a simplified view of this protocol.

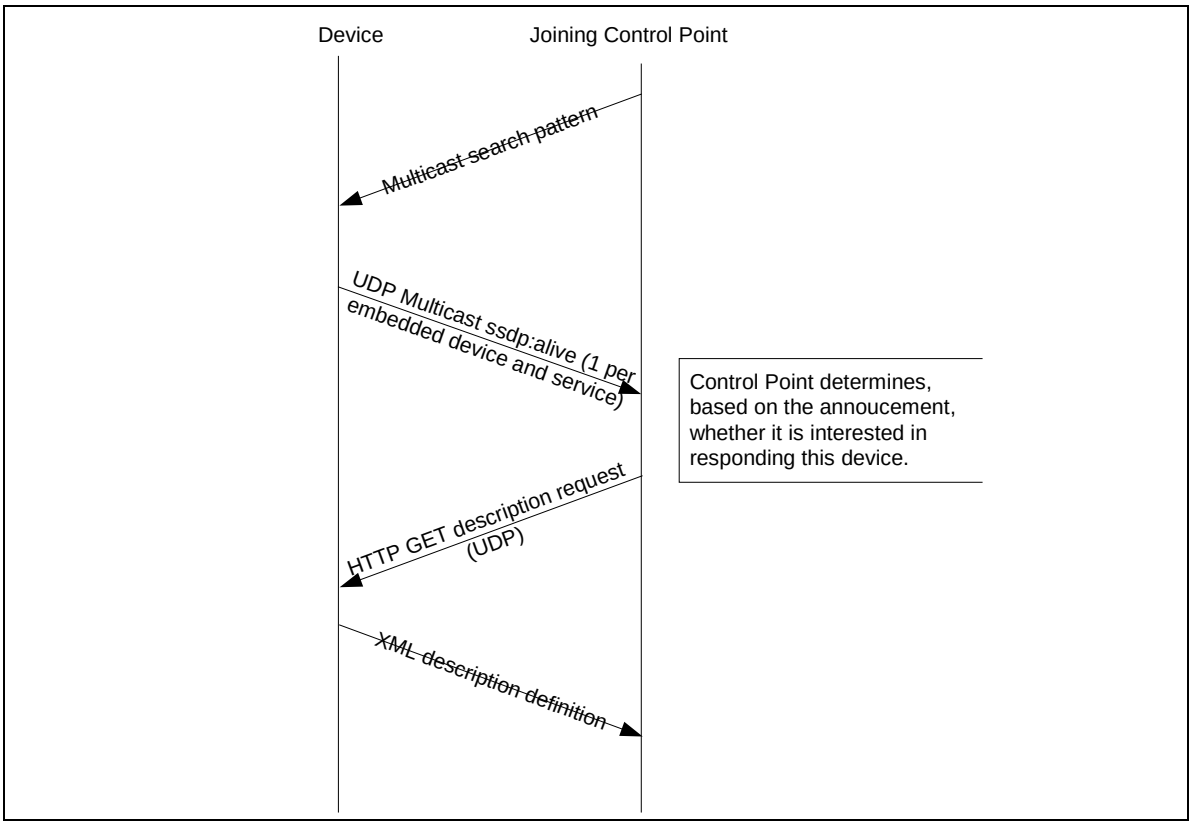

Figure 3 Control Point joining protocol

JINI

Once the device or discovering entity announces its presence to the net, the LUS determines (based on the announcement message) whether it has responded to this device before. If the LUS has not responded before, then a Transmission Control Protocol (TCP) connection is established to the device or entity. Once this connection is established, the entity makes a unicast request over this TCP connection and the LUS will respond with the ServiceRegistrar (SR) object Java byte code. . The SR can either be a proxy or the SR in its entirety (self contained). Using the SR, the discovering entity can now search the SR for the services that the LUS offers or can register a new service that the device or entity desires to offer to the rest of the network. Figure 4 presents a simplified view of this multicast protocol.

