

Comparison of Service Discovery Protocols

Blerta Bishaj

Helsinki University of Technology

bbishaj@cc.hut.fi

Abstract

The proliferation of devices in the home environment, offering many services, has introduced the need for relieving configuration efforts from users, and for offering seamless exchange of services between devices. Several architectures have been proposed, huge effort and money have already been invested into them. This paper overviews these initiatives with the focus on determining their suitability for home environment and on foreseeing the prevalence of any of them in the market. The purpose is to get insight into future developments in this field.

KEYWORDS: Home networking, Service Discovery Protocol (SDP), leasing, Java Virtual Machine (JVM), Service Location Protocol (SLP)

1 Introduction

An increasing number of devices is being used in home environments, and each device offers several functionalities. A future family is expected to have home entertainment and home automation devices, mobile phones, computers, laptops, printers, etc. In a home where devices interact, we might have, for example, an alarm clock connected to the heating system, so that the house is warm when people wake up. Furthermore, devices might be accessed and controlled from one single component. Another future scenario in this field might be that devices be accessed from the internet.

Communication between devices in networks is usually not supported, or is a very basic one. The prospect of interaction and sharing of services and functionalities between devices has aroused particular interest. Currently, the prevalent network architecture is the client-server one, which needs professional administration and fails the needed flexibility for home environments. For example, the addition of new devices to the network requires installing the drivers and often also rebooting. Instead, devices should be made to join the network by simply plugging them in; thereafter, they should be able to interact with other devices already present in the network. In order to facilitate the loose interaction of devices, the architecture has to change: from configured environments to networks with unknown infrastructures.

There are several architectures proposed for service discovery. They provide mechanisms for searching and browsing of the services, for choosing the right service, and for utilizing it. Yet, each one has its own approach to service discovery. The most important proposals are: Jini, Universal Plug and Play (UPnP), Salutation, Service Location Proto-

col (SLP), Home Audio Video Interoperability (HAVi), and the Bluetooth SDP. This paper presents a general overview of the currently available technologies for service discovery, and a comparison between them. Furthermore, we have also made an attempt at predicting future prevalence, in the market, of any of the current solutions. There are also other protocols that have not been analysed in this paper. Framework for Robust and Resource-aware Discovery (FRODO) [12], for example, is another service discovery approach for the home environment, whereas Apple has come forward with the Bonjour SDP.

The structure of the paper is as follows. Section 2 contains an overview of the proposals in service discovery. More in detail, subsection 2.1 gives an overview of the Jini protocol. Subsection 2.2 examines the UPnP protocol. Subsection 2.3 focuses on the Salutation protocol. Further, SLP is discussed in subsection 2.4. Next, subsection 2.5 overviews the HAVi protocol, while subsection 2.6 focuses on the Bluetooth SDP. Section 3 provides a comparison between the protocols. Finally, section 4 concludes this paper, with some remarks on possible future developments.

2 Architectures

This section gives an overview of each of the main solutions that have been proposed for service discovery.

2.1 Jini

Jini [8] is a network architecture for distributed systems; it is developed by Sun Microsystems in Java. The aim of this architecture is to make the network dynamic and self-administered, where services are added and deleted in a flexible manner. More precisely, the purpose is to end up with a system where users are able to share services and resources in a network; where users have easy access to the services, even though the user's location may change; where building, maintaining, and changing devices, software, or users is made simple. In a way, Jini can be considered as an extension of the Java application environment from a single machine to the whole network. The Java environment, on which Jini is based, offers built-in security for executing code from another machine. Jini adds functionality on top of that to support the moving of components in a distributed system, as compared to the easy movement of objects in a Java application environment. The purpose is to have a monolithic system with simple access, easy administration, and support for sharing, while preserving flexibility and control similar to that in a single workstation [8].

Jini makes the assumption that the devices connected to the network have a certain memory capacity and processing power. The limitation on the devices that can be connected directly to the network is a Java legacy; the devices need to have the JVM. Other devices not meeting this criteria need to be presented to the network by means of a proxy, which is a piece of hardware and/or software that meets the abovementioned requirements. The service proxy has the drawback that in order to have no need for drivers, manufacturers must agree to a common interface. This is hard to achieve for every kind of device, and is exacerbated by the fact that each device tends to encompass multiple and different functionalities. There is benefit from the JVM: it makes Jini platform-independent, but the JVM is heavy for handheld devices and embedded systems. As for Java, Jini depends on the Java application environment, rather than on the Java programming language. Any language, as claimed, producing compliant bytecodes can be used. However, practically, only the Java language is used. Therefore, Jini is actually language-dependent, though it is maintained not to be.

