Controlling networked devices: a validation of two middleware architectures

T.M. Tran, P.J.F. Peters, J.J. Lukkien, P.H.F.M. Verhoeven
Eindhoven University of Technology
Department of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Email: {t.tran, p.j.f.peters, j.j.lukkien, p.h.f.m.verhoeven}@tue.nl

Abstract—**Cooperation of networked devices requires specification, advertisement and discovery of services that subsequently can be used either automatically or from a user interface. Middleware technologies support this in various ways. We implemented two technologies, viz., Universal Plug and Play and Corba and performed experiments with them. We compared them with respect to architecture, performance and generality.**

Keywords: *service, UPnP, CORBA*.

I. Introduction

Networked embedded systems can use the network connection to control other systems and to be controlled themselves. To that end it is required that their capabilities are available on the network. A network includes many devices such as PCs, routers, embedded devices, ...and each can expose one or several services. In general, a service can range from the unstructured use of a device to the structured control of some well-defined functions. For example, a display service might accept a digital video stream or it might offer a structured window-interface like X-windows does. Cooperation among services should be automatic, as far as possible. In general, automatic cooperation concerns six aspects:

- **Addressing and naming of services:** how to incorporate and idetify the service as an element of the network.
- **Service discovery:** how the service can be discovered by other elements in the network.
- **Service description:** how to specify the service so that other elements in the network can understand about it (functionality, capability).
- **Controlling a service:** how a service can be monitored and controlled by other elements of the networks.
- **Eventing mechanism:** how to handle event notifications from the service.
- **User interface:** how to provide an friendly interface networking environment to the user.

Most of these aspect are addressed in existing standard middleware architectures, e.g., CORBA, HAVi, Jini, and Universal Plug and Play (UPnP).

II. UPnP

UPnP is meant to be an architecture for peer-to-peer connectivity of devices of all form factors, ranging from PCs to rather simple appliances, within a limited physical environment (e.g., home, office or a public space). The envisaged cooperation is through the roles of a *control point* and *device*. A device is a container of *services*, where a service is the smallest unit of control in a UPnP network. A service exposes actions and state variables. A control point manipulates the device through the services. These three elements are the basic building blocks of a UPnP network. We use the term *terminal* to refer to a control point or a device. In fact, a terminal may play both roles. There are six issues involved in UPnP operation. They are addressed using Internet standards.

Addressing: When the control point or the device connects to the network it must obtain an IP address. If available, the Dynamic Host Configuration Protocol (DHCP) is used for that purpose. If not, Auto-IP is an alternative. In this way resource-scarce networks are included for UPnP operation.

Discovery and advertisement: The control point needs to find devices of interest; the device needs to advertise itself. The Simple Service Discovery Protocol (SSDP) [5] is used here. This allows configuration-free cooperation.

Description: The control point learns about the device and its capabilities (its services); the device needs to specify these. This description is an XML document for the device and an XML document per service that can be retrieved from the device.

Control: The control point invokes actions from the device. The Simple Object Access Protocol (SOAP)

[7] is used.

Eventing: The control point subscribes to events and then listens to the state changes of the device. The General Event Notification Architecture Base (GENA) [6] is used here.

Presentation: The control point controls the device and/or views the device status using a Web browser.

In a sense one can say that UPnP combines several protocols into a single architecture. Using this, a terminal can dynamically join a network, obtain an IP address, convey capabilities, and learn about the presence and capabilities of other terminals. Terminals can subsequently communicate with each other directly, assuming control point and device roles for services.

The example of a TV set helps to explain this. We refer to it as *TVDevice* and it admits the following operations: *Switch* TVDevice *on/off*, *Change channel*, *Change volume*, *Change color* and *Change contrast*. To define *TVDevice*, we divide its functionality into two services: the first one, named *ControlService*, fulfills the first three functions and the second one, *ScreenService*, fulfills the last two. Three XML files are required to specify *TVDevice*: one file specifies *TVDevice* and two others specify its two services. Now, a cooperation of *TVDevice* with some control point *TVControl* proceeds as follows.