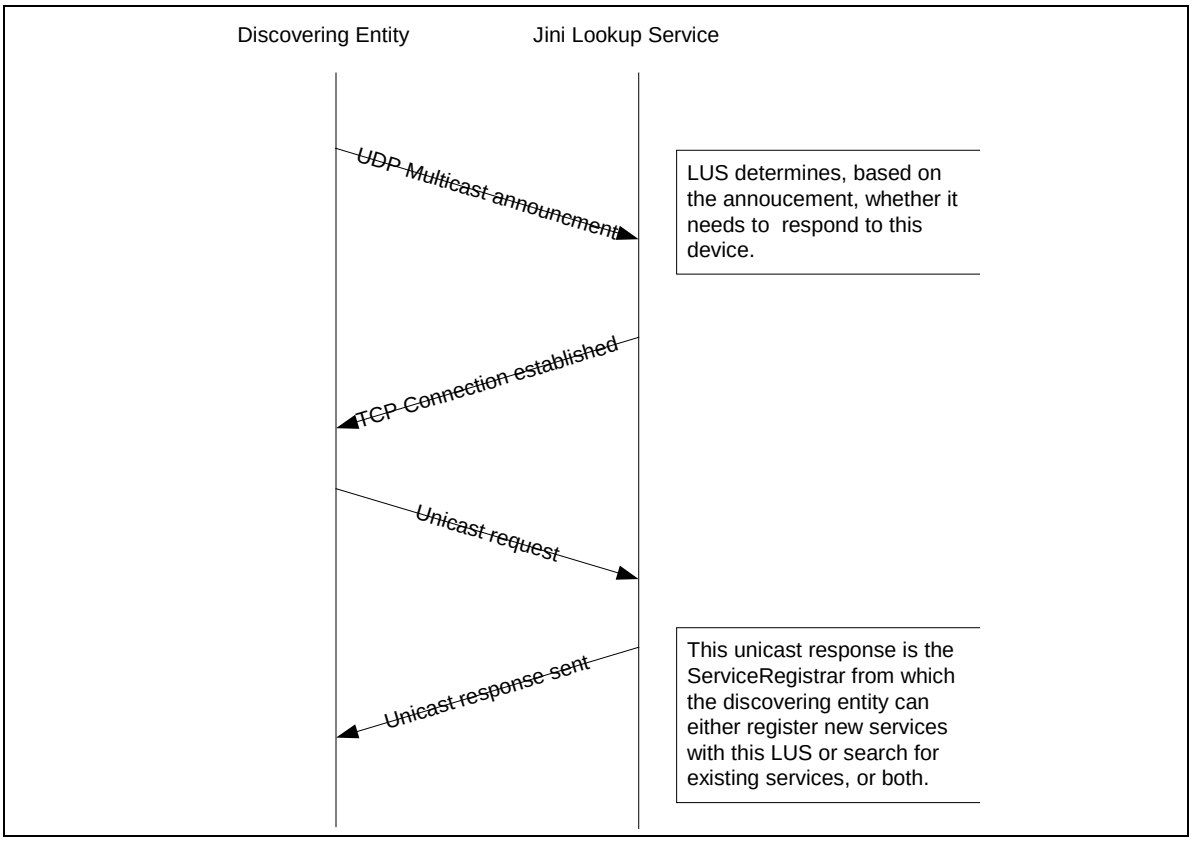

Figure 4 JINI multicast discovery protocol

JINI offers an alternative to the discovering entity using multicast to announce it's presence to an LUS. Multicast allows the entity to discover only local (as defined by the Time-To-Live [TTL] attribute on a multicast broadcast) LUSs. A device or entity may wish to access services that are dispersed more widely (for example in offices in other cities or continents). To this end, the entity must be able to obtain a reference to the LUS of the remote site. This is done by using a 'unicast discovery' protocol. The unicast Jini discovery protocol uses the underlying reliable unicast transport protocol provided by the network instead of the unreliable multicast transport. The discovering entity must know the address of the remote LUS before initiated the unicast discovery protocol.
The entity simply makes a simple request response by initiating a TCP connection to the remote LUS and sends a unicast request to this LUS. The LUS response with the ServiceRegistrar, from which point, the discovering entity can either register a new service, use an offered service, or both. Figure 5 presents this protocol diagrammatically.