Services are crucial to the Jini architecture [8, 11]. They can be used by a person, a program, or another service. Some examples of services are: storage, a computation, a hardware device, a user, devices such as printers, software such as applications, information such as databases. A Jini system is made up of services that can be collected in order to complete a particular task. A service can use another service, a client may be a service for another client. A service protocol is used for the communication between services. The one Jini offers is a set of interfaces that define critical service interaction between services. It can be extended further.

A lookup service is used to find and resolve services. What it does is a mapping of the interfaces that indicate the functionality of a service to sets of objects implementing the service. A service has to be registered in at least one lookup service. A Jini service provider registers itself with every discovered lookup service [3]. An object can itself consist of other services, this makes room for hierarchical services and lookup. Furthermore, objects may also be the encapsulation of other naming or directory services. References to the Jini lookup system can also be placed in other naming and directory services. In this way, bridging between the Jini lookup system and other forms of lookup service is created.

The discovery, join and lookup protocols are the heart of the Jini architecture [8]. The discovery protocol is used when a device looks for a lookup service to register with. The join protocol is used when a service has located the lookup service and wants to join it. The lookup protocol is used when a client needs to locate and invoke a service. Discovery/join is how a service is added to the system. The service provider multicasts a request for local lookup services to identify themselves (discovery). After this, a service object is loaded into the lookup service (joining). This service object has the Java interface for the service including the methods of the interface to be invoked for using the service. There can also be other descriptive attributes. The lookup process ensures that a copy of the service object is loaded into the client. The client can now use the service.

Java Remote Method Invocation (RMI) can be used for the communication between services [8, 11]. RMI basically

implements the remote procedure call mechanisms in Java. It allows data as well as objects to be passed through the network.

As for security, there are two important concepts: a principal and an access control list. The principal is the entity that accesses a Jini service and that can be traced back to a system user. Services can make use of other services by using the identity of the object that offers this service. Each object has an access list that is checked against in order to allow or not access to its services.

In order to regulate the access to many of the services in the Jini architecture, leases are used [8]. Leasing is part of the service protocol; it is a result of the negotiation between the provider and the user of the service for the allocation and freeing of the resource. The lease is time-based and requires renewal in order to prolong validity. Leases can be exclusive or not, not-exclusive ones allow resource sharing among multiple users. Leasing has also another advantage. If the network goes down for shorter a period than the lease time, the resource allocation will still be valid afterwards, until the lease time expires [2, 5].

With Jini, administration is still needed, because when a device is introduced to the network, it needs to be assigned an IP. Jini supports distributed events: an object can register its interest in events of another object and be notified whenever these events happen. This adds reliability guarantees to the architecture. Interfaces to devices have to be implemented for the Jini architecture to work. This is a disadvantage for Jini, since other SDPs have them already available for consumer electronics.

2.2 UPnP

UPnP [9] is an industry initiative by Microsoft to provide simple, robust, peer-to-peer connectivity among devices and PCs. Plug and Play (PnP) made it easier to setup, configure, and add peripherals to a PC, while the purpose of UPnP is to extend this simplicity throughout the network, enabling discovery and control of devices. Its target is zero-configuration, invisible networking, as well as automatic discovery for the devices of many vendors. UPnP encompasses many existing, as well as new scenarios, such as home automation, printing, audio/video entertainment, kitchen appliances, etc. UPnP is independent of operating systems, programming languages, or physical media.

The basic components of a UPnP network are: devices, services, and control points. A UPnP device contains services and nested devices. An XML device description document is hosted by each device; this document lists the set of services and properties of the device. A service is the smallest unit of control in UPnP. It is described in the XML document of the device, which also contains a pointer to the service description. A service consists of a state table, a control server, and an event server. The state table controls the state of the service through state variables. The control server updates the state according to action requests. The event server notifies interested subscribers when the service state changes. A controller is capable of discovering and controlling other devices. For true peer-to-peer functionality, devices should incorporate control point functionality. There

is no central registry in UPnP, at least not necessarily [3, 5].

