On-Demand Component Deployment in the UPnP Device Architecture

Didier Donsez
Laboratoire LSR, Equipe ADELE
Université Joseph Fourier
BP 53, F38041 Grenoble Cedex 9,
didier.donsez@ieee.org

***Abstract*— The standardization of networking home appliances fosters home automation joining the mass market. Controlling the appliances requires several either specialized or generic controls. This paper is interested in the dynamic trading and deployment of software components implementing UPnP control points. It also addresses the bridge between UPnP and the world of micro-appliances. An OSGi-based prototype validates our proposition.** *(Abstract)*

Keywords: Home networked appliances; UPnP DA; Control point; Trading; Deployment; Component Model.

I. Introduction

The home automation is finally becoming a reality for the mass market. Until now, the market was scattered into a large number of appliance manufacturers promoting incompatible and proprietary control protocols, making it difficult for the market to progress. One of the main reasons was that until now, it was very difficult for the integrator (architect, installer...) to provide a completely integrated solution covering all the types of appliances (HVAC, shutters, burglar and fire alarm, patient' healthcare monitors, etc) to their customer. The beginning of more widespread opened norms and standards as X10 and the generalization of domestic IP wire and wireless connections has brought about a new era of home automation [1] and build automation [2]. Indeed, some device discovery technologies [3] such as mDNS, UPnP, DPWS, IGRS and EchoNet, and home middlewares [4] allow the dynamic addition and withdrawal of device in the home network without the necessity of user interaction (ie. zero-configuration, zero-administration).

Universal Plug and Play (UPnP) Forum [5,6] is an open industrial consortium formed in 1999 for the definition of standards simplifying the network set up of communicating devices (i.e. appliances) in homes and companies (SOHO: Small Office Home Office). UPnP Forum published a first version of the required standards for networks and several standard definitions of devices and services associated with these devices. UPnP is a distributed platform with dynamic services for devices (TV sets, DVD players, light control, HVAC, security cameras, etc) and control points (PDA, TV sets, touch panels, etc) connected through an adhoc wired or wireless IP network (figure 1).

UPnP defines the required network standards that enable the live detection (and withdrawal) of devices, the use of provided services through the control points and keeps the user informed of changes in the current state associated with the services. In this platform, some devices actually serve as gateways between "*micro-worlds*" of "*micro-devices*" using other open or proprietary protocols than IP (X10, LonWorks, Konnex/EIB, BACNet IEEE 1394 …) and UPnP control points. The platform can integrate low-cost devices into the UPnP network, which cannot carry an IP stack.

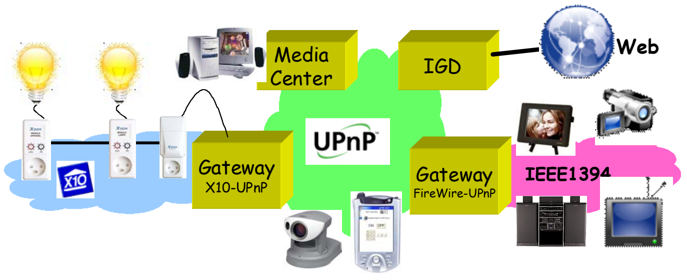

Figure 1. Example of a UPnP device network

The UPnP control points are either remote control specialized for a given manufacturer or a given model of devices, or overall devices with GUI (phones, PDAs, display panels or TVs) more or less refine. In this second case, the overall control points program must have the control interface components of all devices connected in the residence. It also must contain the interfaces of future devices, which maybe added. There are over a hundred different variations of these control interface components due to the wide range of goods produced by manufacturers. Moreover, this task is complicated by the coexistence of several GUI design frameworks. For example, a Java developer has the choice between several alternative frameworks, which can be used to build an GUI (ie AWT, Swing and SWT for the workstations, eSWT for the PDAs, MIDlet for the mobile phones and HAVi and JavaTV for the TVs, etc)

Similarly, the *micro-world* gateway software must also include the conversion components enabling the translation of UPnP actions into network commands used by the micro-devices. Gateways must therefore take into account the addition of any new devices through the update of its firmware, which requires frequently a gateway reboot.

For the moment, the dynamic updating of control point programs and micro-world gateways has been addressed by neither current products nor literature.

