Professor Dr. Stefan Fischer
Institut für Betriebssysteme und Rechnerverbund
Gruppe Verteilte Systeme
Betreuer: Muhammad Khan
Technische Universität Carolo Wilhemina zu Braunschweig
Mühlenpfordtstraße
38100 Braunschweig

Service Discovery in Home Environments

Seminararbeit
(Wintersemester 2002/2003)

Björn H. Gerth
<b.gerth@tu-bs.de>

Contents

List of Figures

List of Tables

1 Introduction

In the modern world, home environments include more and more electronic appliances in kitchen, bath and living rooms. Seldom are all these devices able to communicate with each other, although we are familiar with partial interaction, e.g. the CD player connected to an amplifier and its remote control. But since the devices nowadays contain a microprocessor (not only all sorts of consumer electronics) it just seems reasonable to connect them and have them communicate, interact and share services and functions. This requires some sort of service discovery within the network of appliances, such that devices can find the services offered by other devices. So middleware, the software connecting applications, providing the service discovery and sometimes the exchange of data, increasingly gained interest in recent years.

Taking a look into the computer world, the problem to realize a network of all sorts of devices is the current system setup. Disk-centric and client/server computing force networks with central connection points, a high amount of administration and thus low flexibility. Peripherals like printers are attached to some of the clients and have to be driven by special applications with certain device drivers. Addition of other components may require a complete shutdown and reboot of the network. Finally, relatively limited bandwidth may prevent the devices from meeting the user's quality expectations.

For a loose coupling of devices a different kind of architecture is needed. As mentioned in [8], we have to get away from configured environments to foreign networks with unknown infrastructures. Network-centric computing should allow devices to be integrated into the network by simply plugging them in; furthermore, the new devices should be able to use other entities present in the network and/or offer their own services. Thus, the network consists of a collection of services and clients instead of applications and peripherals, where the flexibility lies not only in the easy installation and removal of components, but also in the way these components interact.

Coming back to home environments, such networks would not only include printers, storage devices and specialized hardware, but cell phones, personal digital assistants (PDA), television, stereo components, telecommunication devices and even modern thermostats [13]. A quite fitting example is given by [3]: An alarm clock may be connected to the coffee machine and the home heating system; hence, the inhabitant will have hot coffee and a warm house when he wakes up. This is just one sample of the envisioned house of the future, with a smart environment: The electronics surrounding us simplify life by offering ways to spare us daily trips to supermarket, post office and baker. Digital technologies significantly enhance the user's entertainment experience [10]. All devices requiring user interaction can be controlled from one component, be it PDA, cell phone or simply the personal computer. Furthermore, home appliances might be remotely accessed through the internet, e.g. to start longterm processes before actually arriving at home.

The next sections will take a closer look at different service discovery architecture and how they are suitable for how environments.

2 Jini

The Jini architecture presents a kind of layer among devices to enable communication among them. Goals of the Jini design, detailed in the next paragraphs, were to allow a federation of easily pluggable and removable components, a low level of administration and avoidance of single points of failure.

The federation concept of Jini mainly means that a certain set of devices is registered with one or a few control points - the Lookup Services (LUS), explained below -, i.e. the network is made up of several federated control points. Although current distributed systems mainly use centralized control, Jini's federations show some advantages towards the expected excessive expansion of networks. It may be difficult to change centralized systems over time - especially when the systems grow to a very large scale. Federations may suit this problem better, for the reason that the control is spread over several points, the LUS, and a minimal set of rules that Jini imposes on the participants of the federation. These demands will be detailed below.

The integration and removal of new members is realized in an efficient way: On the one hand, it suffices to simply plug the new unit into the network and turn it on while on the other hand it is not necessary to update old members or even shut the whole network down for an upgrade.

Administration is still needed for the network. Although the Jini federation takes care of the administration, a device plugged into the network still has to be equipped with an Internet Protocol (IP) address (a possible automatic solution for this would be AutoIP of UPnP, see below). Furthermore for the unicast discovery protocol, see below, some help by an administrator is required.

The avoidance of single points of failure improves the reliability of the architecture. Since Jini also aims at including consumer electronics, which are not known for crashing the whole system due to a failure of one component, the breakdown of one device in the network affects other members as little as possible.