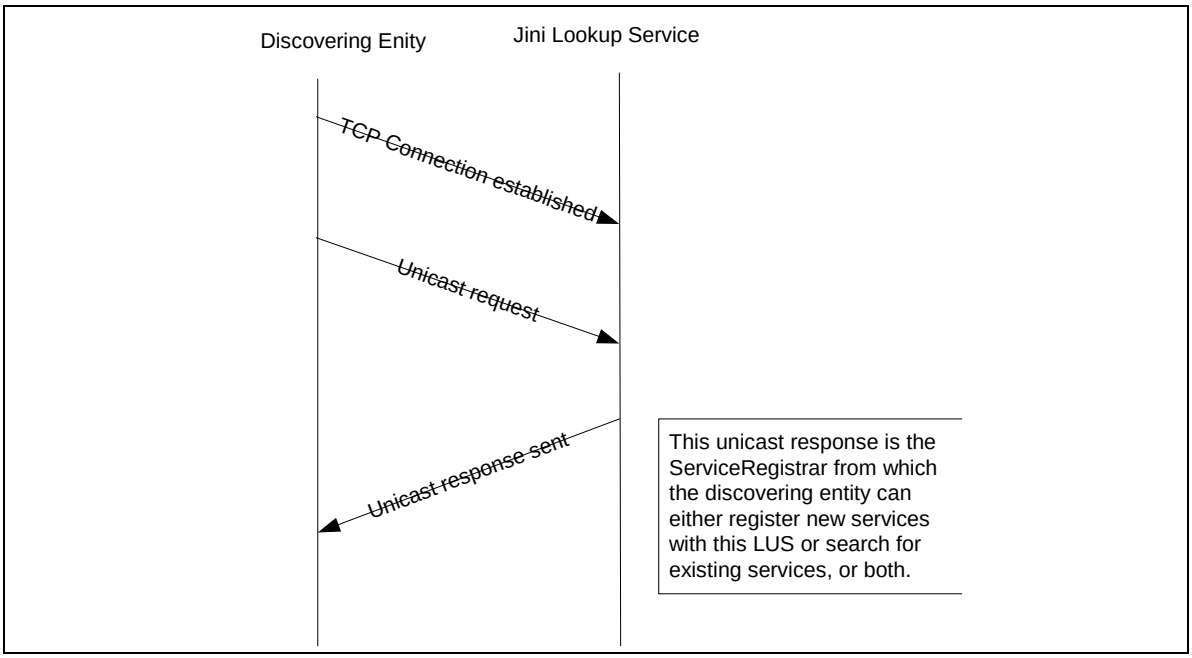

Figure 5 Unicast discovery protocol

Comparison

UPnP provides no method to obtain remote (outside of the local network) services. The devices and the control point must be 'local' (within the TTL of the multicast) to operate together. While the TTL can be increased from the UPnP default of 4, this produces a lot of network 'noise' as each router rebroadcasts the message until the TTL is subsumed. Jini uses UDP multicast for the initial announcement of the discovering entity and for an announcement of a LUS, but switches to the more reliable (but heavier weight) connection protocol (TCP) for further interactions between LUSs and entities.

Service Descriptions

UPnP

The UPnP XML description contains several pieces of specific information including

- Definitions of all embedded devices
- URL for presentation of the device (optional)
- Listing of services including (Service Description URL)
    - URL for control of device or service
    - URL for eventing

The service description and device information is only available via URLs making queries difficult. The control points must parse and filter on their own.
UPnP working groups determine the format and content for describing the device and/or services as well as the Simple Object Access Protocol (SOAP) API that is to be used to control or use the device or service. The Vendor can add to this standard XML content.

Jini

Services registered with an LUS can have an arbitrary number of attribute/value pairs associated with those services. These attribute/value pairs are amenable to queries or entry matching.
Working groups must still determine the API for device control or service invocation to allow discovering entities to use the services.

