A COMPARISON OF SERVICE DISCOVERY PROTOCOLS AND IMPLEMENTATION OF THE SERVICE LOCATION PROTOCOL

Christian Bettstetter and *Christoph Renner*
Technische Universität München (TUM)
Institute of Communication Networks
D–80290 Munich, Germany
Christian.Bettstetter@ei.tum.de

Abstract

With the raising number of Internet services, automatic service discovery will be a very important feature in future network scenarios, e.g., in self organizing ad hoc networks. With service discovery, devices may automatically discover network services including their properties, and services may advertise their existence in a dynamic way. This paper compares some well–known service discovery protocols currently under development, namely the Service Location Protocol (SLP), Jini, Salutation, Universal Plug and Play (UPnP), and the Bluetooth Service Discovery Protocol (SDP). Application scenarios for service discovery are presented, which emphasize the importance of these protocols. We also present our SLP beta implementation, which includes the fundamental protocol transactions and demonstrates IP service discovery in action.

Keywords: Service discovery, service location protocol, SLP, IP autoconfiguration, ad hoc communication.

1 INTRODUCTION AND SCENARIOS

The number of services that will become available in networks (in particular in the Internet) is expected to grow enormously. Besides classical services, such as those offered by printers, scanners, fax machines, and so on, more and more totally new services will be available. Examples are services for information access via Internet, music on demand, and services that use computational infrastructure that is being deployed within the network.

Following this trend, it becomes increasingly important to give users the possibility of finding and making use of services that are available in a network. What is needed is a functionality that enables users to effectively search for available services that are appropriate to solve a given task. Ideally, users would like to obtain access to services automatically, without requiring that they must re–configure their system. For example, they do not want to search for the IP address of the desired service or manually upload device drivers. Especially with the widespread deployment of network–enabled mobile devices (such as notebooks, PDAs, and enhanced cellular phones), dynamic discovery of services in a visited foreign network and automatic system configuration will be very useful features.

This task is addressed by newly emerging *service discovery protocols*, like SLP (Service Location Protocol), Jini, UPnP (Universal Plug and Play), and Salutation. In a service discovery environment, services advertise themselves, supplying details about their capabilities and information one must know to access the service (e.g., the IP address). Clients (e.g., word processing software) may locate a service by its service type (e.g., printer) and may make an intelligent service selection in case multiple services of the desired type are available.

To summarize, service discovery protocols provide mechanisms for dynamically discovering available services in a network and for providing the necessary information to:

- search and browse for services,
- choose the right service (with desired characteristics), and

- utilize the service.

From the user's point of view, service discovery greatly simplifies the task of finding and using services. From the network administrator's point of view, service discovery simplifies the task of building and maintaining a network, especially to introduce new services and new devices.

Let us illustrate the usefulness of service discovery with the following scenario: A journalist reports from a sports event. She carries her notebook in order to write and print out articles, send emails, and get information from the Web. She attaches her notebook to the local area network in the press room in order to access the Internet and use local resources such as network printers and scanners. We have a typical service discovery problem now: Unless someone tells her the name and type of the printer and uploads the corresponding driver on her notebook, she will not be able to print out anything. She also does not know whether the printer supports color or not. Moreover, for Internet access she will have to re–configure her notebook with a valid IP address, subnet mask, default router, and DNS (domain name service) server. Furthermore, the email settings may be re–configured to use the local mail server. The utilization of service discovery would enable her to automatically detect, select, and utilize these services. Service discovery would also inform her about the attributes of the printer, e.g., color support and paper format.

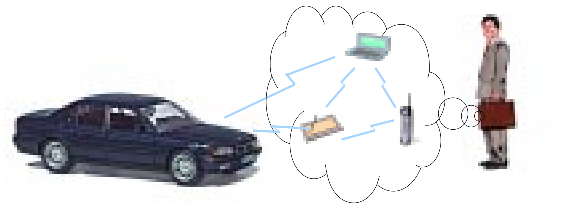

Figure 1: Wireless devices automatically discover services offered by the car.