The communication between services and clients is realized with Service Objects, or

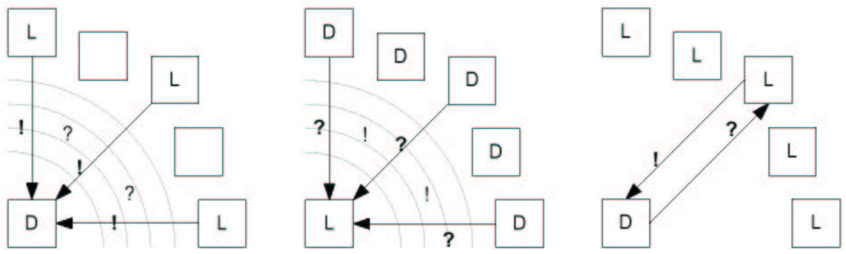

Figure 1: Jini: multicast discovery (left), multicast announce (middle) and unicast discovery protocols [3]

proxies. A proxy is a code object which represents the interface to its service; a client wishing to use the service simply downloads the proxy code and accesses the interface functions.

The communication between a proxy and its own service is not restricted by Jini, i.e. the choice and implementation of the underlying protocol depends on the developer. This process allows an insulation of the programmer from the Advanced Programming Interface (API) [14], and the communication protocol between proxy and service can be changed without any need for the client to be updated.

There are three kinds of proxies: Stubs, smart proxies and full service proxies. A stub simply delegates the request from a client to its service device. A full service proxy manages all the function itself and may not even have an underlying device. A smart proxy is somewhat in the middle, solving easy tasks by its own functions and transferring remaining function calls to the more complex device.

The Lookup Service is the part of the Jini model which allows spontaneous networking and which answers the question how a service can be discovered by a client. In a Jini federation there exists at least one lookup service where other services can register themselves and clients can request connection to the services.

There are three ways to set up communication between LUS and services/clients, shown in figure 1:

A service wishing to join a network sends out a message of the multicast discovery protocol, based on the Transmission Control Protocol (TCP), to a certain port. A LUS which receives such a message sends the requester a service object of its own, which can be used to provide the IP address of the LUS and identify the LUS later. The requester can now open a TCP connection to the LUS. In case the requester is a service it uploads its own Service Object and its service attributes, some other descriptive attributes, to

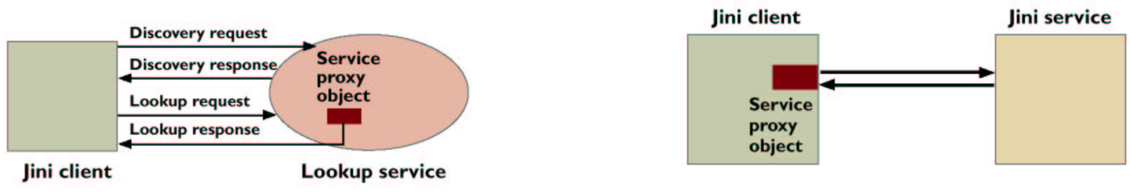

Figure 2: Jini: Client requesting service from LUS (left), client-service communication [13]

any or all of the answering LUS; in case it is a client it may query the replying LUS whether they offer a matching interface of the desired service.

A LUS announces its presence in the network through the multicast announce protocol by sending out packets. Any service or client which receives this message and does not already know that specific LUS (identifiable by the Service Object of the LUS) can now announce itself.

The last protocol is the unicast discovery, where a client or service may connect to a certain LUS. This LUS is usually farther away in the network, and its IP address has to be provided by a network administrator. The discovering entity may then directly connect to the LUS.

A client queries the LUS whether it offers a matching proxy, i.e. a matching Java type and maybe some desired properties in the service attributes. The client can then download the proxy from the LUS and communicate with the service directly, the LUS is not involved anymore (figure 2). Even if it crashes, the communication between service and client can still take place.

The Lookup Services provide a way to spontaneously enter a Jini network, but how does the architecture allow or discover the removal of a service? This and more is achieved by the third part of the Jini model, the concept of Leasing.

As an example, Leasing is used to maintain a connection between LUS and services/clients. The LUS grants a limited service registration lease to the other entity. Then, shortly before expiration of the lease the other entity has to renew its registration, otherwise the LUS assumes the entity has been removed from the network.