UPnP uses many existing, standard protocols, so that UPnP devices can fit seamlessly and without effort into the existing networks [9]. TCP/IP is the base on which the UPnP protocols are built, as well as many protocols that go with it, such as UDP, ARP, DHCP and DNS. HTTP is a core part of UPnP. All UPnP aspects are based on HTTP or its variants.

The Simple Service Discovery Protocol (SSDP) defines how to find network services [9]. It is both for control points to locate services on the network, as well as for devices to announce their availability. A UPnP control point, when booting up, can send a search request to discover devices and services. UPnP devices, on the other hand, listen on the multicast port. If the search criteria match, a unicast reply is sent. In the same way, a device, when plugged in, will send out multiple SSDP presence announcements. Apart from the leasing concept that UPnP also shares, SSDP provides a way for a device to notify that it is leaving the network.

The Generic Event Notification Architecture (GENA) defines the concepts of subscribers and publishers of notifications. The GENA formats are used to create these announcements that are sent using SSDP [14]. Simple Object Access Protocol (SOAP) [13] is used in UPnP to execute remote procedure calls, as well as to deliver control messages and return results or error messages to the control points. XML is used in UPnP in device and service descriptions, control messages and eventing. The advertisement message that a device issues contains a URL that directs to an XML file in the network, which describes the capabilities of the device [3].

The UPnP architecture defines a schema for creating device and service descriptions for any device or service type. Different committees create some standards for different devices and service types [9]. A vendor fills in this template with the information about the device. This data is then encapsulated in the UPnP-specific protocols. The service description and capabilities is done with XML, which is a very powerful feature of UPnP, in contrast to Jini, for example, where the description is short and in the service attributes.

Each UPnP device must be a DHCP client and search for a DHCP server when connected to a network. In the lack of a DHCP server it must use Auto IP to get an address: the device randomly chooses an address, and then makes an ARP request to see if it is already occupied. This mechanism minimizes administration requirements.

2.3 Salutation

Salutation [5] is a major cooperation architecture, developed by the Salutation Consortium. Salutation is nonproprietary. The aim of Salutation is to address the problems of service discovery and utilization among different appliances and equipment in an environment of mobility and diverse network technology.

The building block in the Salutation architecture is a Functional Unit. A collection of Functional Units defines a Service Record. For example, a fax service can comprise the [Print], [Scan], [Fax Data Send] Functional Units. Functional Units also have a descriptive attribute unit, meaning that more detailed information can be available about the

function offered.

The salutation architecture has two major components: Salutation Manager (SM) and Transport Manager (TM). The SM resembles the Jini lookup service; it registers a service provider and its capability. A service provider registers itself with the local or nearest SM. When a client asks its local SM for a service search, it may redirect the search to other SMs in the network. Hence, the search is performed in coordination among the SMs. This communication is done with RPC [5]. The TM acts as an interface between the SM and the communication technology. It offers reliable communication channels, despite underlying transport protocols. A TM can only support one network transport. Thus, TMs provide a transport-independent interface, which comprises service registration, service discovery, and service access function. In case the device is attached to several physically different networks, the SM will have several TMs and it need not deal with transport details.

A summary of the main tasks of the SM follows [5]:

Service Registry. The SM keeps a information about services in a registry. A client can register or unregister itself. This corresponds to the lookup service in Jini.

Service Discovery. A SM discovers other SMs and the services registered there. The communication between SMs is done with the SM Protocol. The result is a lookup service resembling the one Jini offers, but which, instead, is distributed over the network.

Service Availability. An application can ask from the local SM to check periodically the availability of services. The checking is performed between the local and the corresponding SM. This aim of this is the same as the remote event concept found for example in Jini.

Service Session Management. This management handles the service invocation after it has been discovered through service discovery. It can operate in three modes, which have varying control of this unit over the format and transport of the messages exchanged. This is a flexibility choice that the designers have made for Salutation. In case of no control, the messages can be of another format.

A lighter version of the protocol is also offered by the consortium, which is aimed at small and resource-restricted devices. Its target are devices with limited storage space, low communication bandwidth. This version is called Salutation-Lite. It waives some functions, which can be added as user options later and offers applicability to source-restricted devices, as well as to low bandwidth networks [7].

