# PROTOCOLS FOR SERVICE DISCOVERY IN DYNAMIC AND MOBILE NETWORKS

## Choonhwa Lee and Sumi Helal

Computer and Information Science and Engineering Dept.
University of Florida
Gainesville, FL 32611-6120
UNITED OF STATES
{chl,helal}@cise.ufl.edu  http://www.cise.ufl.edu/~{chl,helal}

*Abstract:* - Mobile and wireless computing is permeating the globe, affecting the way we live and conduct business using portable computers such as laptops, hand-held PC's, Palm computers, and even wearable computers. In becoming mobile, users innocently expect and demand the same computing luxury they used to (and still) enjoy in the fixed computing environment. Unfortunately, network resources (e.g. printers, fax machines, and file systems) and application software do not follow the mobile users when they leave their offices or homes, or when they relocate to another temporary office or home. Supporting true mobility of users and business task forces will therefore require changing the way application software is packaged and deployed. It will also require changing the mechanisms by which system resource are configured and managed. The lesson quickly learned by the mobile computing research community is that mobility and dynamism are synonymous. Service discovery protocols (SDP) are re-shaping the way software and network resources are configured, deployed, and advertised, all in favor of the mobile user. Furthermore, so called *wireless local connectivity* technologies (e.g. Infrared and Bluetooth) are fueling the importance of SDP by enabling the mobile user to discover and use proximity services seamlessly and effortlessly. In this paper we discuss the emerging technology of *service discovery* in the context of mobile and wireless computing, with emphasis on Jini, UpnP, and Salutation protocols.

*Key-words:* - Service discovery, service discovery protocols, lookup services, mobile computing, Jini, UpnP, Salutation, pervasive computing.

## 1 INTRODUCTION

With the emergence of wireless ad-hoc networks, specialized information appliances are taking over the technology landscape. These information appliances have been born to aim at supporting mobility, in essence, and hence cooperation among them, since cooperation is an indispensable feature that complements some missing parts in mobile device, compared to conventional, fully-powered computing devices. For this cooperation, several service discovery protocols (SDPs) have been proposed as the part of coordination architectures that ensure device interaction with the ultimate aim of simple, seamless and scaleable device inter-operability. Among emerging SDPs, Jini, Universal Plug and Play, Salutation, and

SLP are conspicuous. Before discussing them, let us take a look at how information appliances powered by SDPs can be used in the real life.

**Scenario 1:** Imagine finding yourself in a taxi cab without your wallet. Fortunately, you have a Jini technology-enabled cellular screen phone, and your cellular provider uses Jini technology to deliver network-based services tailored to your community. On your phone screen, you see a service for the City Cab Company, so you download the electronic payment application to authorize payment of your cab fare. The cab company's payment system instantly recognizes the transaction and sends a receipt to the printer in the taxi. You take the receipt and you're on your way.

**Scenario 2:** Consider an insurance salesman who visits a client's office. He wants to brief new products and their options to the client which are stored in his Windows CE Handheld PC. Since his handheld PC has wireless network and supports UPnP, it automatically discovers and uses an Ethernet-connected printer there without any network configuration and setup. He can print whatever in his H/PC or from computers in his main office and promote the new products.

**Scenario 3:** Consider an intelligent, on-line overhead projector with a library client. After identification to the system, the user may select a set of electronically stored charts or other document(s) for viewing. Rather than bringing foils to a meeting, the user accesses them through the LAN server in the library.

Scenario 1 is a Jini demo scenario told by Sun and scenario 2 is a UPnP scenario by Microsoft. The last one is Salutation scenario. At a glance, they seem to talk about the same stories: mobile devices, zero-configuration and impromptu community enabled by SDPs, and cooperation of the proximity network. Without trademarks such as Jini and UPnP, we could hardly know which scenario is told by whom.

Even though they play in the same arena, these three SDPs have different origins, underlying technologies, flavors, and audiences. Since they see the problem at different angles and take different approaches to it, they have pros and cons, especially compared to others. This survey is focused on not their overall architectures but SDPs. Before the comparison of the big three, let us take a brief look at the basic mechanisms, goals, and environments of them.