Other examples of how Leasing can be used in various situations throughout the Jini architecture are resource Leasing or event subscription. A service may lease its resources to a client so that it can operate successfully; this does not only include memory, but also space on a hard disk or the ability to display a user interface (the service does not have to use leasing for its resources - the developer is free to use other mechanisms). On the other hand, this even allows the use of resources outside of the client or the service - all resources available in the network may be leased. If the client wants to continue to use the service, it has to renew its resource lease in due time. Event subscription means that a client may register a subscription at a LUS, such that the LUS informs the client when a certain service registers or removes itself.

In addition, Leasing has several more advantages:

Jini may run deactivated services. Those services may not use any processing power or memory, but as long as they renew their lease with a LUS, they can still be used and may even receive a wake-up call from a LUS.

If the network goes down temporarily, for a shorter period than the lease time, it will not harm the federation too much. The resources allocated before will still be valid afterwards, until the lease time expires.

An entity may even cancel leases, in case they are in use but should not be. This may not happen too often, but in some cases it may be necessary.

Jini is based on Java technology, so it exploits several of Java's advantages. It turns a network of heterogeneous devices into a homogeneous collection of Java Virtual Machines (JVM), which supports the Write-Once-Run-Anywhere capability: operating system (OS) independence and dynamic loading of object code, e.g. for integration of new functionality or automatic bug fix updates. This also allows a raise of the abstraction level, so that programmers do not have to worry about implementation details of the network or communication protocols. Finally parts of the Java code can be executed at other locations through the Remote Method Invocation (RMI), which is used for the proxies up- and downloaded between services and clients. The most important disadvantage on the other hand is that Java still costs a lot of processing power.

Jini is certainly useful for the home environment, as long as interfaces for the desired devices are being developed. This is one of the drawbacks: Much is still left unspecified, interfaces for certain devices still have to be implemented, while they are already available for consumer electronics in other service discovery solutions. This prevents interworking between devices of different vendors.

3 Universal Plug and Play

Originally thought as an extension of Plug and Play (PnP), where devices of a personal computer did not require user configuration, the Universal Plug and Play (UPnP) forum, led by Microsoft, finally developed an OS and language independent service discovery

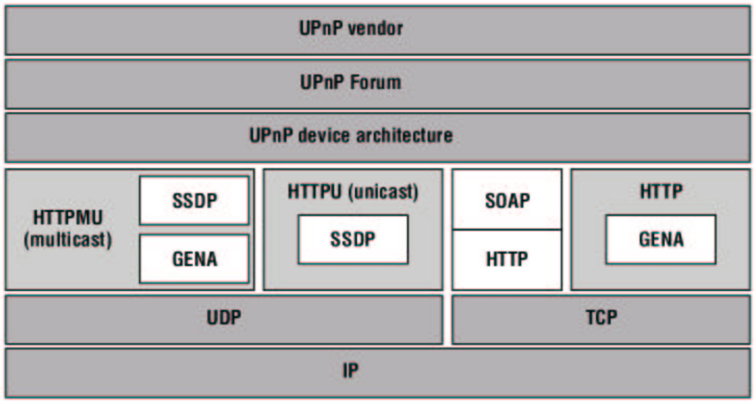

Figure 3: UPnP protocol stack[8]

that does not have much more in common with PnP than the concept.

The UPnP protocol works on a much lower level than Jini. Instead of working with a virtual machine to adapt to heterogeneous devices, protocols that are already implemented on nearly every machine were chosen, namely TCP/IP and other open standards like HTTP and XML (see figure 3. On that basis other protocols were developed to enable advertisement, discovery and control of the components.

The Simple Service Discovery Protocol (SSDP) is used to announce the presence of a component in the network and discover other entities. It works on HTTP multicast and unicast messages via UDP (HTTPMU and HTTPU). The joining device sends out a multicast message to advertise its presence. This advertisement can either be directly seen by other clients or recorded by control points. Control points correspond to LUS in Jini, with the exception that the network is not required to have a control point.

When a control point is added to the network, it sends out a multicast search message, which is answered by unicast replies from the connected devices.

A device may contain a set of services, i.e. corresponding to each functional component of the device. A very powerful feature of UPnP is the description of the services, which is stored in an XML file. This contrasts Jini, where the description is shortly given in the service attributes. The advertisement message of a device contains an URL (Uniform Resource Locator) which points to the XML file, which can be accessed and evaluated by the other devices for usefulness or by a control point.