The transport independence of Salutation complicates the self-configuration issue. Like UPnP, it might use AutoIP in IP networks. To address security, Salutation uses user authentication. Salutation is a very good candidate for home environments. Having said that, it has to be mentioned that the consortium itself has dissolved with no apparent reason on June 30th, 2005 [15]. Also, the Salutation specifications, as well as its web page, are no more available. Therefore, the good prospects of the architecture are now in question.

2.4 SLP

SLP [5] is an IETF standard for service discovery. It enables the discovery and selection of a wide range of services ac-

cessible through an IP network. With SLP, a user needs only to search for a particular type of service, and optionally for attributes associated with it.

It has three major software entities [5]: User Agent (UA), Service Agent (SA), and Directory Agent (DA).

The SA advertises the location and attributes of one or more services on behalf of them by using broadcasts. It also replies with IP unicasts to requests for these services. The services register and deregister with the SA. Each registration has a lifetime and reregistration is required periodically. It is the same leasing concept already discussed in previous protocols.

The UA receives requests from a client application and forwards multicast requests to SAs. Hence, there is little network overhead for SA discovery. The SA unicasts a response back with the service URL and possible attributes if there is a match with the registered services.

DA is a central information repository and it is optional. Its main function is to improve the performance of SLP. It can be thought of as a tier between the UAs and the SAs, which communicate with the DA instead of with each other. DAs relieve the network traffic from many multicast requests; the effect is more visible in large network, where multicast traffic can increase sharply because there are many SAs and UAs. A DA keeps the SA advertisements, and responds to UA requests. An SA registers itself with a DA. The registration contains the URL for the services, the lifetime, and descriptive attributes of the service. The registration should be refreshed periodically. A UA sends a request message to the DA, which in turn sends a message containing the URL of the service matched against the UA needs. SLP can work both with and without a DA: (1) with DA, information about the services is kept in the DA, so that the UA sends a query there for services; (2) without a DA, the UA multicasts service requests to the network, and a SA offering the service would reply back [5]. In small networks, there may be no real need for a DA.

SLP scales well in large networks; the reason is the minimal use of multicast messages and the fact that it can have multiple DAs. It has a flexible and scalable architecture, where service browsing and human interaction is possible. SLP offers filtered search for attributes and predicates, such as AND, OR, comparators, and substring matching [5]. SLP shares the concept of leasing with Jini and UPnP.

Services can be deployed in small networks without any special configuration or deployment. It works even if there is no DNS, DHCP, SLP DA, or routing. Therefore, home networks would benefit much from the automation of service discovery, because they often lack network administration. However, the architecture with DAs makes the system vulnerable to a single point of failure.

SLP is an open source, vendor-independent and already implemented [4]. It has the advantage of not depending on any programming language [5].

2.5 HAVi

HAVi [1] was founded by well-known electronics companies. It focuses on home entertainment devices, especially audio/video ones. This makes it a very good option for home

environments. HAVi enables peer-to-peer communication between devices for detection, querying, and control. Also, groups of devices can provide enhanced services by cooperation. However, given the wide range of electronics devices, and the presence of other non-HAVi devices, this architecture makes room for variations also.

HAVi classifies its devices into: Full AV devices (FAV), Intermediate AV devices (IAV), Base AV devices (BAV), and Legacy AV devices (LAV). FAV supports the entire HAVi architecture, IAV lacks the runtime environment, BAV contains only the basic features, and LAV is intended for integrating non-compliant devices.

In HAVi, each device is represented as a software element in the network, by the Device Control Module (DCM) [1]. The DCM contains a set of Functional Component Modules (FCM). Each FCM is responsible for a function of the device, while the DCM is the interface for accessing the functions of the device. Each device has self-describing data, which provide descriptive information about the device itself, as well as its capabilities.

The messaging system in HAVi has two functions: create unique identifiers for an object when it registers, handle the information exchange between the devices. The Registry is the element that acts as the directory service in HAVi. It keeps the information about objects and their attributes.