The problem of service discovery is also relevant for the car environment (see Figure 1) [1]. Passengers could bring network–enabled devices into the car, connect them to the car area network, and use equipment that is installed inside the car. There could be various kinds of equipment installed in the car. For example, in a "mobile office car" there would be a fax machine, a printer, a hard disc, and a color display.

Service discovery also plays an essential role in ad hoc communications, where no fixed infrastructure is present but the nodes themselves form the network. Bluetooth is an example for wireless ad hoc technology. In Figure 1, the mobile phone, the notebook, and the graphic tablet could form an ad hoc network via Bluetooth links. Since there is no administrative control in such a network, devices must be self organizing, in particular self configuring. Mobile devices that take part in an ad hoc network may also offer services (or serve as access points to services), and other devices must discover these services. For example, the notebook in Figure 1 may offer a "translation" service to the mobile phone, and the phone may offer a "Web search" service via its GPRS (General Packet Radio Service) air interface. Due to the very dynamic nature of such a network (Devices just link together spontaneously and them move away again.), static service configuration makes no sense at all, but dynamic and automatic service discovery functionality is required.

This paper compares the most important service discovery protocols currently under development. Section 2 gives a brief overview of these protocols. Next, in Section 3, we compare them in more detail, according to their functionality, available implementations, and dependency on operating system, platform, and network transport. In Section 4, we present our beta implementation of SLP, and Section 5 shows which mappings between the protocols are defined for interoperation. Finally, Section 6 concludes this paper.

2 SERVICE DISCOVERY PROTOCOLS

As the computer networking community realized the need for service discovery a few years ago, several companies, consortiums, and an IETF (Internet Engineering Task Force) working group started to do research in this field. A variety of service discovery protocols is currently under development. The most well known so far are:

- Service Location Protocol (SLP), developed by the IETF;
- Jini, which is Sun's Java–based approach to service discovery;
- Salutation;
- Microsoft's Universal Plug and Play (UPnP); and the
- Bluetooth Service Discovery Protocol (SDP).

Figure 2 shows the companies that actively contribute to the development of these protocols. In the following, we briefly discuss the fundamental architecture of these protocols and interactions between the instances for service discovery, service registration, and service advertisement. In the next section we then compare the protocols in more detail.

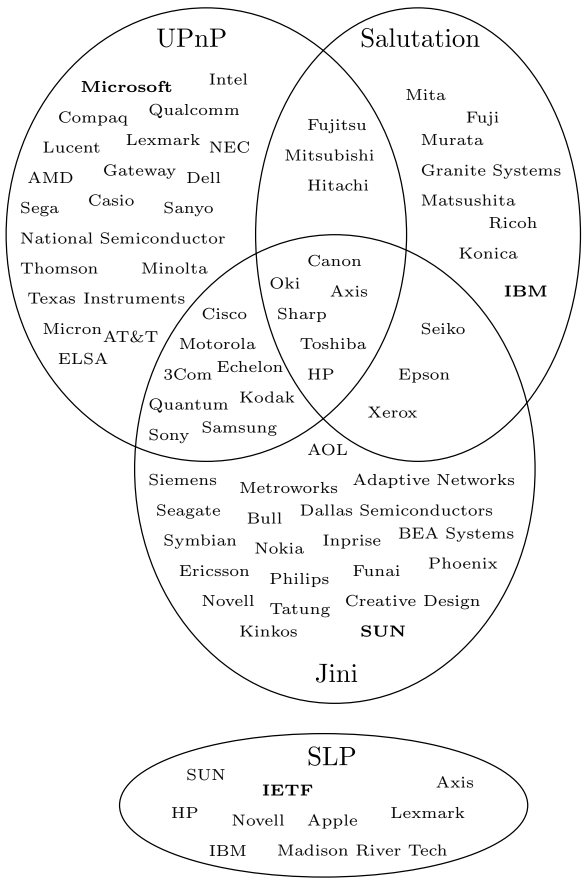

Figure 2: Companies involved in the development of Jini, Salutation, UPnP, and SLP

2.1 Service Location Protocol (SLP)

The Service Location Protocol (SLP) [2] is being developed by the IETF SvrLoc working group and is currently available in Version 2 [3]. SLP aims to be a vendor–independent standard. It is designed for TCP/IP networks and is scalable up to large enterprise networks.