The description contains a list of actions which can be used to access the service, and a list of variables which express the run-time state of the service. When these variables change, the service can cast an update message, written in XML and formatted using the General Event Notification Architecture, which contains the name and new value of the variables. Any control point can subscribe to receive this information. Furthermore, clients may be able to modify these states, forcing the service to update its internal states.

A description may even include a Presentation URL. This URL offers a higher level user interface, which can be loaded into a browser to display the status of a device or even allow the user to modify it.

To control the services, the Simple Object Access Protocol (SOAP) is used to invoke the actions. A SOAP message is sent to the SOAP control object of the service of the device, which returns action-specific values. The UPnP Forum has already developed several device and service definitions and control possibilites for common devices. These templates can be used to ensure interoperability between components of different manufacturers.

When joining the network, a device can receive its IP address in two ways. Either there exists a Dynamic Host Configuration Protocol (DHCP) server in the network, which assigns an address, or the device uses the AutoIP mechanism. In the latter way the device randomly chooses an IP address from a reserved range and then sends out a request with the Address Resolution Protocol to detect whether the address is already owned by another component.

UPnP is a service discovery solution for small to medium size IP networks [4]. Thus, its use for the home environment is ideal, especially with the AutoIP mechanism which does not require any network administration. Microsoft is already shipping its Windows XP operating system for personal computer with UPnP, furthermore the UPnP Forum has developed a specification aiming at consumer electronics, called UPnP AV [10].

UPnP AV is a set of device and service templates for devices handling entertainment content, such that the content can be distributed in the network. It consists of three parts, the Media Server, the Media Renderer and the AV Control Point.

A Media Server is a device that has access to entertainment content and can send it to another device for rendering. Media Servers include familiar devices such as VCRs or TV tuners.

A Media Renderer on the other hand is a device that can receive content and render it using some local hardware. This may be the familiar TV or a stereo system, but innovative renderers can use any type of output hardware that can be controlled by the incoming content, e.g. a water fountain dancing to a song.

The AV Control Point finally manages command and control operations to coordinate the Media Servers and Renderers. It is not, though, involved in the actual transfer of the content, thus the Server and Renderer can choose any transport protocol that they both support. This allows various transfer protocols and content formats.

This specification improves UPnP usability for consumer electronics in home environments and hence UPnP's position among the service discovery solutions.

4 Salutation

"Salutation is shipping while others are still thinking!" claims the Salutation Consortium in [9] while comparing its service discovery solution Salutation to Jini and UPnP. Products equipped with Salutation architecture have already been on the market since 1996. Since Salutation is nonproprietary and thus the specifications open and available for third-party development and since Salutation chooses a middle way between autonomy (Jini) and standardization (UPnP), it is easy for vendors to adapt to the specifications [11].

The main part of Salutation consists of the Salutation Managers and Transport Managers. To join a network, a service registers itself with the local or nearest Salutation Manager, which keeps a registry about all services and clients registered with it. A client requesting a service queries the local or nearest Salutation Manager, which in turn may redirect the search to all other Salutation managers in the network.

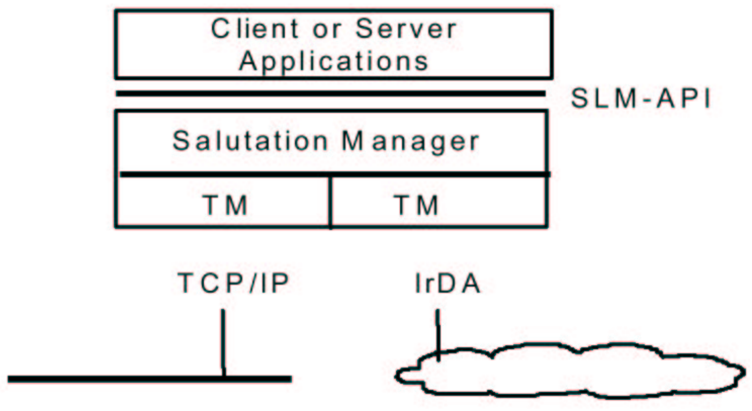

Figure 4: Salutation: Composition device - managers - infrastructure [9]

The Transport Managers offer reliable communication channels, regardless of the underlying network transports [8]. A Salutation Manager is attached to one Transport Manager for each infrastructure it is connected to, and this is what the Consortium points out as one of the strengths of Salutation: The support of multiple infrastructures (see figure 4). The interface between Transport Managers and Salutation Managers is called SLM-TI and provides protocol independence. Thus, one Salutation Manager can work with a TCP/IP as well as with an infrared-based network.