Comparison

UPnP offers a standard and defined method of describing services using a declarative language – XML. This allows control points to be able to depend on certain attributes always being available, yet makes the query of attributes or the inclusion of binary data more difficult.
Jini on the other hand uses Java and the concept of interfaces to 'describe' or match to services. For example a discovering entity could look for a 'print' interface to use to output a document. Jini also allows the service provider to associate attribute/value pairs to the service to further define the service. I.e. copies per minute, colour, etc.
In the end, each must provide some method of describing the services being offered so that clients can find services that they are interested in.

Service Invocation

UPnP

UPnP offers basically 2 methods of service invocation. The first is simply to encapsulate the call using SOAP, and then initiate the action via an HTTP request. Using this method of service invocation, the control point sends a SOAP XML string to the service control URL as specified in the service description, via HTTP. The result is returned to the control point upon completion of service execution via an HTTP response.
The 2nd possible invocation method is via the presentation URL. The control point can retrieve a page via this URL, load the page into a browser, and depending on the capabilities of the page, allow a user to control the device and/or view device status.

Jini

Services registered with a LUS are in fact the byte code required to run the service. When a discovering entity requests or gets a service from an LUS, the entity actually gets the byte code required to run that service. The service is then loaded and instantiated in the discovering entity and executed. The service obtained can be in one of 3 forms.
1. The service may represent a proxy of a service. When this is the case, the execution of an action or method of the service results in an RMI call to the physical implementation of the service. In this way the service being executed on the discovering entity is a 'proxy' for the 'real' service.
2. The 2nd form is when the implementation of the service is loaded and executed on the discovering entity. The service is no longer a proxy.

3. The 3rd form is a mixture of the first two forms in which a 'proxy' service takes the form of a software implementation that contacts the original service provider when required but attempts to do much of the processing on the client.
There is another method by which a service can be obtained by the discovering entity. In the service description above, this document mentioned that the service comes with associated attribute/value pairs. These attribute/value pairs do not need to be simple base types, but can be objects in and by themselves. For example, Jini can provide several User Interfaces (UI), from complex to simple (speech, gui, text) to the specific service simply by attaching the different methods of access as attribute/value pairs. Since these pairs are in fact the byte code required to execute the interface within a JVM, this allows a user to select which UI s/he wishes to use (or can use in the case of the disabled), even though they have not programmed the interface in advance on their interface device.

Comparison

In the service invocation model of UPnP and Jini comes a very real and significant difference in the 2 approaches.
UPnP is fixed on specifying an URL to which to pass SOAP encoded Remote Procedure Calls (RPCs) to invoke an action. While simple, this kind of method invocation does not provide much flexibility in which to operate.
Jini depends on and utilizes all of the advantages of the Java programming language and JVM. For example these advantages allow for multiple interfaces to be offered by the same service, byte code loading for purely local execution or proxy via RMI or any other RPC execution model – including SOAP if so desired.
Java's use of serialized objects and a code base, allows resource-constrained devices to offer services even when they themselves cannot hold the code necessary to implement those services. For example a device could offer a rich but heavyweight GUI service for device control but point the code base address to a less resource-constrained source from which to obtain the service.

Events

Software based on the notion of reacting to changes in other software or devices is common in a single address space. Less common, is the notion of reacting to events across a distributed network. Both Jini and UPnP have a remote event distribution mechanism.

UPnP

UPnP uses the common notions of a Publisher (originator of events) and a Subscriber (receiver of events) in its event notification system. UPnP allows a Subscriber to subscribe to changes in variables that have been flagged as 'eventable'. Some constraints are placed on UPnP events however. You cannot subscribe to a change on single variables, but will receive notification if *any* variable (flagged as eventable) changes. This is offset by the fact that UPnP has the option of combining changes into a single message instead of a separate message for each change. UPnP has the concept of a sequence number associated with the event, by which it would be possible to ascertain if you missed any events or received any 'old' events. UPnP expects the subscriber to respond to event notifications with an acknowledgement.