- When *TVDevice* connects to the network it obtains an IP address and advertises itself through SSDP.
- When *TVControl* connects to the network it obtains an IP address and issues a discovery request through SSDP.
- Subsequently, *TVControl* retrieves the device description and gets a list of associated services, it retrieves service descriptions of interesting services, it invokes actions to control the service and it subscribes to the service's event sources. Each time the state of the service changes, an event to the control point will be generated.

Retrieving XML documents is done through HTTP; in addition, SSDP, SOAP and GENA use HTTP as carrier. As a result the architecture contains HTTP as a layer and several HTTP servers must be present, one of which is a web server.

Different categories of UPnP devices will be associated with different sets of services. Consequently, different working groups will standardize on the set of services that a particular device type will provide. All of this information is captured in the XML device description document that the device must host.

UPnP uses open standard protocols, like TCP/IP, HTTP and XML. However, other technologies could be used to network devices together for reasons of cost, technology requirements, or legacy support. Examples are Jini [3] and HAVi [4]. These can participate in the UPnP network through a UPnP bridge. This bridge acts as a proxy application between UPnP terminals (devices, control points) and components which belong to other networking technologies. Discussing these other technologies is beyond the scope of the current paper. For a complete description of UPnP we refer to the site of the UPnP forum [2].

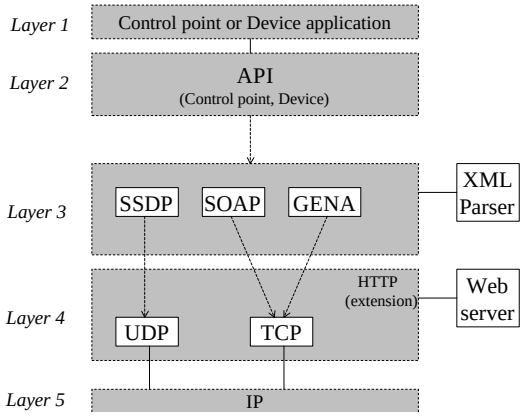

Fig. 1. *architectural model of the UPnP system*

We have constructed an API for both control point and device and implemented it in Java. The API is target to embedded systems. The architectural model is described in Figure 1. There are 5 layers in this model. The top layer is the application which is built on top of the API - the second layer. The API is a collection of several classes and the application is built from these classes. The UML [10] class diagrams for the control point and the device are given in Figure 2. The specific device application is an extension of class *Device*; the specific control point application extends class *Discovery*. Within the control point, classes *CDevice* and *CService* represent proxies of the actual device and service. Class *GUIControl* represents the framework for the user interface if needed. Layer 3 includes three components that implement the three protocols SSDP, SOAP and GENA. Components in layer 3 are started by the API. The dotted line in Figure 1 represents a call-back function; the direction of the arrow represents the direction of data transfer.

Layer 4 represents two known protocols: UDP and TCP. To implement the discovery, the SSDP component uses UDP. For controlling (SOAP) and eventing (GENA), TCP is employed. The format of data transfer between layer 3 and layer 4 conforms to the HTTPe

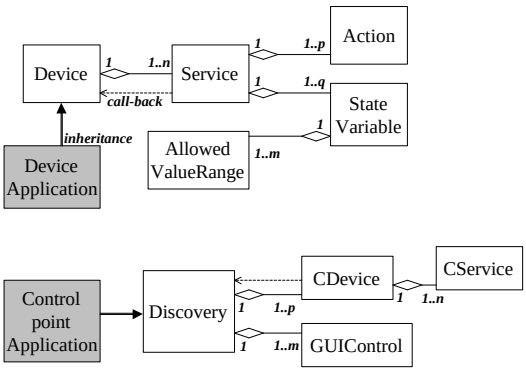

Fig. 2. *class diagrams for the API*

protocol which is an extended version of HTTP/1.1 with the purpose of fulfilling the requirement of SSDP, GENA and SOAP. In layer 4 two essential classes are *UDPListener* and *TCPListener*. UDPListener is an HTTPe server running on top of UDP to handle the service discovery process in SSDP. TCPListener is an HTTPe server running on top of TCP to deal with the requirements of SOAP and GENA.

Since IP is the last layer, every element in the system must be IP-enabled.