## 2   JINI CONNECTION TECHNOLOGY

The purpose of the Jini architecture is to federate groups of devices and software components into a single, dynamic distributed system [2]. 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 database and files; and users of the system.

The heart of the Jini system is a trio of protocols called *discovery*, *join*, and *lookup* [2]. A pair of these protocols – discovery/join – occurs 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. The following steps show what interactions are needed among a client, a service provider, and a lookup service for a service to be used by the client in a Jini community [2] [1].

1. Service provider locates a lookup service by multicasting a request on the local network or a remote lookup service known to it in priori.

2. A service provider registers a service object and its service attributes with the lookup service. This service object contains 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.
3. A client requests a service by Java type and, perhaps, other service attributes. A copy of the service object is moved to the client and used by the client to talk to the service.
4. Then, client interacts directly with the service provider via the service object.

Jini connection technology consists of an infrastructure and a programming model which address the fundamental issues of how devices connect with each other to form an impromptu community. Based on Java technology as shown in Fig.1 [1] [2], Jini technology uses Java Remote Method Invocation protocols to move code around the network. Network services run on top of the Jini software architecture.

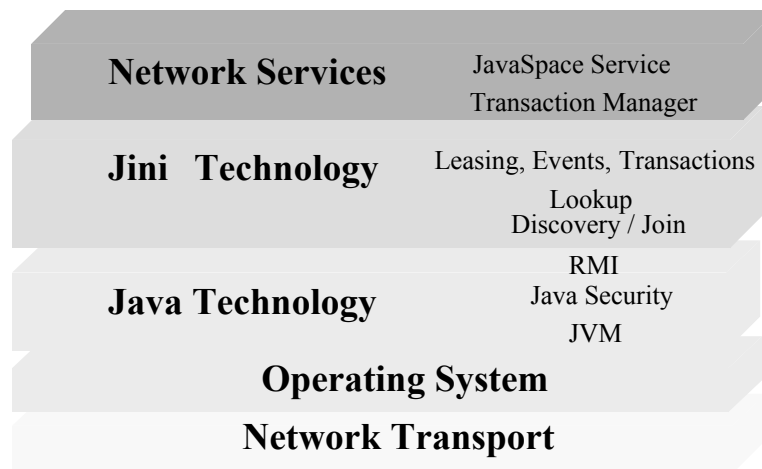| Network Services | JavaSpace Service |
| | Transaction Manager |
| Jini  Technology | Leasing, Events, Transactions |
| | Lookup |
| | Discovery / Join |
| Java Technology | RMI |
| | Java Security |
| | JVM |
| Operating System | |
| Network Transport | |

Fig.1  Architecture of Jini Connection Technology

## 2.1 Lookup Service

The lookup service can be viewed as a directory service, in that services are found and resolved through it. In a Jini community, services register their proxy objects with a lookup service through discovery and join process, and clients query the lookup service to find out the services they want. Jini uses three related discovery protocols, useful in different situation [3]  [4]. *Multicast Request Protocol* is used when an application or service first becomes active, and needs to find lookup services in the vicinity. *Multicast Announcement Protocol* is used by lookup services to announce their presence to the services that may have interest in the community. *Unicast Discovery Protocol* is used to establish communications with a specific lookup service known to it in priori over a wide-area network.

But a Jini lookup service does much more than a simple name server. Client sees a service as an interface, including methods that the client will invoke to execute the service, along with any other descriptive attributes. The lookup service maps interfaces seen by clients to set of service proxy objects. Client downloads the service proxy, which is actually RMI stub that can communicate back with the server. This proxy object enables client to use the service without knowing anything about it. Hence, no

need for device driver scenario. Although service proxy object is typical scenario of service invocation, i.e. accessing services through RMI method invocation, the downloaded service object can be the service itself or a smart object capable of speaking any private communication protocol.

