An Extensible Ubiquitous Architecture for Networked Devices in Smart Living Environments

Thierry Bodhuin, Gerardo Canfora, Rosa Preziosi, and Maria Tortorella

RCOST - Research Centre On Software Technology,
Department of Engineering, University of Sannio,
Via Traiano, Palazzo ex-Poste – 82100, Benevento, Italy
{bodhuin, canfora, preziosi, tortorella}@unisannio.it

Abstract. Continuous technological innovation is entailing that living environments be equipped with products that improve the quality of daily life. Unluckily, the adopted solutions do not always represent an adequate support and people continue to execute repetitive tasks that software infrastructures could perform automatically. This is partially due to the fact that the existent technological solutions cannot be always integrated in a coherent communication platform, as they use proprietary protocols and ad hoc implementations not easily reusable. This paper proposes an extensible ubiquitous architecture for networked virtualized devices in smart living environments. The aim is realizing ubiquitous applications and integrating networked devices through an architecture that hides their complexity and heterogeneity. Several intelligence techniques have been integrated for offering a smart environment through the use of automatic learning techniques.

1 Introduction

People thirst for technological products helping them to have a better quality of the everyday life. They equip with these products their professional, personal, transit, transport, and so on, living environments. Academic and industrial world feel inclined to promote technological progress and terms as *home automation*, *domotic system*, *smart home* are diffusing in the industry, while expressions as *pervasive computing*, *ubiquitous computing*, *nomadic computing*, *ambient intelligent*, *context-aware computing*, *augmentation of the real world*, indicate academic research topics. The available technologies do not always represent an adequate support as they are often unable to interact with other products made by different makers and/or based on different solutions. Their communication and integration too often requires human intervention, and people are discouraged by the complexity of the new Information and Communication Technologies (ICT) facilities and by the redundancy of the needed administrative and configuration activities. In addition, the use of proprietary communication protocols in software architectures for smart environments does not facilitate the interoperability of the networked devices and the reusability and maintainability of software packages forming part of the architecture. This forces developers to perform repetitive implementation tasks.

T. Enokido et al. (Eds.): EUC Workshops 2005, LNCS 3823, pp. 21 – 30, 2005.

The work presented in this paper has been carried on within the *Demonstrator* project of the Regional Centre of Competence in Information and Communication Technology, CRdC ICT. This Centre involves many researchers and industrial partners of the Campania Region in Italy. It aims at analysing, defining and realizing hardware and software platforms for permitting the provision of networked services and the implementation of advanced technologies. In particular, the activities carried on in the unit of the University of Sannio, RCOST (Research Centre On Software Technology), aim at developing a platform in the field of home automation that is endowed with different levels of intelligence. It addresses the following aspects: *virtualization of devices*, for defining a generic functional characterization of the networked devices, making the applications independent from the characteristics of a particular device and supporting implementation tasks of software developers; *abstract description of devices*, for defining a semantic characterization of the networked devices, making applications more aware of the triggered actions in the physical world and supporting human intervention and interaction; *abstract description of services*, for providing a functional and semantic characterization of the services with reference to their relations with the other services and devices.

The proposed software architecture aims at facilitating the interoperability of networked devices, based on different technologies, and produced by different manufacturers; offering a middleware supporting different levels of intelligence as awareness, reactivity and adaptiveness; and permitting to activate services, through suitable applications respect to the typology of the client accessing it.

In the following, Section 2 presents some related work, Section 3 describes the software architecture, Section 4 discusses an example of virtualization, and the final section summarizes the main conclusions and sketches future directions of research.

2 Related Work

The increasing request of telecommunication solutions conducted to the development of sophisticated networked heterogeneous devices, supporting one or more of the available communication protocols (e.g., X-10, EIB, LonWorks, Ethernet-TCP/IP) and/or service and discovery-focused standards (e.g., HAVi, Jini, OSGi, UPnP). Currently, these standards are complementary, rather than competitive, even if they are sometimes partially overlapped in some provided facilities. The use of networked devices supporting different protocols and standards requires the adoption of more complex networking techniques, facilitating the interaction and interoperability of the devices and their accessibility from both local-area and wide-area networks.

In this scenario, it would be expected that different interconnected networks, supporting distinct features of smart living environments, exist. Consequently, manufacturers of different communication technologies, such as LonWorks and EIB, continuously upgrade their systems for increasing the reciprocal interoperability [4] and allowing devices from different vendors to communicate each other. However, the communication between devices is still not supported [4, 10, 11] in many cases. For example, it is possible to find living environments including EIB controlled devices, Ethernet networked devices and Bluetooth mobile devices, but it is unlikely to find living environments where other components, such as a X-10 and a EIB controlled lamp, interoperate.

In many cases, the effort addresses the integration of various physical elements, including sensors, actuators, microcontrollers, computers and connectors [5]. But, many of the proposed solution are mostly manual and ad-hoc, lack of scalability and are too close to the third parties. Likewise, each time a new component is inserted into the considered space, conflicts and uncertain behaviours may be verified in the overall system, requiring programming and testing interventions. For facing these problems, a middleware automating integration task is required for ensuring pervasive space openness and extensibility [6]. It must enable programmers to dynamically integrate devices without interacting with the physical world, and, then, decouple programming tasks from construction and integration of physical devices.