The SLP architecture consists of three main components:

- *User Agents (UA)* perform service discovery, on behalf of the client (user or application);
- *Service Agents (SA)* advertise the location and characteristics of services, on behalf of services; and
- *Directory Agents (DA)* collect service addresses and information received from SAs in their database and respond to service requests from UAs.

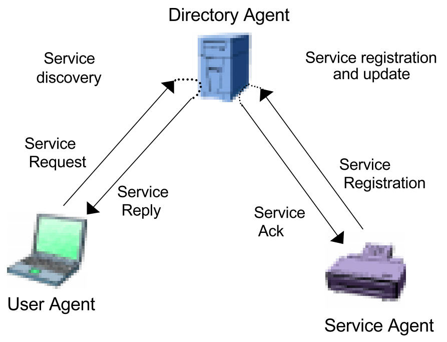

Figure 3: SLP agents and their transactions for service discovery and registration

Figure 3 shows the interactions between the three agents. When a new service connects to a network, the SA contacts the DA to advertise its existence (*Service Registration*). When a user needs a certain service, the UA queries the available services in the network from the DA (*Service Request*). After receiving the address and characteristics of the desired service, the user may finally utilize the service.

Before a client (UA or SA) is able to contact the DA, it must discover the existence of the DA. There are three different methods for DA discovery: static, active, and passive. With static discovery, SLP agents obtain the address of the DA through DHCP (Dynamic Host Configuration Protocol [4]). The necessary DHCP options for SLP are defined in [5]. DHCP servers distribute the addresses of DAs to hosts that request them. In active discovery, UAs and SAs send service requests to the SLP multicast group address (239.255.255.253). A DA listening on this address will eventually receive a service request and respond directly (via unicast) to the requesting agent. In case of passive discovery, DAs periodically send out multicast advertisements for their services. UAs and SAs learn the DA address from the received advertisements and are now able to contact the DA themselves via unicast.

It is important to note that the DA is not mandatory. In fact, it is used especially in large net-

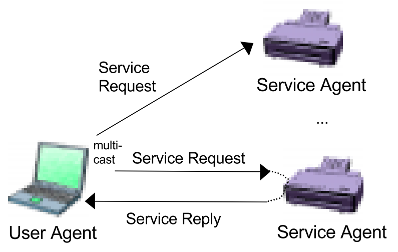

Figure 4: Service discovery without DA

works with many services, since it allows to categorize services into different groups (scopes). In smaller networks (e.g., home or car networks) it is more effective to deploy SLP without a DA. SLP has therefore two operational modes, depending on whether a DA is present or not. If a DA exists on the network (as shown in Figure 3), it will collect all service information advertised by SAs. UAs will send their *Service Requests* to the DA and receive the desired service information. If there is no DA (see Figure 4), UAs repeatedly send out their *Service Request* to the SLP multicast address. All SAs listen for these multicast requests and, if they advertise the requested service, they will send unicast responses to the UA. Furthermore, SAs multicast an announcement of their existence periodically, so that UAs can learn about the existence of new services.

Services are advertised using a *Service URL* and a *Service Template* [6]. The Service URL contains the IP address of the service, the port number, and path. Service Templates specify the attributes that characterize the service and their default values. A Service Template associated with a network printer could look like the following:

```
service:printer://lj4050.tum.de:1020/queue1
scopes = tum, bmw, administrator
printer-name = lj4050
printer-model = HP LJ4050 N
printer-location = Room 0409
color-supported = false
pages-per-minute = 9
sides-supported = one-sided, two-sided
```

SLP Version 1 (defined in [7]) has been implemented in several commercial products, for example in Hewlett Packard's JetSend Technology (which supports printers, digital cameras, scanners, projectors, and the PDA platforms Windows CE and Palm; see *http://www.jetsend.hp.com*). SLPv2 is expected to be widely deployed as well. It is already included in Solaris 8 and HP's Web JetAdmin. Two SLP reference implementations may be downloaded from *http://www.srvloc.org/*. For detailed information on SLP we recommend the book by J. Kempf and P. Pierre [8].