## 2.2 Leasing

Access to services in the Jini system is granted on lease basis: A service is requested for a time period and, then, granted for negotiated period between the service user and provider. This lease must be renewed before its expiration. Otherwise, the resources associated with the services are released. For the example, the lookup service grants lease to a service registration and the service should continue to renew the lease. A device can leave the community or fail abruptly without having a chance to deregister itself. So, It is the leasing that enables the Jini system to be kept robust and maintenance-free.

## 2.3 Remote Events and Transactions

Besides the basic service discovery/join and lookup mechanism, Jini supports *remote events* and *transactions* that help programmers write distributed programs in reliable and scalable fashion. Remote event enables an object to be notified when desired change occurs in the system. These events can be triggered by newly-published services or some state changes of services. For example, a Jini palmtop that registered its interest in printers can be notified by the lookup service, when a printer becomes available. Also, Jini supports two-phase commit (2PC) protocol. By nature, Jini is used to build distributed systems where reliability and robustness are likely to get impaired by partial failures and recovery. But Jini 2PC allows flexibility, in that it does not dictate this protocol to be followed strictly. Rather, it is being left to applications (transaction participants) to implement necessary actions intended by the application logic.

## 3  UNIVERSAL PLUG AND PLAY

Universal Plug and Play (UPnP) is an architecture for pervasive peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs of all form factors. Although it's introduced as an extension to the plug and play peripheral model, UPnP is more than a simple extension to it. In UPnP, a device can dynamically join a network, obtain an IP address, convey its capabilities upon request, and learn about the presence and capabilities of other devices. Finally, a device can leave a network smoothly and automatically without leaving any unwanted state behind [6]. Universal Plug and Play leverages TCP/IP and the Web technologies, including IP, TCP, UDP, HTTP and XML, to enable seamless proximity networking in addition to control and data transfer among networked devices in the home and office.

UPnP uses Simple Service Discovery Protocol (SSDP) [7] for service discovery. This protocol is used for announcing a device's presence to others as well as discovering other devices or services. Therefore, SSDP is analogous to the trio of protocols in Jini: discovery, join, and lookup. SSDP uses HTTP over multicast and unicast UDP which are referred to as HTTPMU and HTTPU, respectively.

A joining device sends out a *advertisement (ssdp:alive)* multicast message to advertise its services to control points. They are the potential clients of services embedded into the device.

In contrast to Jini, there is no central service registry in UPnP. The other message of SSDP is *search (ssdp:discover)* multicast message sent when a new control point is added to the network. Any device that hears this multicast should respond to it with a unicast response message.

   XML is used to describe device features and capabilities. The aforementioned advertisement message contains a URL that points to an XML file in the network, describing the UPnP device's capability. Hence other devices, by retrieving this XML file, can inspect the features of this device and decide whether it fits their purposes. This XML description allows complex, powerful description of device capability as opposed to Jini's simple service attribute.

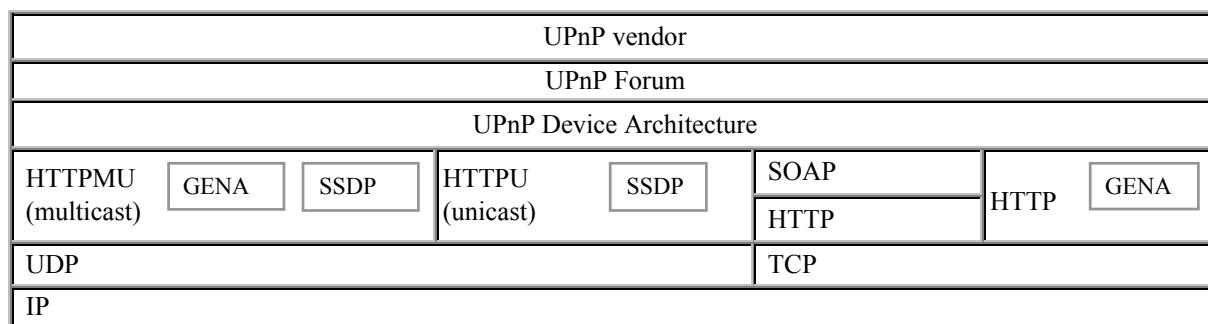| UPnP vendor | | | | | | | |
|---|---|---|---|---|---|---|---|
| UPnP Forum | | | | | | | |
| UPnP Device Architecture | | | | | | | |
| HTTPMU (multicast) | GENA | SSDP | HTTPU (unicast) | SSDP | SOAP | HTTP | GENA |
| | | | | | HTTP | | |
| UDP | | | | | TCP | | |
| IP | | | | | | | |