Another element is the 1394 Communication Media Manager, which is based on the IEEE 1394 standard [10]. This manager provides a mechanism for sending requests and receiving indications from remote devices. It also informs the Event Manager about changes in the network. The IEEE 1394 standard enables high data rates and QoS guarantees, which are important for the real-time traffic in AV devices. IEEE 1394 corresponds to the physical and link layers of the ISO network model. IP traffic can run over the IEEE 1394 standard: RFC 2734 and RFC 3146 specify how to run IPv4 and IPv6 over IEEE 1394.

The change of state of a software element is defined as an Event in HAVi. The Event Manager is the element with which software elements register in order to be notified of these events.

The element called Stream Manager configures the connections within a device, as well as connections between devices [1]. These streams go through the FCMs of the devices. The role of the manager is to handle the requests for connections, by allocating and managing the resources (especially the bandwidth). Therefore, HAVi provides QoS guarantees for multimedia streams. The Stream Manager can enable connection reestablishment after a reset, by using its information. Another element in HAVi is the Resource Manager, which deals with the own resources of a device, namely the use and release of the FCMs of a device.

HAVi addresses security in two distinct ways. The first way is through access levels, where APIs are divided into trusted and untrusted ones and the devices can decide whether or not to run untrusted ones. The other way uses the RSA scheme, with keys being released by the HAVi Certificate Authority (CA) to certain vendors. This is done so that only certified Java code can be run in the runtime environment.

The objective of HAVi is to expand the scope of the imple-

mentation to home control systems, security systems, communication systems and PC based applications. Besides, it is also trying to work out bridges with other SDPs. HAVi is independent of the platform or language of implementation [1].

2.6 Bluetooth SDP

Bluetooth is a transmission technology, meant for short-range (10m) wireless communication between low-power devices. Bluetooth devices form a personal area network, a piconet, with a maximum of 8 members. Groups of piconets communicating with each-other form a scatternet. Every Bluetooth SDP device has to implement a SDP server that provides services. The server has a list of service records, each with a list of attributes that represent different service classes. A Bluetooth SDP client sends a request message with the list of service classes.

Bluetooth is designed for Bluetooth environments, therefore, it offers limited functionality compared to other SDPs. Its functionalities are: search for services by service type; search for services by service attributes; and service browsing without a priori knowledge of the service characteristics. The Bluetooth SDP does not include functionality for accessing services. After services have been discovered with it, the selection, access, and usage can be done with mechanisms out of the scope of SDP, for example by other SDPs such as SLP and Salutation. SDP can coexist with other SDPs, but they are not a necessity [4].

The strong point of Bluetooth in the home environment is the lack of wires, while the short range it provides is a disadvantage. Also, it has a peer-to-peer connectivity, which does not scale well, because typically the systems lack resources. Another disadvantage is the lack of event notification when services become unavailable [5]. The Bluetooth security mechanisms offer either one-way, two-way, or no authentication. When authentication is used, the Bluetooth devices generate a secure connection with a pairing process that makes use of PIN codes entered by users. But for home environment usage, maybe security and privacy have to be addressed also in higher layers.

Bluetooth has huge implementation opportunities in home environments. Apart from cell phones and PCs which have already been implemented, there are other potentials. Home automation, for example, could replace doorbells. This would eliminate unnecessary wires around the place. Other goods, such as toys or entertainment devices, could be another area of implementation.

3 Comparison

Comparing different architectures is difficult, because they sometimes have few things in common. Nevertheless, the comparison here is structured by looking at them from different viewpoints. A summary of the comparisons is presented in Table 1.

License. SLP, Salutation, and HAVi are open source and their specifications are freely available. The other architectures are not open source. For UPnP, being a Microsoft initiative, it is not surprising. Jini is open source, but a fee is

required for commercial use. Bluetooth requires membership in the Bluetooth Special Group of Interest (SIG).

It seems like counter-productive for not open-source architecture developers to close the doors on possible device vendors wanting to jump on their bandwagon. Obviously, they are supposing that only vendors big enough to afford their fees deserve to enter, and that they would bring a real benefit to increasing the market share. On the other hand, this contributes to the polarization of the architectures in the market, which would cause the users to get stuck with one architecture. However, users will inevitably have items from different architectures, be it even only from legacy. To counter this, bridges between architectures are also being proposed, not because the developers want to, but because they have to.