In this article, we suggest a brokerage and an "on-demand" deployment framework of software components for the control points which we call *controllet*, and components for gateways, which we call *bridglet*. This framework is based on the Service Oriented Architecture paradigm (SOA) [7,8]. SOA allow building applications in which any service implementation can be substitute with another one as long as the latter respects the contract of the service.

The rest of the article is organized in the following way: it starts with an introduction of the two types of components controllet and bridglet, which are associated the UPnP devices. Section 3 introduces the brokerage, deployment and composition principle of these components. Section 4 introduces a prototype of this framework build over the OSGi services dynamic platform [9,10]. Section 5 will show the framework in relation to related works. Finally, this article concludes with several perspectives.

II. UPnP Device Model

Any UPnP device embeds its own description and can provide it for requesting control points. The description follows a hierarchical model composed of (sub-) devices and services (figure 2).

The description (which is expressed in an XML grammar) includes a list of provided services and eventually a list of other embedded (sub-)devices. For example, a video recorder is both a media source (called Media Server) and a media recorder (called Media Renderer). The device description also contains a type identifier (standard or proprietary), the manufacturer, the model and serial number. A standard type refers to a standardized device profiles according to the UPnP Forum.

Each service had also a type. It includes states variables and actions. A variable can be, for example, the volume level of a TV set and an action can be to increase this level, another to lower it and a last to enter or exit the mute mode. Some variables can notify the change of value to control points which subscribe for those changes. This notification makes it possible to use context-aware control points [11,12].

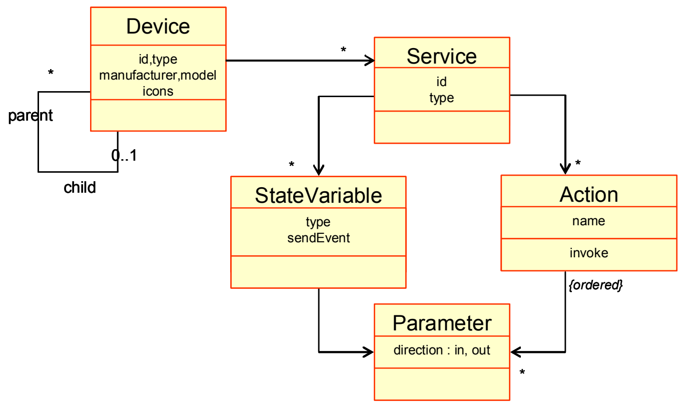

Figure 2. UPnP Device Model

The UPnP Forum standardizes and publishes descriptions of standardized profiles for devices and services (named Device Control Protocol (DCP)). These descriptions can incorporate some mandatory elements and some others optional. For example, a video surveillance camera could not provided zooming and focusing functions. Moreover, manufacturer can extend a device standard definition by adding proprietary extensions such as the night vision mode and rotating motion for a security camera.

A candidate to the next generation of UPnP is DPWS (Device Profile for Web Services) [13]. It relies on standard WS-* protocols and it is already implemented by MicroSoft and provided in Windows Vista.

III. Controllets and Bridglets

In this article, we propose an "on-demand" deployment framework of software components for the control points, the controllets, and components for gateways, the bridglets (figure 3). This framework can trade components according to the UPnP device hierarchical model.

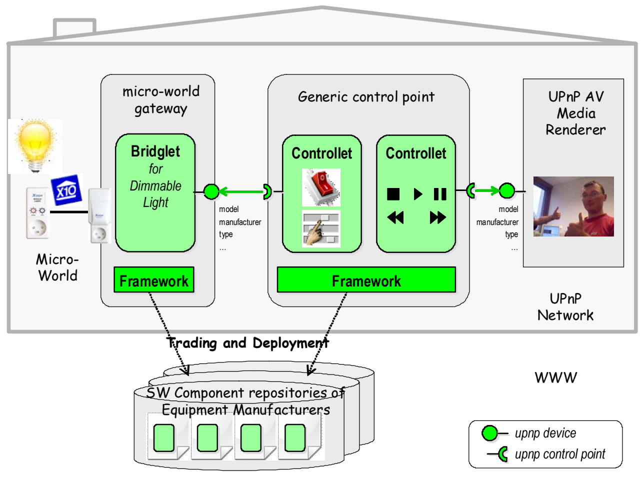

Figure 3. Deployment of Bridglets and Controllets

A controllet is a component for controlling a device or a service. It gives the user a graphical interface in order to show the device or service current state. It is linked to the controlled service variables in order to make the GUI reactive to the value changes. For example, the stop/start button in figure 4 only switches when the state change is notified. A controllet can log the actions carried out by the user as well as the last known values of the state variables. The controllet can then continue to post the supposed state of a device temporarily disconnected from the point of control (i.e. transitory loss of the WiFi network...).