Fig.2  UPnP Protocol Stack

For communication between devices, UPnP uses the protocol stack shown in Fig.2 [6]. According to the latest specification [6], UPnP features can be epitomized as the following five steps.

1. *Discovery*: The UPnP discovery protocol is based on SSDP. When a device added to the network, the device advertises its services to the control points on the network. Similarly, when a control point is added to the network, the UPnP allows that control point to search for devices of interest on the network. The fundamental exchange in both cases is a discovery message containing a few, essential specifics about the device or one of its services, e.g., its type, identifier, and a pointer to more detailed information.

2. *Description:* After a control point has discovered a device, the control point still knows very little about the device. For the control point to learn more about the device and its capabilities, or to interact with the device, the control point must retrieve the device's description from the URL provided by the device in the discovery message. The UPnP description for a device is expressed in XML and includes a list of any embedded devices or service, as well as URLs for control, eventing, and presentation.

3. *Control:* After a control point has retrieved a description of the device, the control point can send actions to a device's service. To do this, a control point sends a suitable control message to the control URL for the service. Control messages are also expressed in XML using the Simple Object Access Protocol (SOAP). Like function calls, in response to the control message, the service returns any action-specific values.

4. *Eventing:* An UPnP description for a service includes a list of actions the service responds to and a list of variables that model the state of the service at run time. The service publishes updates when these variables change, and a control point may subscribe to receive this information. The service publishes updates by sending event messages. Event messages contain the names of one of more state variables and the current value of those variables. These messages are also expressed in XML and formatted using the General Event Notification Architecture (GENA).

5. *Presentation:* If a device has a URL for presentation, then the control point can retrieve a page from 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.

Another important feature of UPnP is automatic configuration of IP addresses being plugged in. Being introduced for this purpose, AutoIP [13] enables a device to join the network without any explicit administration. When a device is connected to the network, it tries to acquire an IP address from a DHCP server on the network. But in the absence of a DHCP server, an IP address is claimed automatically from a reserved range for the local network use. So, named as AutoIP. The device claims an address by randomly choosing an address in the reserved range and then making an ARP request to see if anyone else has already claimed that address.

## 4  SALUTATION

Salutation is another major cooperation architecture, which is being developed by the Salutation Consortium, to solve the problems of service discovery and utilization among a broad set of appliances and equipment and in an environment of widespread connectivity and mobility. Given the diverse nature of target devices, it is processor, operating system and communication protocol independent. The architecture provides a standard method for applications, services and devices to describe and to advertise their capabilities to other applications, services and devices. The architecture also enables application, services and devices to search other applications, services or devices for a particular capability, and to request and establish interoperable sessions with them to utilize their capabilities [8] [9].

As shown in Fig.3 [8], the Salutation architecture is composed of two major components: *Salutation Manager* and *Transport Manager*. The Salutation Manager is the core of the architecture, similar to the lookup service in Jini. It is defined as a service broker: "A service provider registers its capability with a Salutation Manager. When a client ask its local Salutation Manager for a service search, the search is performed by coordination among Salutation Managers. Then, the client can use the returned service." A Salutation Manager sits on the Transport Managers that provide reliable communication channels, regardless of what the underlying network transports are.