Network transport. The architectures that are independent of the network transport are Jini, Salutation, Bluetooth SDP. HAVi is based on the IEEE 1394 standard which allows for higher data rates, and offers QoS. UPnP is TCP/IP based, because as a whole it is tries to solve the problem with the most popular protocols. SLP is also TCP/IP based.

Programming language. All of the architectures are independent of the programming language. Jini claims that any language producing the bytecodes for the virtual machine can be used. Practically, however, it can only be implemented with Java.

Central cache repository. From observing these architectures, it can be said that opposing the traditional client-server architecture, both client and server functionalities are put on the devices. Of course, the details of service negotiation differ in different architectures. They have made their decisions on how scalable they want their solutions to be, by introducing some central cache repositories. The purpose of these is to make the traffic scale well, especially in large networks. It is interesting to note that this is mainly a feature of the open source initiatives. Namely, SLP and Salutation offer it, Jini also, if it cooperates with SLP. On the other hand, UPnP, HAVi, and Bluetooth do not have this feature. UPnP has no central cache repository, it uses peer-to-peer communication for service exchange.

Storage of service information. In SLP service information is kept either on DA or on UA and SA. In Jini, it is kept in the lookup service. Salutation keeps a service registry in every SM, whereas UPnP in every control point. Bluetooth SDP keeps it in every server.

Operation without directory. Except Jini, which needs its lookup table in order to operate, the other architectures can manage without it.

Leasing. Leasing causes some traffic overhead, because reregistration is needed periodically. Nevertheless, the benefit is that service entries are kept up to date. It is a concept that the majority of the architectures have adopted: SLP, Jini, UPnP, and HAVi. It is not used in Salutation, or in the Bluetooth SDP.

Security. UPnP does not address security. As for Jini, it offers Java-based security. SLP includes optional authentication. Salutation offers user authentication, whereas HAVi offers access levels and signatures. As for the Bluetooth SDP, it has basic authentication at a low layer. Higher layer security mechanisms are needed, for example, application layer security mechanisms.

Properties	SLP	Jini	Salutation	UPnP	HAVi	Bluetooth SDP
Developer	IETF	Sun Microsystems	Salutation Consortium	Microsoft	HAVi Organization	Bluetooth SIG
License	open source	open license, but fee for commercial use	open source	open only for members	open source	membership
Version	2	1.2	2.1	0.91	1.1	-
Network transport	TCP/IP	independent	independent	TCP/IP	IEEE 1394	independent
Programming language	independent	Java	independent	independent	independent	independent
Central repository	yes (optional)	centralized	optional, P2P or centralized	no, P2P	no	no, client-server
Storage of service info	in DA, or in UA and SA	in lookup service	serv. registry in every SM	in every control point	-	in every server
Operation without directory	yes	lookup table required	yes	yes	yes	yes
Leasing	yes	yes	no	yes	yes	no
Security	optional auth. of DA and SA	Java-based	user authentication	-	access levels, signatures	basic authentication, PIN codes

Table 1: Comparison among the protocols

Lack of inexact matching. The current SDPs allow exact matching when finding a service, but they are not good at finding services based on some other criteria. For example, a protocol can find a color printer, by matching the color attribute of the printer, but fails to find the printer that is the closest to the user. SLP is the exception that proves the rule, it offers more inexact matching than other SDP initiatives.

4 Conclusion

This paper reviewed the key architectures for service discovery. Several initiatives are tackling this issue at the same time, trying to dominate the market. The reason these initiatives have come out is to solve the problem that, currently, devices in a network need to be configured in order to be accessed. The home environment would benefit a lot from this, because the devices would be autoconfigured into the network. Moreover, integrated services can result from the seamless exchange of services between devices, which is valuable in many home environment scenarios. The developers see a business opportunity here, this is why there are several initiatives. Some of them are not targeted towards home environments alone, still, they try to encompass this area too.

It should be noted that, from the architectures discussed in this paper, HAVi and Bluetooth cannot become dominant, because they are very specific. Namely, HAVi addresses audio video traffic, whereas Bluetooth handles shortrange wireless connections.