2.2 Jini

Jini technology is an extension of the programming language Java and has been developed by Sun Microsystems. It addresses the issue of how devices connect with each other in order to form a simple ad hoc network (a Jini "community"), and how these devices provide services to other devices in this network. Jini consists of an architecture and a programming model.

Each Jini device is assumed to have a Java Virtual Machine (JVM) running on it. The Jini architecture principle [9] is similar to that of SLP. Devices and applications register with a Jini network using a process called *Discovery and Join*. To join a Jini network, a device or application places itself into the *Lookup Table* on a lookup server, which is a database for all services on the network (similar to the DA in SLP). Besides pointers to services, the Lookup Table in Jini can also store Java–based program code for these services. This means that services may upload device drivers, an interface, and other programs that help the user to access the service. When a client wants to utilize the service, the object code is downloaded from the Lookup Table to the JVM of the client. Whereas a service request in SLP returns a Service URL, the Jini object code offers direct access to the service using an interface known to the client. This code mobility replaces the necessity of pre–installing drivers on the client.

The Jini specifications are open source and may be used freely. However, Sun charges a licensing fee for commercial use. A reference implementation may be downloaded at *http://www.sun.com/jini*. The Jini code can be implemented in 46K of Java binaries.

2.3 Salutation

Salutation is another approach to service discovery. The Salutation architecture [10] is being developed by an open industry consortium, called the Salutation Consortium (see *http://www.salutation.org*). The Salutation architecture consists of *Salutation Managers (SLMs)* that have the functionality of service brokers. Services register their capabilities with an SLM, and clients query the SLM when they need a service. After discovering a desired service, clients are able to request the utilization of the service through the SLM.

Salutation is a rather settled approach, with some commercial implementations, including fax devices and Windows enablers (95/98 and NT), e.g., from IBM and Axis. An example is IBM's NuOffice, a Salutation enhancement for Lotus Notes. Further implementations for Palm OS and Windows CE are planned for the near future.

2.4 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) is being developed by an industry consortium (see *http://www.upnp.org*), which has been founded and is lead by Microsoft. One can say that it extends Microsoft's Plug and Play technology to the case where devices are reachable through a TCP/IP network. Its usage is proposed for small office or home computer networks, where it enables peer–to–peer mechanisms for auto–configuration of devices, service discovery, and control of services. In UPnP's current version (release 0.91) there is no central service register, such as the DA in SLP or the lookup table in Jini. The Simple Service Discovery Protocol (SSDP) [11] is used within UPnP to discover services. SSDP uses HTTP over UDP and is thus designed for usage in IP networks. For a description of the UPnP system architecture refer to [12].

2.5 Bluetooth Service Discovery Protocol (SDP)

Bluetooth is a new short range wireless transmission technology. The Bluetooth protocol stack contains the Service Discovery Protocol (SDP), which is used to locate services provided by or available via a Bluetooth device. SDP is described in the Bluetooth specification part E [13]. It is based on the Piano platform by Motorola and has been modified to suit the dynamic nature of ad hoc communications. It addresses service discovery specifically for this environment and thus focuses on discovering services, where it supports the following inquiries: search for services by service type; search for services by service attributes; and service browsing without *a priori* knowledge of the service characteristics. SDP does not include functionality for accessing services. Once services are discovered with SDP, they can be selected, accessed, and used by mechanisms out of the scope of SDP, for example by other service discovery protocols such as SLP and Salutation (see, e.g., [14], which shows a mapping from Salutation to SDP). SDP can co–exist with other service discovery protocols, but it does not require them.