The Salutation Manager provides a transport-independent interface to Server and Client applications. This interface (SLM-API) includes service registration, service discovery, and service access function. The communication protocol independence of Salutation architecture is achieved by the interface (SLM-TI) between Salutation Manager and Transport Manager. Transport Manager is an entity, dependent on the network transport it supports. A Salutation Manager may have more than one Transport Manager, in case it is attached to multiple, physically different networks. But Salutation Manager sees its underlying transport through the transport-independent interface (SLM-TI).
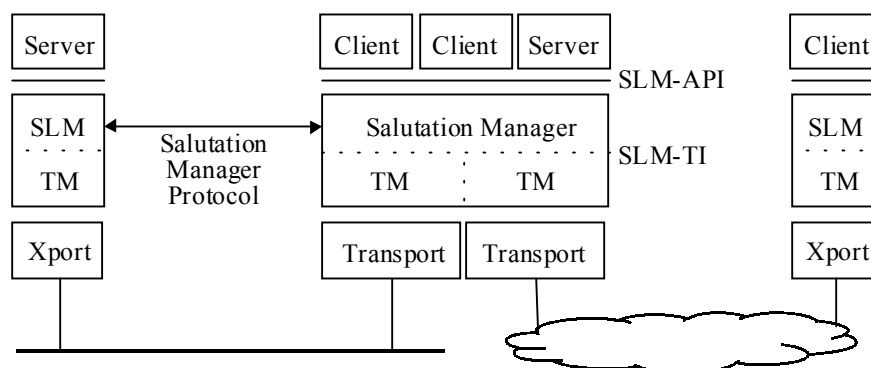
Fig.3  Model of the Salutation Manager

The main tasks provided by the Salutation Manager can be summarized as follows.

- **Service Registry**: The Salutation Manager contains a registry to keep information about Services. A client registers or unregisters itself. All registration is done with the local Salutation Manager or near one connected to the client. This is correspondent to the lookup service in Jini.

- **Service Discovery**: The Salutation Manager discovers other Salutation Managers and services registered there. Remote services are discovered by matching type(s) and set of attributes specified by local Salutation Manager. This communication protocol between Salutation Managers is called the Salutation Manger Protocol using Sun's ONC RPC. This unique feature, called capability exchange, is needed, because services are basically registered with the local Salutation Manager in the same equipment. This cooperation among Salutation Managers forms the conceptually same lookup service, but distributed over the network, as Jini does.

- **Service Availability**: A client application can ask the local Salutation Manager to periodically check the availability of services. This checking is done between the local manager and the corresponding manager. This is a narrow version of Jini's Remote Event concept.

- **Service Session Management**: This session management addresses the service invocation aspect of Salutation. A service session is established when a client wants to use a service discovered through Service Discovery. The service session is operated in one of 3 different modes: *native mode, emulated mode, and salutation mode*. The Salutation Manager may or may not be involved in message exchanges in the service session, depending on the modes. In the *native mode*, messages are exchanged through a native protocol and Salutation Manager is never involved in message exchange. In the *emulated mode*, the Salutation Manager Protocol is used to carry messages between client and service but Salutation Manager doesn't inspect the contents. In the *salutation mode*, Salutation Managers not only carry messages, but also define the message formats to be used in the session.

A *Functional Unit* is defined as a basic building block in Salutation architecture. In other words, it is the minimal meaningful function to constitute a client or service. A collection of Functional Units defines a Service Record. For example, a fax service can be defined by [Print], [Scan], and [Fax Data Send] Functional Units. Each functional unit is composed of descriptive attribute record. These Service/Funtional Unit/Attribute records are specified with ISO 8824 ASN.1.

Salutation-Lite [10] is also worth to mention here. Salutation-Lite is a scaled down version of the Salutation architecture targeted at devices with small footprints. The Salutation Consortium envisions that Salutation-Lite has tremendous applicability to small information appliances such as palm-size and hand-held computers (i.e. Palm and WinCE devices). Salutation-Lite also lends itself well to low bandwidth network such as IR and Bluetooth.

# 5 SERVICE LOCATION PROTOCOL AND OTHERS

Service Location Discovery (SLP) [17] is an IETF version of service discovery protocol but it has unique backgrounds, target areas, and features, as other service discovery protocols do. SLP is a decentralized, lightweight, scale and extensible protocol for service discovery within a site [16]. SLP defines *Service URL* which defines service type and address for the service. For example, "service:printer:lpr://hostname" is the Service URL for line printer service available at hostname. Based this Service URL, a user browses services available in its site and makes use of selected services to meet the user's need. For example, a user (application) uses SLP to find out any color printer on the same floor.