The communication between Salutation Managers and devices takes place through the SLM-API, a transport independent interface for service discovery, registration and access. It includes callbacks to the device for certain events, e.g. when data is arriving or another services makes itself available or unavailable to the Salutation Manager.

When a client requests a service, the local Salutation Manager specifies certain types and sets of attributes, which have to be matched by the services registered with other Salutation Managers. This capability exchange again allows the connection of devices of physically different networks.

There are three ways for components to communicate with each other: In the native mode, the communication takes place directly. In the emulated mode, the Salutation Managers carry messages between the devices, hence providing transport protocol independence. Finally in the Salutation mode the Salutation Managers do not only carry messages but also define their format, using well-defined standards for the interoperation between Functional Units.

A Functional Unit is a "minimal meaningful function to constitute a client or service" [8], i.e. the type or a feature of the device. Usually a service consists of a collection of such units. Each unit is composed of a descriptive attribute list, giving more details, which complies to the Abstract Syntax Notation One (ISO 8824). These units have to be matched during a search query.

Since Salutation is a transport-independent framework, the matter of self-configuration has not been solved. In IP networks, approaches similar to UPnP might be usable.

For security reasons Salutation included a user authentication by user identification and password scheme.

The Consortium also offers Salutation-Lite, a reduced version of the Salutation Architecture especially designed for small devices, such as PDAs. It is particularly suitable for devices with limited storage space, low communication bandwidth and little power consumption. The Lite implementation therefore abandons some function, e.g. the availability check callbacks. In case the dismissed functions are still needed, they can be later added as a user option [12].

Though Salutation may be a candidate for home environments, vendor developments have gone in direction of the small office. Most of the Salutation products available on the market are office automation products [8].

5 Home Audio Video interoperability

The Home Audio Video interoperability (HAVi) standard focuses on home entertainment products, i.e. mainly audio/video (AV) equipment. Therefore, it is probably the best candidate for service discovery in home environments. The progression of digital AV sources, the extension of network bandwidth and the more and more powerful processors in AV electronics are the basis for this middleware, which depends neither on any operating system nor on any brands.

The networking software specifies the protocols to be used by the components. These are communication mechanisms, which provide multi-directional AV streams, registries and the share of resources. Since the addressed product range is so narrow, it is easier to develop and implement special services and to meet the particular demands of (digital) audio and video. Concerning the development, HAVi offers advanced programming interfaces to facilitate the companies' software production. The components of the HAVi architecture, depicted in figure 5, are as follows [7]:

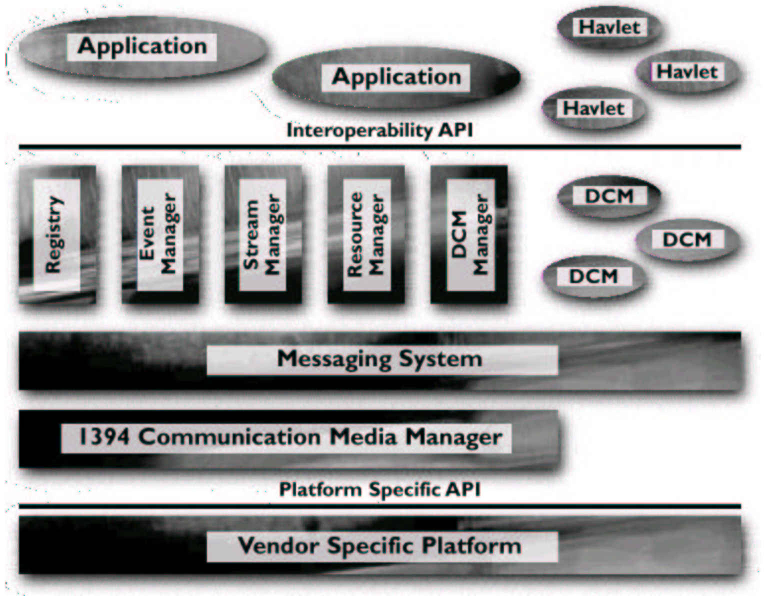

Figure 5: HAVi architecture [6]