Note that in [15] we present our SDL (Specification and Description Language) description of SDP.

3 A COMPARISON OF SLP, JINI, SALUTATION, AND UPNP

Let us now compare the different approaches for service discovery. Although all protocols are using similar architectures, there are several differences. Table 1 summarizes the main features of the four service discovery protocols SLP, Jini, Salutation, and UPnP.

UPnP Universal Plug and Play is the youngest of these protocols, and it is still in an early state of development. Up to now there do not exist any commercial implementations of UPnP, but Microsoft plans to implement it for all Windows platforms. The specifications and a sample source code are available freely. UPnP is designed for TCP/IP networks only. In its current version, it does not allow clients to search for service attributes (as, e.g., SLP is able to).

As shown in Figure 2, UPnP is supported by a large number of companies. Among them are many global players in Internet and telecommunications. This will probably make UPnP a successful approach in a couple of years.

Salutation Service discovery in Salutation is defined on a higher layer, and the transport layer is not specified. Thus, Salutation is independent on the network technology and may run over multiple infrastructures, such as over TCP/IP and IrDA. It is not limited to HTTP–over–UDP–over–IP, as UPnP is. Moreover, Salutation is independent on the programming language, i.e., it is not limited to nor does it have a prerequisite for Java (as Jini). Its major advantage compared to UPnP and Jini is that there already exist commercial implementations.

Jini Jini is also a rather new approach and no products are in the market yet. Jini distinguishes from the other approaches mainly by the fact that it is based on Java.

On the one hand, this concept makes Jini independent of the platform and operating system to run on. Most important, Jini uses Java Remote Method Invocation (Java RMI) protocols to move

Feature	SLP	Jini	Salutation	UPnP
Developer	IETF	Sun Microsystems	Salutation Consortium	Microsoft
License	open source	open license, but fee for commercial use	open source	open (only for members)
Version	2	1.0	2.1	0.91
Network transport	TCP/IP	independent	independent	TCP/IP
Programming language	independent	Java	independent	independent
OS and platform	dependent	independent	dependent	dependent
Code mobility	no	yes (Java RMI)	no	no
Srv attributes searchable	yes	yes	yes	no
Central cache repository	yes (optional)	optional using SLP	yes (optional)	no
Operation w/o directory	yes	Lookup Table required	yes	–
Leasing concept	yes	yes	no	yes
Security	IP dependent	Java based	authentication	IP dependent

Table 1: Comparison of Service Discovery Protocols

program code around the network. This introduces the possibility to move device drivers to client applications, which is its main advantage over the non–Java based service discovery concepts.

On the other hand, the fact that Jini is tightly tied to the programming language Java makes it dependent on the programming environment. It also requires its devices to run a JVM, which consumes memory and processing power. This can be a hard requirement for large device drivers and might not be fulfilled in embedded systems.

Due to the dynamic nature of ad hoc networks, Jini employs the concept of *leasing*. Each time a device joins the network and its services become available on the network, it registers itself only for a certain period of time, called a lease. This is especially useful for very dynamic ad hoc network scenarios.

SLP The Service Location Protocol is standardized and well documented through the IETF, and there exist several reference implementations as well as commercial products.

SLP offers a flexible and scalable architecture and the utilization of service templates make service browsing and human interaction possible. Since SLP is able to operate with or without a DA, it is suitable for networks of different sizes, ranging from very small ad hoc connectivity to large enterprise networks. SLP also includes a leasing concept with a lifetime that defines how long a DA will store a service registration. Whereas Jini is dependent on Java, SLP is independent on the programming language.

The SvrLoc working group is working actively on improving the protocol. DHCP options to configure SLP are already defined. Under development is a concept to use LDAP (Lightweight Directory Access Protocols) servers as a back–end for SLP, i.e., DAs may use LDAP servers as their repository [16]. Furthermore, SLP will be adapted to use with IPv6 [17] and DHCPv6 [18].