There are three agents in SLP: *User Agent (UA), Service Agent (SA), and Directory Agent (DA)*. UA is a software entity that sends service discovery requests on behalf of a user application. SA is an entity that advertises service on behalf of a service. As a centralized service information repository, DA caches advertisements from SAs and, afterwards, responds to requests from UAs. An SA advertises itself by registering with a DA. This registration message contains the URL for the advertised service, lifetime for the service, and a set of descriptive attributes for the service. The SA should periodically refresh the registration with DA before its expiration. This lifetime is meant to prevent the network from being left in transient state and similar concept is found at other service discovery protocols such as Jini and UpnP. A DA caches the registration and sends an acknowledge message to the SA. A UA send a service request message to the DA to request the location of a service. Then, the DA responds with a service reply message including the URLs of the services matched against the UA needs. Now, the UA can access the service pointed by the returned URL. In SLP, DA is optional. There may be no DA in small networks. In this case the UA's service request message is directly sent to SAs.

SLP supports service browsing and string-based query for service attributes which allow UA to select the most appropriate service from among services on the network. The UA can request query operators such as AND, OR, comparators (=, <, <=, >, >=), and substring matching. This is more powerful than others. For example, in Jini, service attribute matching can be done only against equality.

Finally, SLP is said to be a solution to the intranet service discovery needs but it scales well to larger network. The scalability is supported by various features such as the minimal use of multicast messages, *scope* concept, and multiple DAs.

Bluetooth protocol stack also contains a SDP [14] for service discovery. Since Bluetooth SDP is designed specically for Bluetooth environments, it supports limited functionality, compared to other service discovery protocols. Basically, SDP supports search by service class, search by service attributes, and service browsing. Service browsing is used when a client has no priori knowledge about services

available in the client's vicinity. Service discovery application profile [15] defines protocols and procedures used by a service discovery application to locate services in other devices. Bluetooth SDP runs on a predefined connection-oriented channel of L2CAP.

Bluetooth SDP is optimized for Bluetooth devices with limited complexity. Thus, it addresses primarily service discovery problem. It provides neither access to services, brokering of service, service advertisement, nor service registration. There's no event notification when services become unavailable. Therefore, other service discovery protocol might be used to complement these lacks. For example, Salutation can be used above Bluetooth SDP. Such mapping [11] seems to be neat because of Salutation's transport-independent architecture.

There are other players in this area: Zero Configuration Networking (zeroconf) [20], MIT's INS (Intentional Naming System) [21], and the Berkeley Service Discovery Service [22]. With a different objective, each of them takes a different approach from others. As a result, they have some strong and weak features, relatively compared to other protocols.

# 6 COMPARISON

Several service discovery protocols are proposed to facilitate dynamic cooperation among devices/services with minimal administration and human intervention. In order to be able to support the impromptu community, they should provide the means to announce its presence to the network, to discover services in the neighborhood, and to access to services. Basically, all Jini, UPnP, Salutation, and SLP address these aspects, but in different perspectives. A direct comparison must be avoided, since they put different weights on the above functionality. Nevertheless, such comparison is tried here, since it would be helpful to understand each of them. Table 1 summarizes the features of major service discovery protocols.

Jini and UPnP envision pervasive computing environments being enabled by their solutions, whereas Salutation and SLP are primarily dealing with the service discovery problem. Note that Jini provides 2PC transaction and JavaSpace to help develop network services [3]. UPnP's SSDP is just a part of UPnP specification. A good comparison among Jini, UPnP, and Salutation is presented in [19].

Jini has a dependence on Java to enable all its promises. It assumes that devices support Java Virtual Machine, even though a Jini-proxy can be used for a cluster of resource-poor devices [3]. Moreover, Jini/RMI is not supported by J2ME CLDC (Connected Limited, Device Configuration) configuration for small information devices such as cell phone, pager, and POS [24].