Bridglets are components that carry out conversions of operation and notification between "micro-worlds" devices and the UPnP network. The brigdlet introduces all or a part of a "micro-world" device such as an UPnP standardized or proprietary device or service. In the example in figure 3, the bridglet controlling the X10 light bulb exposes the standardized UPnP device type called "Dimmable Light".

In this framework, the components, controllets and bridglets, have the following properties:

- Modular: a component can process either a complete device, either one of the embedded sub-device, or one of the services of the device root or one of the sub-device. For example, a service controllet (cf button ON/OFF in figures 4 and 5) can be used for all the start and stop operations of all electric appliances. Therefore, it is reusable across several device controllets (thumbnails 5 and 6 in figure 5).

- Composable: as in many hierarchical component models, a component can itself be made up of other components, which can be used for other sub-devices or services. The composition of controllets (as well as bridglet) generally follows the hierarchical structure of the UPnP devices. In figure 4, the TV controllet is made up of a power controllet, a channel selection controllet, and a volume adjustment controllet.

The sub-components can be statically setted at design time or can be dynamically traded and setted at runtime according to the SOA approach. In the case of the controllets, an order and a priority can be also specified for the sub-components displaying. The priority is used to display only the important controllets when the GUI context limits the controllets displaying or when the user desires a simplified interface. For example, the TV controllet hides currently the color and contrast adjustment controllet or the Dolby and 5.1 extensions sound controllet.

- Polymorphic: the framework handles overall components well-able to deal with a generic manner the largest number of devices as well as specialized components which only deal with a device of a particular model from a particular manufacturer. Specialization in general takes advantage of owned extensions of devices and services. For example, a specialized controllet for the selection of 99 channels of a TV set is represented by the 10 usual keys of an ordinary remote control, while a generic controllet will represent this service either by a text field to fill up or a 99-increments scrollbar.

- Substitutability and negotiable: In conformity with SOA paradigms, several components can replace each other to take the control of a device. However, the framework always looks for the most specialized component first. For this, each component is attached to a description listing the model and the manufacturer of the controlling device, the type of the controlled device and service as well as the supported execution environment (especially if the component uses C/C++ native libraries). The controllet description can be completed by the culture (user language, etc) and its environment description includes the required GUI framework.

IV. Component Trading and Deployment

The framework uses the description attached to the controllets and the bridglets in order to carry out the trading, the composition and the deployment.

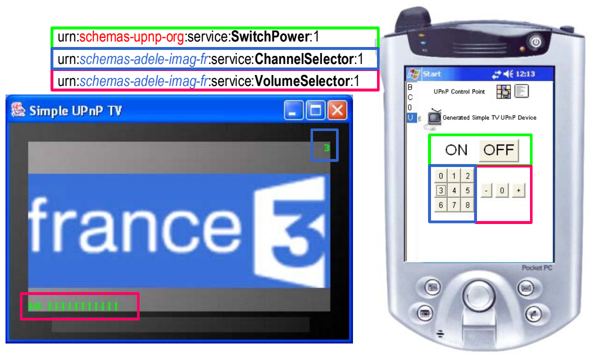
urn:schemas-upnp-org:service:SwitchPower:1
urn:schemas-adele-imag-fr:service:ChannelSelector:1
urn:schemas-adele-imag-fr:service:VolumeSelector:1
Simple UPnP TV
france 3
Start
12:13
UPnP Control Point
Generated Simple TV UPnP Device
ON
OFF
0 1 2
3 4 5
6 7 8
- 0 +
Pocket PC

Figure 4. A composite controllet

A. Component Trading

For all new devices discovered in the UPnP network (as well as in the “micro-world”), the framework launches an trading operation to determine the most appropriate controllet (as well as the bridglet) for the discovered device.