Last but not least, SLP is developed by an open and vendor–independent forum and its implementation is freely available. We expect SLP to play a major role in service discovery.

4 OUR SLP IMPLEMENTATION

Since SLP seems to be a very promising approach to service discovery, we have implemented the basic functionality of SLPv2 in ANSI C using UDP sockets. The aim of our implementation has been to demonstrate service discovery in action. We have implemented two different architectures, one with a DA and one without DA, and two monitoring tools.

Architecture with DA This architecture consists of one or more UAs, one DA, and one or more SAs.

The service agent sa is configured with a service template stored in a text file. The template contains the service URL, the scope, and attributes of the service. Once an sa is started, it performs active DA discovery (as explained in Section 2.1) to locate a DA. On response of the DA, it extracts the DA's IP address and then sends a *Service Registration* message to the DA. After receiving a *Service Ack* message from the DA, sa terminates.

The directory agent da announces its presence via IP multicast upon startup. It is then ready to process *Service Registrations* from SAs and *Service Requests* from UAs, as well as answer messages for active DA discovery. Whenever da receives a *Service Registration* message it adds the service to be registered to its database and replies with a *Service Ack*. The database is implemented as a text file.

The user agent ua must be initialized with the attributes of the service it should discover, namely the service type, scope, and optionally desired attributes. This is done via command line options. For example `ua service:printer: tum color=true` requests all color printer services in the scope tum. Moreover, ua has two modes of operation: Mode 1 sends out a *Service Request*, and mode 2 sends out an *Attribute Request*, which will return all attributes of the desired service. Mode 1 is used to search for a service with specific attributes, whereas mode 2 is the basis for service browsing. The ua terminates after receiving either a *Service Reply* or an *Attribute Reply*, respectively.

Architecture without DA The user and service agents in this architecture are titled mini_ua and mini_sa, respectively. The mini_sa does not register its services with a directory, but advertises its existence via IP multicast. On receiving a *Service Request* or *Attribute Request*, mini_sa searches in its service template and responds to the requesting mini_ua. The functionality of mini_ua is the same as in the architecture with DA with the difference that mini_ua listens to advertisements of SAs.

SLP Monitoring Tools We have also developed two monitoring tools, which help us to better understand the SLP protocol transactions and serve as testing tools for further implementation work. The slp_listener listens to the SLP multicast group and outputs all SLP messages. The slp_talker is a tool to generate any of the SLPv2 messages and to send them to an IP address of a specify host or to the SLP IP multicast group. It can be used to test the functionality of SLP agents.

Further Information Besides the basic SLPv2 message transactions, we have also implemented additional SLPv2 features mentioned in the standard [3], such as transaction identifiers XID, stateless boot time stamp, previous responser's list, scopes, language tag, and attribute query. Detailed information about our implementation is available at *http://www.lkn.ei.tum.de/~chris/slp/*.

Test environment Our test environment currently consists of two Linux PCs and two Linux notebooks connected via a wireless LAN (IEEE 802.11 Lucent WaveLAN) as well as two Linux PCs connected to the WaveLAN base station via fixed Ethernet. All computers can act as DA, SA, or UA.

5 BRIDGES BETWEEN SERVICE DISCOVERY PROTOCOLS

The variety of different service discovery protocols makes it important to have bridges between the methods to enable service discovery also between devices that do not run the same service discovery protocol. For example, we would like to discover SLP printers with Salutation devices.

We have already mentioned the mapping from SLP and Salutation to Bluetooth SDP, but there exist several other mappings. Proxied Jini devices, for example, may also be discovered via SLP [19]; the Java version of Salutation–Lite can emulate the functions of Jini [20]; and release 2.1 of the Salutation specification demonstrates a mapping between Salutation and SLP.

6 CONCLUSIONS AND OUTLOOK

Service discovery will be an important feature in future network scenarios, e.g., in self organizing ad hoc networks. With service discovery, devices may automatically discover network services including their properties, and services may advertise their existence in a dynamic way. From the user's point of view, service discovery is about "coming to an unknown network environment with a mobile device and then detecting the available services according to one's need."