Jini's service proxy concept is one of strongest features not found at others. But this no-need-for-drivers scenario presumes the Jini devices to support standard interfaces are already available in the network. It's not as simple as it sounds, since it means all manufactures of a certain device type must consent to the standard interface. First, the standardizations for printer and storage device interfaces are under way by the consortia of manufactures.

UPnP relies on the existing IP and Web technology. It seems unique in terms of its XML use for service/device description. XML allows for powerful description of device capability, control command issued to the device, event from it. UPnP introduces new features for self-configuration which exploit AutoIP and DHCP, but these features are also found in IPv6 [19]

The Salutation is well defined but confined to the service discovery protocol and session management. Salutation accordingly doesn't address features like remote event notification, which are no doubt useful in distributed environment. When it comes to transport protocol, IP is given top priority by

Jini, UPnP/SSDP, and SLP. Salutation can operate in any network, including IP, IR and, in the future, wireless. This transport independence is the strongest feature of Salutation.

More than one SLP DAs are likely to be deployed for an enterprise network, since a DA becomes the single point of failure. These DAs can be organized in a hierarchy to provide better performance. Also, there may be some overlap in their coverage of organization/departments to provide reliability. This interaction or cooperation between DAs for performance and reliability is being explored by SLP society. SLPv2 can ensure the integrity and authenticity of SLP messages by including authentication information in SLP message. It deals with security problem directly, while others have to rely on other security protocols.


# 7  CONCLUSION


This survey attempts to take a look at key service discovery protocols that address more or less the same concepts. These service discovery protocols are vying with one another to be final winners. On the other hand, they are also making efforts to integrate themselves with other protocols. A Jini-to-SLP bridge developed by Sun is one of such efforts. Salutation architecture has already aligned its architecture with SLP (enhanced to support directory-based service discovery by utilizing SLP). As another example, the use of Jini's service proxy object in Salutation platform is made possible by [DOC Storage] Functional Unit, although its use is not limited to Jini proxy object. It can be used for device drivers and application programs.

There seems to be no clear choice today. All of them have wonderful architectures but technological superiority is not the only factor to be prevalent in the market. We expect more changes and competitions in the upcoming years, accelerated by industry feedback following their deployment. It would be very interesting to watch where this dynamic evolution converges.


## REFERENCES

[1] Sun Microsystems, "Jini Connection Technology," http://www.sun.com/ jini
[2] Sun Microsystems, "Jini Community Resources: Jini Technology Architectural Overview," January 1999. http://www.sun.com/jini /whitepapers/architecture.html
[3] Sun Microsystems, "Jini Community Resources: Jini Specification v1.0.1," www.sun.com/jini/specs
[4] W. Keith Edwards, *Core JINI, The Sun Microsystems Press Java Series*, Prentice Hall, 1999.
[5] Microsoft Corporation, "Universal Plug and Play: Background," http://www.upnp.org/resources/UpnPbkgnd.htm
[6] Microsoft Corporation, "Universal Plug and Play Device Architecture Version 1.0," June 8, 2000. http://www.upnp.org/UpnPDevice_ Architecture_1.0.htm
[7] Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu, and Shivaun Albright, "Simple Service Discovery Protocol," IETF Draft draft-cai-ssdp-v1-03.txt, October 28, 1999. http://www.ietf.org/internet-drafts/draft-cai-ssdp-v1-03.txt
[8] Salutation Consortium, "Salutation Architecture Specification Version 2.0c – Part 1," The Salutation Consortium, June 1, 1999.  http://www.salutation.org