A UPnP Publisher will respond to a subscribe request, with an initial event message containing the current value/state of all eventable variables when a Subscriber first subscribes to a Publisher. A Publisher will also respond with a unique identifier for the subscription and duration for the subscription. This duration is an integer or the keyword 'infinite'. To keep the subscription active, a subscriber must renew its subscription before the subscription expires by sending a renewal message. All event messages are sent to the subscriber supplied delivery URL using encapsulated SOAP over HTTP.

Jini

Jini uses the features and standards inherent within the Java programming language to manage events and event reporting. "There is no single way of identifying the events that are reasonable for all objects and all kinds of events, and so there is no single way of registering interest in events"[1]

The suggested method of handling events uses the standard, existing interfaces `RemoteEventListener` and EventGenerator, along with the classes `RemoteEvent` and `RemoteException`.

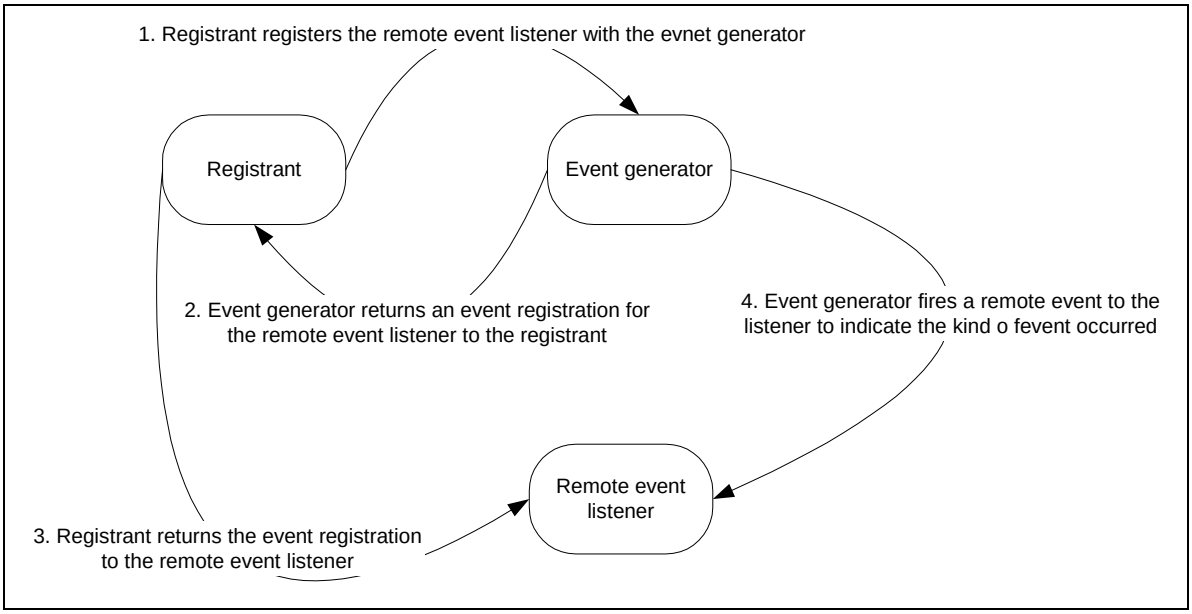

Figure 6 Jini remote event handling

It should be noted that using this suggested method of event handling, it is possible to have a registrant register for event that separate event listener would handle.

[1] Waldo et al, "The Jini™ Specifications Second Edition", Addison-Wesley, Upper Saddle River, NJ, 2000, p.150

Comparison

UPnP comes with a fixed method of handling events that seems to be very heavy weight in it's implementation while Jini (Java) allows you to tailor your event handling mechanisms using standard language constructs for distributed event handling.

Leasing

"The basic concept of leasing is that access to a resource or the request for some action is not open ended with respect to time, but granted only for some particular interval. In general, this interval is determined by negotiation between the object asking for the leased resource and the object granting access for some period. "[2]

UPnP

UPnP have no specific 'leasing' capabilities, but do have the similar concept of renewal of presence in both the device/services and the event mechanisms. During the announcement of a devices' presence on the net to the control points, an UPnP device is required to include within that announcement, a duration for which the advertisement is valid (as part of the CASHE-CONTROL header). After this duration, control points should assume that the device/service is no longer available, and removed from the control point cache. UPnP also suggests that a device send a multicast SSDP:BYEBYE message for each SSDP:ALIVE messages it multicasted that have not already expired. If a device is removed abruptly from the network, then the control point cache is cleaned up at the end of the duration as described above. The minimum duration is 1800 seconds (1 hour).