There are two components that do not belong to any layer. The component named *XML Parser* can be invoked by any component in layer 3 to parse XML content. The component named *Web server* is an HTTP/1.1 server running only in the device. This server handles the requests for the device and service descriptions.

The complete explanation of our design and implementation is found in [8]. The source code size of the device API is 80.1 Kbyte and of the control point API 106 Kbyte. When they are compiled into Java byte code, the device counts for 62.3 Kbyte and the control point counts for 85.7 Kbyte.

III. UPnP experiments

In order to validate and evaluate the UPnP API we have built two prototypes. The validation concerns the six aspects as specified in the introduction part. The first prototype is *TVControl* controlling *TVDevice* that we discussed before. This is a simulation of a user interface running in a computer to control the television using two services *ControlService* and *ScreenService*. A system consisting of a single device and a control point was easily constructed. The services were described in XML files in such a way that a user interface could be generated automatically. In order to discover a device the user has to enter the identification or the type of the device. This is a disadvantage as it prohibits a directory lookup.

The use of SSDP for the discovery requires a fixed interval of the advertisement. For the system with two elements (a control point and a device), SSDP works reliably. However, if we increase the number of TVControl(s) gradually from 10 to 100, the system becomes less sensitive to recognize newly added elements, due to that long interval. Perhaps there could be a dynamic way to assign this interval , depending on the network size. When there are 100 TVControl(s), from a functional perspective the system still works well but the time lapse between request and response becomes unacceptable, especially in the eventing part: the 101st *TVControl* receives the event after roughly 10 seconds. The reason for this problem lies mainly in the use of HTTP as the transport protocol. If we have 101 *TVControl*(s), then the 101st *TVControl* receives the event only when the previous 100 *TVControl*(s) have already received it, i.e., after setting up and destroying 100 TCP channels. Although eventing is, in fact, multicasting this is not implemented effectively. The result could be slightly better by communicating the events concurrently but that would impose a high demand on the processing power of the embedded device. Clearly, the protocol is not intended for that.

We see that the use of HTTP has both advantages and disadvantages. On the upside is the standardization which allows communication of a device with a Web browser. In addition, HTTP is connectionless which is good in the dynamic context of devices that come and go all the time. And finally, an HTTP communication itself is realized using a connection-oriented protocol which guarantees reliability. On the downside there is the significant overhead in setting up connections, the relative complexity of the protocol implementation and a significant timeout penalty when a device disappears *during* an HTTP transaction.

The second prototype is a coffee machine controller. We have built a Web-enabled coffee maker before, for the purpose of research on Web-enabled embedded systems[9]. The user can control a real coffee machine via a Web browser using an applet and a dedicated protocol. There is an application server that mediates between the Internet connection and the coffee machine, named *Coffee Server*. From a Web browser, the user can do the following: *Switch machine on/off*, *Select taste level*, *Inspect amount of coffee in the container* and *Inspect temperature*. In order to bring the system to work in a UPnP environment, we built a

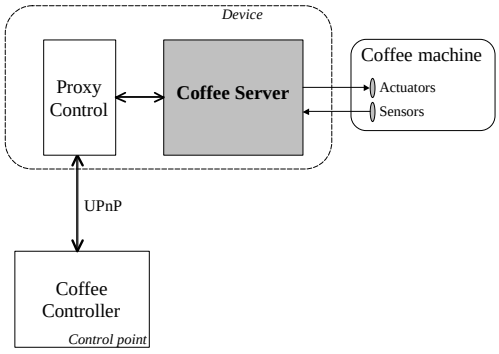

Fig. 3. *UPnP-enabled coffee machine*

control point, named *CoffeeController* and a proxy, named *ProxyControl* (see Figure 3). *CoffeeController* also can process the four commands. This prototype shows how to upgrade a legacy and non-UPnP system to become UPnP-enabled.

IV. UPnP conclusion

We have designed a UPnP API and implemented it. Two aspects that have driven the design process are that the API is built for embedded systems and that UPnP is applied for home networking environments. We also have built two prototypes for the purpose of evaluation. It is clear that UPnP can really be used for embedded systems in the home environment provided that IP is available. In applying UPnP for this kind of environment, several conclusions can be drawn:

Flexibility of design: Even though the UPnP forum has given the UPnP specification, vendors are free to construct their own environment, depending on the type of in-house devices they have. In particular, the API and its implementation are not defined. In our case, the API is designed for embedded systems with limited processing power.