HAVi discriminates between four types of devices: Full AV devices (FAV), Intermediate AV devices (IAV), Base AV devices (BAV) and Legacy AV devices (LAV). FAV completely support all elements of the HAVi architecture (see below), though some are optional, and additionally provide a Java runtime environment. IAV offer all these elements except the runtime environment, and BAV have only the base equipment. An LAV is a device which was not designed for HAVi, e.g. an older device, but it can still be integrated into a HAVi network.

Each device is represented by a Device Control Module (DCM) as a software element in the network. The DCM contains a set of Functional Component Modules (FCM) similar to the functional unit of Salutation. Each FCM represents a controllable function of the device, numerous FCMs and their APIs have already been defined in HAVi. From the FCM on, native language commands are sent to the appropriate target device. The DCM is the interface through which the function of the device can be accessed. An LAV does not possess such a DCM, since it was not developed for HAVi; it must be provided elsewhere in the network. IAV bring some embedded DCM for LAV (as well as BAV), while FAV are capable of uploading DCMs (possibly from the internet) and execute them through the runtime environment.

Each device except the LAV has self describing device data (SDD data), which contain descriptive information about the device and its capabilities. In case of the BAV, it may also include a DCM code unit, uploadable by the FAV.

The FAV and IAV offer some services which handle the communication in the HAVi architecture. Thus, it is not possible to make up a network containing only BAV and LAV. The services are the following:

The Messaging system has two functions: Firstly, it creates unique Software Element Identifiers (SEID) for an object, before it is registered. The SEIDs are later used to identify the elements in the network. Secondly, it performs the exchange of information between the devices.

The directory service within HAVi is called the Registry. Any software element which wants to make its service available to the network registers there, the Registry keeps track of the element's SEID and its attributes. For clients the Registry offers a query interface so they can find a service matching a set of criterias.

The 1394 Communication Media Manager provides a transport mechanism to send requests to and receive indications from remote devices. It also abstracts the network activities and presents information to the HAVi system; finally, it informs the Event Manager (see below) about network changes. The interconnection medium handled by the Communication Manager is the IEEE 1394 standard, which is also called i.Link (by Sony) or FireWire (by Apple). Not requiring any PC, this serial bus allows speeds of up to 400 Mbit/s, and it has a mode of transmission which guarantees bandwidth which makes it ideal for AV streams. Furthermore a digital copy protection standard known as "5C" was developed especially for this medium [2].

An Event is the change of state of a software element or of the network, e.g. when a device is added to the network. The Event Manager is a directory where software elements can register to receive notifications about such events. In case of global events, indicated by the event poster, the Manager notifies all other Event Managers in the network, which in turn address their local software elements.

If an FAV or IAV wants to host the DCM of a BAV or LAV, a DCM Manager is needed. It installs or uninstalls the DCM code units, usually on network resets (when the network topology changes, i.e. when devices get activated or deactivated or plugged or unplugged). Every FAV has a DCM Manager, but it is optional for an IAV, which indicates through its SDD data whether it is able to host other DCMs. The user usually does not have to interact in this setup, the configuration is called "plug and enjoy" [5] by the HAVi organization.

The Stream Manager configures both internal connections (within a device) and external connections (between devices). Streams take place between the FCMs of devices, the Manager handles requests for such streams and the allocation of system resources coming with it (in this case bandwidth and channel numbers) and releases the resources afterwards. It also provides global connection information and supports connection restoration after network resets.

A different kind of resources is dealt with by the Resource Manager: It handles device resources, i.e. the use and release of FCMs within a device. Certain strategies are implemented in HAVi, among them all-or-nothing allocation, share of read-access resources, and reservation of certain resources including an event when the reserved items become available.

The HAVi system services cannot only allow physical devices into the network, it is also possible to run applications, supported by the runtime environment of FAV. This includes a special feature of HAVi: The support of user interfaces (UI) on remote display devices. The Data Driven Interaction (DDI) protocol is used to connect the component with the UI and the display device. If the display is Java-enabled, it may extract and run a Havlet from the DCM of the other component (or even an application), where a Havlet is a Java application. So the display may show the graphical user interface of the Havlet, which allows the user to control the remote component, e.g. by accessing vendor-specific features.