Among the list of the available components, the trading algorithm eliminates firstly the components, which do not meet the environment constraints, such as the graphical toolkit or the architecture and operating system if the component has native libraries. The algorithm then seeks in priority the components the model and manufacturer attributes match with the device or service type in the candidates list. If no component related to the device model is found, the algorithm then seeks the specialized components (i.e. without model and manufacturer attributes) for the device or service type. Finally, if no component related to the type is selected, the trading algorithm returns the generic component by default. The generic component for a device aggregates and displays the controllets traded for its sub-devices and services. So the trading can recursively done for the sub-components. The generic component for a service lists the state variables and enables subscriptions for their notifications. The generic controllet displays a button and parameter fields for each action as most generic control points available in development toolkits (Intel, Siemens, Domoware, etc).

B. Component Composition

A component can itself be a composite, made up of sub-components in order to allow modular developments. These sub-components are related with sub-devices or device services. However, it is possible to consider other compositions related, for instance, to the versions of the service specifications, to the proprietary extensions or to the features available or disable across of product family. The sub-components can be statically defined at design-time or can be dynamically replaced at runtime according to the SOA approach. In the second case, the component uses in its turn the trader in order to seek the most appropriate sub-components. This composition by trading is very similar with the concept of component template introduced in the Fractal component model [14] to define generic assemblies of components.

C. *Component Deployment Policies*

The deployment operation can be carried out immediately when the device is discovered or may be delayed until the user starts to use the device. The first is known as immediate while second is known as on-demand since the deployment is done when the end-user demands to control the appliance.

The immediate policy has the major drawback of using the resources (CPU and RAM) of the control point or of the gateway even though the user never demands to use any of the discovered devices. It could even be impracticable when the number of different devices increases in the network. However, it has the advantage of being able to notify the user of state changes of the device (for instance, with tickers, blinking icons or pop-up alerts) as of its discovery by the control point.

A first solution was to statically enrich the controllet description with an attribute defining statically the policy that has to be used for the component deployment. A second solution is to design a controllet as a composition of 2 sub-components deployed in two steps: one is responsible for notifying changes to the user since the other for controlling the actions on the device. The first sub-component is deployed according to the immediate policy while the second oneaccording to the delayed policy. However, this solution does not simplify the controllet development and packaging and the sharing of the device internal representation between the 2 sub-components.

Finally, the deployment policy can be determined according the history of the devices actions that could be saved in the user context. The idea is if the user is used to a kind of device, then it is highly probable than he will use them again.

V. PROTOTYPE

This framework was validated through the development of a generic control point for PDAs and a "micro-world" gateway for an OneWire bus.

The control point and the "micro-world" gateway are both based on the OSGi framework [9]. The choice of OSGi for this prototyping was justified not only by the fact that OSGi makes it possible to conceive Java-based plugin applications, in which the plugins can be deployed and redeployed without starting the main application, but also by the fact that the OSGi specification reifies the UPnP devices in the form of OSGi services. Information needed to trading operations is gathered in an index (following a XML grammar) extending the deployment information used by the current deployment services of Oscar and Felix, two well-known open-source implementations of the OSGi specification. Our framework delegate component installation to the deployment service on the OSGi platform. This latter deploys the selected controllet or bridglet as well as its dependences (i.e. other bundles providing codes or services).

The PDA control point uses a lightweight graphic environment (ie AWT) in order to be supported by most of the JRE for PDAs. A main panel lists the icons of the UPnP device discovered (cf. thumbnail 1 in figure 5). Once the icon is selected by the user, the framework deploys the bundle (i.e. the deployment unit of OSGi) chosen by the trading operation. The bundle provides then a factory of objects implementing the Controllet interface for each device of the same model or type. The controllet for this control point implements the class java.awt.Component as well as a life cycle interface allowing the controllet iconification. It was tested on the HP iPAQ and Dell Axim with Oscar and Felix using the JVMs J9 and CReME for WinCE.

The "micro-world" gateway towards an OneWire bus (http://www.ibutton.com) controls sensors (temperature, hygrometry, etc) which can be connected and detected on the bus. Several brigdlets were developed for various types of OneWire sensors. The bridglet specialized for the temperature periodically polls the sensor and notifies changes. In the case of OneWire, the bridglet trading is carried out based on the model or the type (encoded on 8 bits) of the OneWire sensor (for example, 21 for Thermochron DS1921 and 23 for Hygrochron DS1923).