It is not too difficult to upgrade a non-UPnP device to become UPnP enabled. However, for a system with several standards living together like UPnP, Jini and HAVi, it is not easy to solve the problem of interoperability.

Service discovery: SSDP allows control points and devices to discover each other using IP multicast. The use of a fixed interval for the advertisement seems not a good way in a large network. There could be a way to assign adaptive intervals, depending on the network size.

Eventing mechanism: UPnP employs TCP to transport events. If the system has a limited number of control points and when there are few events this is suitable, since TCP maintains the reliability. Within a system with hundreds of control points (most of them just playing the role of monitoring devices), the use of TCP can cause the system to break down due to too much traffic. In addition, in case a control point is removed from the system the TCP timeout mechanism causes long delays. It appears that a new UDP-based protocol should be designed that deals more effectively with the broadcasting and dynamic connections.

User interface: The way of searching for a device can be inconvenient for a user. To find a device, one has to type either its type, its name or its identification number. In practice, a user likes to find his device by using some *browse-and-click* way. For example, to find the television, one may choose the first floor of the house, then come to the living room and click on the television. One should not have to remember an identification of the television. In general, how to create a suitable naming mechanism in UPnP is still an open question. The above example could be included as just another UPnP service though.

Quality of service/content: The way to describe a device by using XML also raises another question: is an XML-based document rich enough to cover such cases as real-time requirements or very complex interfaces? Research in this direction can lead to a specification of the quality of service (QoS) and the quality of content (QoC) in the home networking environment through XML documents.

Extensibility: Moving from a single environment to wider area networks introduces several new challenges. One UPnP model may no longer suffice. Consider the case when the user wants to control the house remotely, from his office. The existence of a UPnP gateway (or bridge) is a reasonable answer. Then, how about Internet firewalls, and security concerns in general. These things have not been addressed in UPnP explicitly. The security concern in UPnP seems to be delegated to the problem of maintaining the security in each HTTP server in the UPnP architecture.

V. CORBA

CORBA[11] is an object oriented middleware architecture for distributed program development, as defined by the object management group (OMG). It originates from large business integration systems, but it is also applicable to smaller systems to provide network abstraction in an object oriented service. Many CORBA applications use the client-server model, although it is also possible to create peer-to-peer applications.

CORBA itself relies on reliable connections between

fixed end-points, which does not necessary hold for an in-home network environment where devices come, go and move around. To address these issues, the Telecom working group of OMG defined wireless CORBA [12], [15] as an extension on CORBA to handle hand-overs and access recovery. With the extension, it also provides service discovery based on movement and direct usage of low-level network protocols.

The six aspects of automatic cooperation also apply to devices using wireless CORBA.

Addressing: CORBA relies on the underlying link protocol to handle the addressing issues, whether that is based on DHCP or some protocol specific solution, as with BlueTooth or IrDA. With wireless CORBA, it is not necessary to have a TCP/IP connection, so it is possible to use BlueTooth or IrDA addresses directly. If the communication goes beyond address boundaries, CORBA will take care of the address translation.

Discovery and advertisement: Discovery in CORBA is usually done with the naming service[13], where available services are registered. Within wireless CORBA, a hand-over results in access through a different network node, which can provide a different naming service, thereby creating location-aware service discovery.

Description: Within the naming service, the registered services are stored under a certain name within a certain context, similar to a file within a directory structure. As a service is an object within CORBA, the object type provides information about the available operations on the service.

Control: When the client uses a service, it uses remote method invocations to perform operation on the object as provided by the service.

Eventing: The client can subscribe to the service to receive notifications. This can be based on the event service of CORBA or it can use a service specific implementation.

Presentation: The presentation of the service to the user is up to the client and is very service specific. CORBA does not provide standard ways to create a user interface from a service.

When the service provides a user interface, the last three aspects are closely related. The presentation provides access to the control operations, the control operations might update the presentation and eventing also causes updates to the presentation. As the implementation of these aspects is service specific, it is useful to provide a generic solution which can be used by different services which require a user interface.