At present there exist a variety of service discovery protocols, most important SLP, Jini, Salutation, and UPnP. Also Bluetooth has a rather simple service discovery protocol. We have compared these approaches and listed their advantages and drawbacks. Each of these protocols approaches the vision of service discovery from a different perspective. We expect that also in the future there will be various kinds of service discovery protocols. This makes it important to have bridges between the different protocols to enable service discovery with various devices.

To test service discovery in action, we have developed a beta implementation of SLP, with the mandatory protocol transactions. Currently, we are improving our SLP testbed with enhanced protocol functions, such as optional messages and leasing functionality. In our testbed, we will also use the WaveLAN "ad hoc mode" for direct peer–to–peer connection between the computers. Moreover, we plan to implement a service browser with a graphical user interface.

Acknowledgments

The authors would like to thank Wolfgang Kellerer (TUM) for several useful discussions and comments.

References

[1] Christian Bettstetter. Toward Internet–based Car Communications: On Some System Architecture and Protocol Aspects. In *Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School*, Twente, Netherlands, September 2000.

[2] Erik Guttman. Service Location Protocol: Automatic Discovery of IP Network Services. *IEEE Internet Computing*, 3(4):71–80, July 1999.

[3] Erik Guttman, Charles E. Perkins, and Michael Day. Service Location Protocol, Version 2. Internet RFC 2608, June 1999.

[4] Ralph Droms. Automated Configuration of TCP/IP with DHCP. *IEEE Internet Computing*, 3(4):45–53, July 1999.

[5] Charles E. Perkins and Erik Guttman. DHCP Options for Service Location Protocol. Internet RFC 2610, June 1999.

[6] Erik Guttman, Charles E. Perkins, and James Kempf. Service Templates and Service: Schemes. Internet RFC 2609, June 1999.

[7] John Veizades, Erik Guttman, Charles E. Perkins, and Scott Kaplan. Service Location Protocol. Internet RFC 2165, June 1997.

[8] James Kempf and Pete St. Pierre. *Service Location Protocol for Enterprise Networks*. Wiley, 1999.

[9] Sun. Technical White Paper: Jini Architectural Overview. http://www.sun.com/jini/, 1999.

[10] Salutation Consortium. White Paper: Salutation Architecture: Overview. http://www.salutation.org/whitepaper/ originalwp.pdf, 1998.

[11] Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu, and Shivaun Albright. Simple Service Discovery Protocol. Internet Draft, draft-cai-ssdp-v1-03.txt, October 1999.

[12] Universal Plug and Play Forum. Universal Plug and Play Device Architecture. Version 0.91, March 2000.

[13] Bluetooth Specification Part E. Service Discovery Protocol (SDP). http://www.bluetooth.com, November 1999.

[14] Brent Miller and Robert Pascoe. Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer. http://www.bluetooth.com, July 1999.

[15] Christian Schwingenschlögl and Anton Heigl. Development of a Service Discovery Architecture for the Bluetooth Radio System. In *Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School*, Twente, Netherlands, September 2000.

[16] James Kempf, Ryan Moats, and Pete St. Pierre. Conversion of LDAP Schemas to and from SLP Templates. Internet Draft, draft-ietf-svrloc-template-conversion-05.txt, October 1999.

[17] Erik Guttman. Service Location Protocol Modifications for IPv6. Internet Draft, draft-ietf-svrloc-ipv6-08.txt, January 2000.

[18] Jim Bound, Mike Carney, and Charles E. Perkins. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). Internet Draft, draft-ietf-dhc-dhcpv6-14.txt, May 2000.

[19] Manfred Bathelt and Jochen Nickles. Das Plug and Play der Automatisierung. *Elektronik (Volume 6)*, pages 54 – 62, March 1999.

[20] Robert A. Pascoe. The Salutation Consortium Newsletter. April 1999.