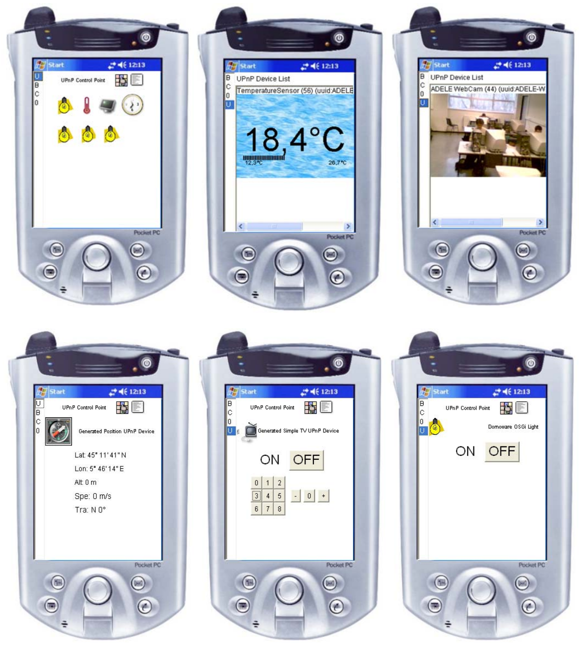

Figure 5. Controllets deployed on our PDA based generic control point

VI. RELATED WORKS

This work is found at the intersection of research on the software components [15], on the service-oriented dynamic architecture [7,8], on the deployment of software components [5] and on the context-aware and plastic GUIs [11,16]. This section positions our work in relation to several other similar developments or bringing solutions to the problems addressed.

Several research works and products were undertaken in the field of the software deployment. Carzaniga et al. [17] presents the field of software deployment and lists the main criteria for comparison. In the world of the Java applications, Java Web Start (JNLP), MIDP OTA, and OSGi (on which we have based our prototype) are among the most advanced. Java Web Start and MIDLet each only define the static dependencies to the libraries and the environment (i.e. packages included in the environment), which is not suited for the trading and dynamic deployment of the controllets and bridglets. The OSGi specification standardizes a service named Device Access, to enable the driver refinement and deployment on device detection. However, we think this service does not fit for the hierarchical composition of controllets. OMG D&C [18] specifies a very sophisticated component deployment framework. However it does not address the on-demand component trading at runtime. Some generic frameworks propose to select and deploy the "right" components according to contextual information [19] or quality of services constraints (memory usage, etc) [20]. Our trading algorithm could be refined by using such information about the environment.

The Gravity and Beanome projects [21] explore the GUI creation equipped with autonomous capabilities of dynamic adaptation based on components (graphic resources) available in the system. Gravity and Beanome based their work on a model with a component-oriented service, simplifying the task of developers in dynamism management of the arrivals and departures of components. Gravity and Beanome, however, only use the already deployed components and do not address dynamic trading. Comets (COntext of uses Mouldable widgETs) [22] are graphic components adapting their plastics according to the context and to the execution environment. They maybe composed like the controllets. Trading and deployment are not addressed by the Comets frameworks. So the controllets could be used as deployment containers for the comets.

VII. Conclusion.

This article presents a trading and on-demand deployment framework of software components intended for control points, controllets, and components intended for gateways, bridglets. These components are dynamically deployed (installed and activated) as soon as the device is discovered. These components follow the hierarchical structure of the UPnP devices and they may be made dynamically. A framework prototype was carried out based on the OSGi services dynamic platform. Examples of controllets and bridglets illustrating this paper can be found at http://www-adele.imag.fr/users/Didier.Donsez/dev/osgi.

Recently, MicroSoft had announced Vista Side Show which enables to have generic control points to DPWS devices [13]. So, a short-term perspective is to apply our deployment framework to the DPWS-enabled devices. Two other perspectives are considered. The first one concerns physical control devices such as phydgets [23]. The framework could be used for the deployment of the *phydgetlets*, which could be software components associated with the phydgets discovered dynamically. The second one concerns the "choreographies of devices" which compose the services provided by several devices [24]. For instance, the shutter can be closed when the user starts playing a DVD. The deployment of a given choreography could be triggered when the devices required for it appear in the network or sub-network.

References

[1] D. Marples, S. Moyer, "Home Networking and Appliances", in Diane Cook, Sajal Das, *Smart Environments: Technologies, Protocols and Applications*, Wiley (2004)

[2] D. Snoonian, "Smart Building", IEEE Spectrum, August 2003

[3] M. W. M. Feng Zhu and L. M. Ni., "Service Discovery in Pervasive Computing Environments," IEEE *Pervasive Computing*, vol. 4, no. 4, pp. 81–90, 2005.