Within HAVi[4], the data driven interface (DDI) protocol provides such a generic solution for constructing user interfaces. However, the protocol is not directly usable as it is not optimized for CORBA and the different network conditions, such as disconnections and hand-overs. Within Philips and the ITEA project VIVIAN[14], the DDI protocol is adjusted to make it more Internet friendly by using XML and to improve the network characteristics. The result is the remote user interface (RUI) protocol, which can be used on top of different middleware layers.

With the RUI protocol, the last three aspects are slightly different if the service is in fact a RUI service.

Control: When the client subscribes to the service, it receives an XML description of user interface. All actions on this user interface are forwarded to the service, which responds with information on how to update the user interface.

Eventing: When the status of the service changes, the service sends a notification to the client with information on how to update the user interface.

Presentation: A user interface description is provided by the service, which is rendered by the client application. The user interface can be updated due to user actions on this interface or due to service notifications.

With the RUI protocol on top of CORBA, it is possible to reuse the user interface engine on the client for different services without adding service specific code. However, a human user of the service is required to interpret the user interface that is presented.

The RUI protocol uses incremental updates to the existing user interface in order to reduce the network usage and to be more responsive. For example, if the interface is replaced completely as is done in Web browsers, it often results in glitches in the screen updates that can confuse the user. Furthermore, interface elements like sliders might require a short feedback loop to the user, which are not possible with full screen replacements.

VI. CORBA experiments

The combination of CORBA and the RUI protocol have been implemented and tested in order to evaluate its usability. Java was used as implementation language to be able to test the system under different configurations. The target platforms are a PC with Windows, a laptop with Linux and a PDA with EPOC (a Revo). As the memory, storage and processing resources on the PDA are limited, it is important to stick to the library standards and not use the library

specific extensions or features. In the final client application, it is possible to replace the CORBA library and XML parser library without significant problems.

The Java AWT library is used for the graphical user interface which is constructed by the client application based on the XML descriptions that it receives. This choice is based on the limitation of the PDA, which does not support more advanced user interface libraries.

The client application can be used unchanged on all three platforms under different Java virtual machines with different CORBA libraries and XML parser libraries. For the PDA, only one configuration was possible due to the resource requirements on the needed libraries. The communication between the two high-end configurations was based on Ethernet, while the communication with the PDA was based on PPP on top of a serial line or infrared link.

A server application is constructed to allow easy definition of services. The service definition is based on a state machine to indicate which user interface changes apply to which state changes. For service specific operations, a special Java handler class can be provided.

With the server application, several services have been defined to test the protocol and the client application. Similar to the UPnP experiments, the TVDevice and the coffee machine controller were constructed as RUI services. Although the actual connection to the coffee machine was not constructed, there are no technical problems in making this connection.

To verify the slider interface element, one service was constructed where a slider is used to control a volume setting, thus creating a feedback loop between the client's user interface and the controlled audio device. The performance of this feedback loop is sufficient for practical use, even when the client is a PDA that communicates across an infrared link.

In the setup that is used, wireless CORBA is not used, as the Java based CORBA libraries do not support this extension yet. As the extension requires changes to the internals of the library, it is not possible to add this extension as a separate collection of Java classes. Wireless CORBA mainly handles the roaming aspects of the communication and hides this from the application itself. The location based service discovery provided by wireless CORBA is not tested yet. Instead, the naming service is used, where the client connects to a fixed naming service and retrieves the available services.

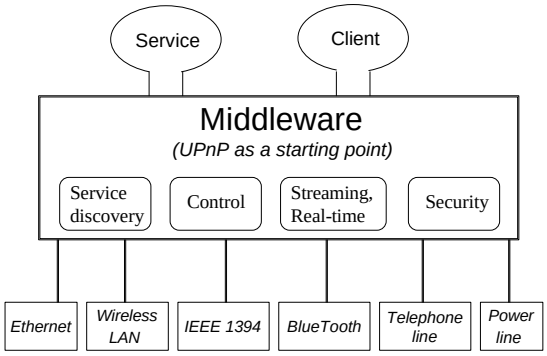

Fig. 4. *home networking environment architecture*

VII. CORBA conclusion

Within CORBA, the six aspects of automatic cooperation are available, but it still requires that the client has service specific information about how to access the service. The same holds for UPnP, as it is not possible to control another device without knowing what actions the available operations perform.