Jini

Jini has specified the following concepts of a lease.

- A lease is a time period during which the grantor of the lease (resource/service/device) ensures that the holder of the lease will have access to the resource/service/device.
- The time of the lease can be negotiated between the grantor and holder of the lease or set by the grantor.
- The holder of the lease can cancel the lease. This allows the grantor of the lease to clean up any resources associated with the lease.
- A leaseholder can request that the lease be renewed. The renewal period can be negotiated as again but not shortened.
- A lease can expire allowing the grantor and holder to clean up.

Comparison

It is obvious that the Jini leasing mechanism is more robust, with more control from both sides regarding negotiation. At the same time it is more complex to deal with than simple, set time expiration. One of the subtleties is that the UPnP duration and renewal message are sent using UDP multicast messaging which is potentially unreliable (lost messages) while Jini depend on more reliable TCP connections to do the negotiations. It would be

[2] Waldo et al, "The Jini™ Specifications, Second Edition", Addison-Wesley, Upper Saddle River, NJ, 2000, p. 123.

possible for a renewal advertisement in UPnP to be lost using a UDP message and the control point cache to be cleaned up of that service while it actually existed. The control point would have to wait for the next time the device advertised it's presence (presumably close to the minimum of 1800 seconds) before it could use the device again.

Security

UPnP

At the time of writing there is nothing in the UPnP specification to handle any level of security or encryption [2]. This is a recognized weakness for UPnP and the consortium will probably address this soon.

Jini

Version 1 of the Jini specification offers no built-in security mechanisms. It is always possible for the application to build in encryption, key exchange, etc. With version 2 of the Jini specification and corresponding protocol, there are now optional security and encryption mechanisms that can be applied. These are negotiated between the LUS and the Jini client, beginning at the request or announcement phase of the protocol. For. Example an LUS may be configured to only respond to requests from clients that specify the use of an x500 protocol (i.e. `net.jini.discovery.x500.SHA1withDSA`) AND contain a valid signer. By the same token, the client may be configured to only respond to announcements from LUSs that conform to an x500 protocol (i.e. `net.jini.discovery.x500.SHA1withDSA)` format AND contain a valid signer. The Jini protocol, version 2.0, specifies a set of standard discovery formats that can be used, but specifically does NOT preclude the creation of use of other discovery formats (for example Simple Authentication and Security Layer (SASL)). The initial specification is meant to provide baseline functionality, from which to build.

These formats are identified by the following interfaces:

1. `net.jini.discovery.plaintext`
2. `net.jini.discovery.x500.SHA1withDSA`
3. `net.jini.discovery.x500.SHA1withRSA`
4. `net.jini.discovery.kerbos`

These security features are enhanced by optional packages that provide Java Authentication and Authorization Service (JAAS), Java Cryptography Extension (JCE), and Java Secure Socket Extension (JSSE)

Notes:
DSA – Digital Signature Algorithm
SHA - Secure Hash Algorithm
RSA – Rivest, Shamir, Adleman [4]

Comparison

Since Jini alone provides a security mechanism, aside from application specific methods, there is no comparison.

References:

[1] Waldo et al, “The Jini™ Specifications Second Edition”, Addison-Wesley, Upper Saddle River, NJ, 2000.

[2] http://www.upnp.org/download/UPnPDA10_20000613.htm, “Universal Plug and Play Device Architecture, Version 1.0”, Microsoft Corporation, 1999-2000

[3] “Jini™ Technology Core Platform Specification”, Sun Microsystems, version 2.0, June 2003.

[4] Rivest, R.L., Shamir, A., Adleman, L.M., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, vol. 21, 1978, pp. 120-126.