[4] A. Dhir, "Home Networking Middleware", Xilink whitepaper, WP136 (v1.0) March 2001, http://direct.xilinx.com/

[5] UPnP Forum, "Understanding UPnP™: A White Paper", June 2000, http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc.

[6] M. Jeronimo, J. Weast, "UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play", Pub. Intel Press, ISBN: 0971786119, 2003.

[7] G. Bieber, J. Carpenter, Introduction to Service-Oriented Programming, *OpenWings whitepaper*, 2001, http://www.openwings.org/

[8] H. Cervantes and R. S. Hall: "Chapter I: Service Oriented Concepts and Technologies," *in the "Service-Oriented Software System Engineering: Challenges and Practices,"* (ISBN 1-59140-426-6) edited by Zoran Stojanovic and Ajantha Dahanayake, Idea Group Publishing, 2005

[9] OSGi Alliance, http://www.osgi.org

[10] D. Marples, P. Kriens, "The Open Services Gateway Initiative, an Introductory Overview", *IEEE Communications Magazine*, Dec. 2001.

[11] A.K. Dey, G.D. Abowd, "Towards a Better Understanding of Context and Context-Awareness" In the *Workshop on The What, Who, Where, When, and How of Context-Awareness*, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000)

[12] S. Meyer, A. Rakotonirainy, "A survey of research on context-aware homes". Proc. of the Australasian information security workshop conference on ACSW frontiers, 2003, pp 159–168.

[13] F. Jammes, A. Mensch, H. Smit, "Service-oriented device communications using the devices profile for web services", Proc. *3rd international workshop on Middleware for pervasive and ad-hoc computing*, Grenoble, France, Nov. 2005

[14] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, J-B. Stefani, "An Open Component Model and Its Support in Java", *7th International Symposium on Component-Based Software Engineering (CBSE)*, Edinburgh, UK, May 24-25, 2004, LNCS 3054 pp 7-22

[15] C. Szyperski, "Component Software: Beyond Object-Oriented Programming", Addison-Welsey, 1997.

[16] D. Thevenin, J. Coutaz, "Plasticity of User Interfaces: Framework and Research Agenda". Proc. Interact99, Edinburgh, Eds, IFIP IOS Press Publ., 1999, 110–117

[17] A. Carzaniga, A. Fuggetta, R.S. Hall, A. Van Der Hoek, D. Heimbigner, A.L. Wolf, "A Characterization Framework for Software Deployment Technologies". University of Colorado Tech. Rep CU-CS-857-98, 1998.

[18] Object Management Group, Deployment and Configuration Distributed Applications Specification, http://www.omg.org/docs/ptc/04-08-02.pdf

[19] D. Ayed, C. Taconet, G. Bernard, "A Data Model for Context-aware Deployment of Component-based Applications onto Distributed Systems". *ECOOP '04 Workshop on Component-oriented Approaches to Context-aware Computing*, Oslo, Norway, June 14-19, 2004.

[20] J-C. Tournier, V. Olive, v Babau, "Qinna, an Component-Based QoS Architecture". *8th International Symposium on Component-Based Software Engineering (CBSE)*, Saint-Louis, USA, June 2005

[21] H. Cervantes, R.S. Hall, "Automating Service Dependency Management in a Service-Oriented Component Model", *6th International Symposium on Component-Based Software Engineering (CBSE),* Portland, OR, 2003

[22] G. Calvary, J. Coutaz, O. Dâassi, L. Balme, A. Demeure, "Towards a new generation of widgets for supporting software plasticity: the « comet »", *EHCI-DSVIS'2004*, Hamburg, Germany, LNCS 3425, 2004

[23] S. Greenberg, M. Boyle, "Customizable physical interfaces for interacting with conventional applications". Video Proceedings of the ACM UIST 2002 15th Annual ACM Symposium on User Interface Software and Technology. ACM Press.

[24] W. Trumler, F. Bagci, J. Petzold, T. Ungerer, "Smart Doorplates - Toward an Autonomic Computing System", 5*th Intl WS on Active Middleware Services*, June 2003, Seattle, WA, USA pp 42-47.