| | Jini | UPnP | Salutation | SLP |
|---|---|---|---|---|
| **Home Page** | www.sun.com/jini | www.upnp.org | www.salutation.org | www.svrloc.org |
| **Main Entities** | Lookup Service, Client , Service | Control Point, Devices (Services) | Salutation Manager, Transport Manager, Client, Server | Directory Agent, Service Agent, User Agent |
| **Service Repository** | Lookup Service | No | A set of SLMs (Salutation Managers) | DA (Directory Agent) |
| **Service Announcement** | Discovery/Join protocol | Advertisement (ssdp:alive) | Registering with local Salutation Manager | Service registration |
| **Service Discovery** | Query to lookup service | Contact control point, or listen to advertisement | Query to local SLM and cooperation among SLMs | Contact DA, or multicast to SAs |
| **Access to Service** | Service proxy object based on RMI | Invoking action to the service (SOAP), Query for variable state | Service Session management | Service type (service protocol) for the discovered service |
| **Service Description and Scoping** | Interface type and attribute matching | Description in XML | FU (Functional Unit) and attributes within it | Service type And attribute matching (fairly powerful matching) |
| **Service Registration Lifetime** | Leasing | CACHE-CONTROL header in alive message | No | Lifetime in service registration |
| **Service Group** | Group | No | No | Scope |
| **Event Notification** | Remote Events | Service publishes event(GENA) when state variable changes | Availability Checking (periodic & automatic) | No |
| **Others** | Java-centric architecture | Automatic con-figuration (AutoIP) | Transport independence | Authentication security feature |

Table 1  Summary of major service discovery protocol

[9] Salutation Consortium, "Salutation Architecture Specification Version 2.0c – Part 2," The Salutation Consorium, June 1, 1999. http://www.salutation.org

[10] Bob Pascoe, "Salutation-Lite: Find-and-Bind Technologies for Mobile Devices," Salutation Consortium, June 6, 1999. http://www.salutation.org/whitepaper/Sal-Lite.PDF

[11] Brent Miller, "Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer," Bluetooth Consortium 1.C.118/1.0, July 1, 1999.

[12] Bob Pascoe, "Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun," Salutation Consortium, June 6, 1999.
http://www.salutation.org/whitepaper/Jini-UPnP.PDF

[13] Ryan Troll, "Automatically Choosing an IP Address in an Ad-Hoc IPv4 Network," IETF Draft draft-ietf-dhc-ipv4-autoconfig-05.txt, March 2, 2000. http://www.ietf.org/internet-drafts/draft-ietf-dhc-ipv4-autoconfig-05.txt

[14] Bluetooth Consortium, "Specification of the Bluetooth System Core Version 1.0 B: Part E, Service Discovery Protocol (SDP)," Nov 29, 1999.
http://www.bluetooth.com/developer/specification/specification.asp

[15] Bluetooth Consortium, "Specification of the Bluetooth System Profiles Version 1.0 B: Part K:2, Service Discovery Application Profile," Dec 1, 1999.
http://www.bluetooth.com/developer/specification/specification.asp

[16] IETF SVRLOC Working Group, "Service Location Protocol Home Page," http://www.srvloc.org

[17] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," IETF RFC 2608, June 1999. http://www.ietf.org/rfc/rfc2608.txt

[18] E. Guttman, C. Perkins, and J. Kempf, "Service Templates and Service: Schemes," IETF RFC 2609, June 1999. http://www.ietf.org/rfc/rfc2609.txt

[19] Rekesh John, "UPnP, Jini and Salutation – A look at some popular coordination frameworks for future networked devices," California Software Labs, June 17, 1999.
http://www.cswl.com/whiteppr/tech/ upnp.html

[20] IETF ZEROCONF Working Group, "Zero Configuration Networking (zeroconf)," http://www.ietf.org/html.charters/zeroconf-charter.html

[21] INS: Intentional Naming System, http://wind.lcs.mit.edu/projects/ins

[22] The Berkeley Service Discovery Service, http://www.cs.berkeley.edu/ ~caerwin/sds-project.html

[23] R. Droms, "Dynamic Host Configuration Protocol," IETF RFC 2131, March 1997. http://www.ietf.org/rfc/rfc2131.txt

[24] Sun Microsystems, "Java 2 Platform Midcro Edition (J2ME) Technology for Creating Mobile Devices," White Paper, May 19, 2000. http://java.sun.com/products/cldc/wp/KVMwp.pdf

[25] World Wide Web Consortium, "Extensible Markup Language (XML)", http://www.w3.org/XML