There are two ways in which HAVi provides some security: The first one are access levels. It sorts APIs into sets of trusted and untrusted APIs, devices can then decide whether they want to run untrusted APIs or only trusted ones. The second way is based on the Rivest-Shamir-Adleman coding schema (RSA). A special organization, the HAVi Certification Authority creates private keys and gives out public keys. Furthermore, it assigns private keys to certain vendors. FAV may then run only certified Java code in their runtime environment, this prevents download of harmful code (e.g. from the internet).

The HAVi Organization was founded by eight well known consumer electronics companies, among them Grundig, Sony and Toshiba, and is supported by numerous others. Objectives are on the one hand to expand the system to include home control systems, security systems, communication systems and PC based applications; on the other hand, interconnecting bridges to other service discovery solutions like Jini are being developed.

Feature	**Jini**	**UPnP**	**Salutation**	**HAVi**
Developer	Sun Microsystems	Microsoft	Salutation Consortium	HAVi Organization
License	open license, but fee for commercial use	open (only for members)	open source	open source
Version	1.0	1.0	2.1	1.1
Network transport	independent	TCP/IP	independent	IEEE 1394
Programming language	Java	independent	independent	independent
OS and platform	independent	dependent	dependent	independent
Code mobility	yes (Java RMI)	no	no	yes
Srv attributes searchable	yes	no	yes	yes
Central cache repository	optional	no	optional	no
Operation w/o directory	Lookup Table required	-	yes	Registry required
Leasing concept	yes	yes	no	yes
Security	Java based	IP dependent	authentication	access levels, signatures

Table 1: Comparison of service discovery solutions

6 Conclusion

The middleware solutions above approach the problem of service discovery quite well, but it is still a long way to go to a complete and standardized device connection possibility. Table 1 shows a brief comparison between the four solutions, which reveals some advantages, but also disadvantages.

Apart from the individual handicaps like unsupported infrastructures or unspecified configuration etc, it is very difficult for all frameworks to address mobile devices [8], which already appear freqently in any home. This is a very severe failing, since future customers will expect ad-hoc networks including their own devices like PDAs or mobile phones. Another lack is the unexploited context information of the devices in the network, like their locations.

So, some waiting time is to be expected before the alarm clock truly adjusts heating and coffee machine.

References

[1] Christian Bettstetter, Christoph Renner: *A comparison of service discovery protocols and implementation of the service location protocol*
Sixth EUNICE Open European Summer School, Sept. 2000

[2] Dick Davies: *1394 Trade Association supports "5C" copy protection plan*
1394 Trade Association homepage, http://www.1394ta.org, March 2001

[3] Kees-Jan Dijkzeul: *Jini: Middleware solution of the future?*
Xootic Magazine, Nov. 2001

[4] Adrian Friday, Nigel Davies, Elaine Catterall: *Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments*
Second ACM international workshop on Data engineering for wireless and mobile access, May 2001

[5] HAVi Organization: *HAVi - Home Audio / Video Interoperability* website
http://www.havi.org

[6] HAVi Organization: *HAVi - The A/V digital network revolution*
White paper

[7] HAVi Organization: *The HAVi Specification*
May 2001

[8] Sumi Helal: *Standards for service discovery and delivery*
IEEE Pervasive Computing, July/Sept. 2002

[9] Bob Pascoe: *Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun*
Salutation Consortium, June 1999

[10] Yassir Rasheed, John Ritchie: *High-Quality Media Distribution in the Digital Home*
Intel Corporation, Nov. 2002

[11] John Rekesh: *UPnP, Jini and Salutation - A look at some popular coordination frameworks for future networked devices*
California Software Laboratories, June 1999

[12] Salutation Consortium: *Salutation-Lite Code Programmers Guide*
Salutation Consortium website, http://www.salutation.org/lite, July 2000

[13] Jim Waldo: *The Jini architecture for network-centric computing*
Communications of the ACM, July 1999

[14] Hinkmond Wong: *Developing Jini Applications Using J2ME Technology*
Addison Wesely, Sept. 2001