The addition of the wireless CORBA extension improves the addressing and service discovery aspects. The addition of a RUI protocol improves the presentation and control aspects.

The experiments showed that it is easy to replace one CORBA library for another without any problems. This is an important issue, as a library for embedded system does not provide the same development facilities as a library for a PC platform. This also applies to XML parsers, where the small parsers don't provide features like XML validation.

The RUI protocol is a convenient protocol for fast service construction without having to update clients with service specific software. Furthermore, it provides an abstraction for the user interface with respect to the diversity of the client device.

VIII. Future Research

We will create a home networking environment based on IP. The house will include a television, DVD, coffee machine, bell,.... We first use UPnP as a back-bone protocol for devices to communicate with each other. The architecture for this environment is depicted in Figure 4. When building, we will investigate in more detail many issues that come with UPnP, based on different type of networking technology: Ethernet, Wireless LAN, IEEE 1394, BlueTooth, telephone line and even power line.

Media streaming is an issue needed to address, in concern with the Quality of Service. UPnP forum now is working in defining a standard for media in

UPnP. We will build several multi-media applications and investigate the possibility to add real-time facility to the environment.

The home networking environment is also connected to the Internet. The user from outside can access the house for many purposes, also the user from inside can access the outside world. We also investigate this possibility. Security is a problem that needs to be addressed here.

Since UPnP does not provide a standard for presenting the service to the user, the RUI protocol can be used on top of UPnP, as it does not specifically depend on any CORBA features. For example, a UPnP device can provide a RUI based service as an alternative for accessing the other UPnP services directly. If the control device does not support a particular UPnP service, it can present the user interface provided by the RUI service to let the human user determine what UPnP services the device provides and how to use them.

In the future , we intend to develop a uniform architecture for a home networking environment. UPnP is only a middleware standard. It is not perfect but is a good starting point. Improving UPnP can lead to a good middleware standard for home networking environments.

References

[1] *Universal Plug and Play specification v1.0.* Online: `http://www.upnp.org/resources/documents.asp`.

[2] The Universal Plug and Play Forum. Web site: `http://www.upnp.org`.

[3] K. Arnold et al. *The Jini Specification.* Addison-Wesley Longman, Reading, Mass, 1999.

[4] *HAVi specification 1.0.* The HAVi Organization, January 2000.

[5] Yaron Y. Goland et al. *Simple Service Discovery Protocol/1.0.* Internet Draft, October 1999.

[6] J. Cohen et al. *General Event Notification Architecture Base: Client to Arbiter.* Internet Draft, September 2000.

[7] *Simple Object Access Protocol(SOAP) 1.1.* World Wide Web Consortium, May 2000.

[8] T.M. Tran, P.J.F. Peters, J.J. Lukkien, L.M.G. Feijs. *Design and implementation of an API for UPnP-based embedded systems.* Eindhoven Embedded Systems Institute, Eindhoven University of Technology, 2002.

[9] P.J.F. Peters J.J. Lukkien, M.F.A. Manders and L.M.G. Feijs. *An architecture for web-enabled devices.* In *Proceedings of the 2001 International Conference on Internet Computing, Las Vegas*, June 2001.

[10] J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley Longman, Reading, Mass, 1999.

[11] Object Management Group. *The Common Object Request Broker: Architecture and Specification.* Technical Report formal/01-02-01, version 2.4.2, OMG, February 2001.

[12] Borland, Highlander, Nokia, and Vertel. *Wireless access and terminal mobility in CORBA.* Technical Report dtc/2001-06-02, OMG, May 2001.

[13] Object Management Group. *Naming Service Specification.* Technical Report formal/00-06-19, OMG, April 2000.

[14] Vivian - opening mobile platforms for the development of component-based architectures. Online: `http://www-nrc.nokia.com/Vivian/`.

[15] P.H.F.M. Verhoeven, J. Huang and J.J. Lukkien. *Network middleware and Mobility* In *Proceedings of the second PROGRESS workshop, Veldhoven*, October 2001.