Jini has overcome the fact that it is heavy for small devices by introducing the Java Micro Edition Platform, which is specially designed for small devices. Another problem with Jini is that the interfaces to the devices, according to the

architecture of Jini require to be implemented. If in UPnP there are committees that tackle this issue, it is unclear who should do it in Jini. UPnP stands good chances for becoming dominant, since it is based on popular protocols, and is supported by important players in the field. As for Salutation, it is independent of network transport, it has a lighter version for smaller devices, also it is open-source. It had very good chances of becoming dominant, but it has recently been dissolved. Therefore, its chances have been reduced. SLP offers no need for configuration, is open-source, and is already implemented. Therefore, it is also a candidate for being a winner.

It remains to be seen whether any of them will prevail in the future, especially in the home environment. Even if this happens, it will have to comply with existing devices that have other protocols. As of today, there is no clear winner, maybe there never will, and maybe this is a good thing. Because if a non-open source initiative becomes the prevalent one, then newcoming vendors would have to join it in order to be compliant with the devices most users have.

But joining these initiatives is not free; as a result, devices would be more expensive. New or existing vendors could lose their market attractiveness because of non-compliance. In this way, such an initiative might even end up controlling whether a new vendor enters/stays or not in the device market. On the contrary, it would be nice to see convergence towards an open source initiative. In this viewpoint, the dissolution of Salutation is a big step backward.

Convergence, however, seems not possible, since smooth transition from one architecture to another is generally impossible, in practical terms. The developers of the architectures seem to have understood this, therefore they have already started developing bridges from one architecture to another. But do these bridges need configuration? Most likely,

they do, because the architectures were not designed with compliance with other architectures in mind. In this way, we end up at the very point that we started out from: these architectures were meant to minimize configuration, but they replace it with other configuration. As a result, we can say that they solve the problem separately, but not jointly.

This field anticipates interesting developments. New architectures for service discovery might come out. The ongoing competition between the different solutions will maybe draw a winner after all. Additionally, the settling of user expectations will probably drive the industry towards more specific directions.

References

- [1] Roger Lea, Simon Gibbs, Alec Dara-Abrams, Edward Eytchison. Networking Home Entertainment devices with HAVi. Computer, September 2000.
- [2] Kevin Bowers, Kevin Mills and Scott Rose. Self-Adaptive Leasing for Jini. Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.
- [3] Adrian Friday, Nigel Davies, Nat Wallbank, Elaine Catterall and Stephen Pink. Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments. Wireless Networks 10, 631-641, 2004.
- [4] Christian Bettstetter and Christoph Renner. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. Proc. EUNICE Open European Summer School. September 2000.
- [5] Choonhwa Lee and Sumi Helal. Protocols for Service Discovery in Dynamic and Mobile Networks. International Journal of Computer Research, 2002.
- [6] Dipanjan Chakraborty and Harry Chen. Service Discovery in the Future for Mobile Commerce. Crossroads, September 2000.
- [7] Sumi Helal. Standards for Service Discovery and Delivery. Pervasive Computing, IEEE, 2002.
- [8] Sun Microsystems. Jini Architecture Specification, version 1.2, December 2001. <http://www.sun.com/software/jini/specs/jini1.2.html/jini-title.html>, 09 April 2007.
- [9] UPnP Device Architecture. http://www.upnp.org/download/UPnPDA10_20000613.htm, 09 April 2007.
- [10] IEEE Standard 1394-1995, Standard for a High Performance Serial Bus. IEEE Computer Society. 30 Aug 1996. IEEE Computer Society. <http://ieeexplore.ieee.org/servlet/opac?punumber=3871>, 09 April 2007.
- [11] Jim Waldo. The Jini architecture for network-centric computing. Communications of the ACM, July 1999.
- [12] Vasughi Sundramoorthy. At Home In Service Discovery. CTIT PhD. Thesis Series, 2006. http://doc.utwente.nl/57341/1/thesis_Sundramoorthy.pdf, 9 April 2007.
- [13] W3C Recommendation. 24 June 2003. <http://www.w3.org/TR/soap12-part1/>, 9 April 2007.
- [14] Universal Plug and Play in Windows XP. <http://www.microsoft.com/technet/prodtechnol/winxppro/evaluate/upnpxp.msp,9> April 2007.
- [15] Announcement of the Dissolution of Salutation Consortium, Inc. <http://web.archive.org/web/20050627074915/http://www.salutation.org/>, 9 April 2007.