The typical approach that is applied regards the connection of sensor-actuator devices using classical network infrastructures, such as OSI, CORBA, and so on, at a low level. Unfortunately, the use of these kinds of infrastructures does not ease the integration of the devices. The approach in [12] is based on the integration of the devices at high-level, and ad-hoc networking infrastructures that dynamically integrate sensors and actuators into complex interactive systems while providing services and interfaces.

The architectural design presented in this paper has been defined for partially solving the problems of integrating devices, and for controlling and monitoring personal living environments from heterogeneous terminals. It considers requirements of *interoperability*, *portability*, *extensibility*, *reusability* and *maintainability* from the developer's point of view and *usability* and *adaptability* from the end-user's point of view. In addition, the proposed solution is based on the *OSGi* (*Open Service Gateway initiative*) [9], an emergent open architecture, which permits the deployment of a large array of wide-area-network services to local network services such as smart homes and automobile [5]. OSGi defines a lightweight framework for delivering and executing service-oriented applications. It presents advantages, such as: platform independence, different levels of system security, hosting of multiple services and support for multiple home-networking technologies.

3 Extensible and Ubiquitous Architectural Design

Figure 1 shows the proposed extensible ubiquitous architectural design. The various layers are grouped in six levels, going from **A** to **F**, and they are next presented.

3.1 Levels F, E, D

Level **F** in Figure 1 depicts the heterogeneous networked devices to be accessed. They may be produced from different manufacturers and/or using different communication protocols and, service and discovery-focused standards. Level **E** includes the needed drivers, grouped in two layers: a hardware layer and a layer of network IP cards, audio cards, RS-232 ports, etc., necessary for connecting the devices of level **F**. Level **D** concerns the portability of the implemented software and includes the operating system and the Java Virtual Machine (JVM).

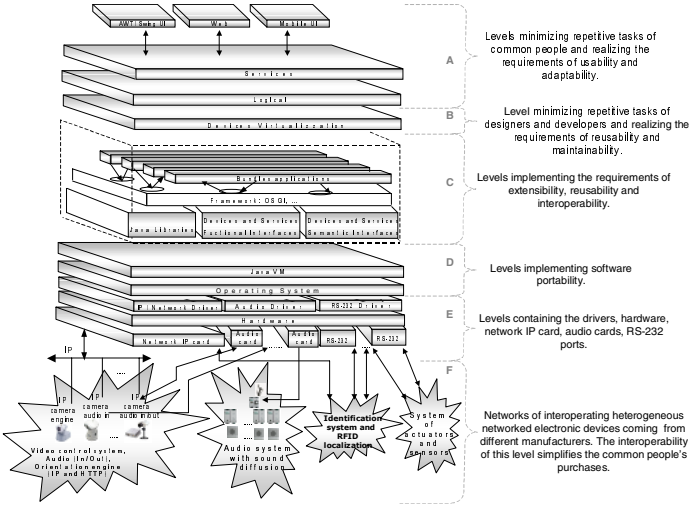

Fig. 1. Extensible and ubiquitous architectural design

3.2 Levels A, B, C

Levels **A**, **B** and **C** form the *Domus intelligent Keeper* (*DiK*) software infrastructure. *DiK* is composed of three main components: a *framework* component, which aims at minimizing the activities of developers and helping the extensibility and ubiquity capability of the architecture; a *service oriented applications* component which uses the framework and aims at simplifying and minimizing human intervention and interaction activities; and an *intelligence* component aiming at decreasing repetitive daily activities and facilitating the automatic evolution of the software infrastructure, on the basis of people's continuous changing habits and modifications of the networked devices adopted in the living environments.

Level **C** assures the characteristic of interoperability of the proposed architectural design. It includes the *OSGi* (*Open Service Gateway initiative*) Framework [9] enabling the connectivity and management of the devices based on different transmission technologies. It defines a platform model where the software applications are installed and executed. These applications are Java archives, called *bundles*, which cooperate to the implementation of a service. The OSGi Framework represents a common environment hosting bundles. The bundles use: the resource of the OSGi Framework, all the standard Java libraries, virtualized devices and service interfaces. In addition, they access level **E** and, consequently, monitor and control the networked devices of level **F**. In particular, the OSGi Framework is the part that changes a JVM from a single application environment into a multiple one. The advantages are many. Actually, running multiple applications in a single JVM means less process swaps, fast inter-application communication, and significantly less memory consumption. Moreover, the OSGi Framework makes possible the interoperability among different devices, service providers, network operators, service gateway manufacturers, and home appliance manufacturers. Level **C** in Figure 1 manages the life cycle of the bundles and solves their interdependence, keeps a

registry of services and manages the events informing the listeners when the state of a bundle is changed, when a service is stored and when an error occurs. Besides the usage of OSGi bundles, level **C** includes an alternative device access solution based on the *Java Remote Method Invocation* (*RMI*) and the Jini technology. RMI/Jini and OSGi solutions are not the only ones to be considered for level **C**, as this level is a dynamic container with changeable content according to the technological progress so that it can deliver access to services over any network and protocol. Level **C** was developed with the intention of ensuring the satisfaction of the extensibility requirement. This aspect is strongly related to the capability of evolving the software when new technologies are introduced and needs of the end-user change. The extensibility requirement is also preserved by the usage of class libraries in the level **C**. In this manner, developers can take advantage from the object-oriented techniques, which facilitate a more modular designing and encourage the use of constructs related to inheritability for better organizing the source code, avoiding repetitions, gaining time and reducing development costs.

Level **B**, including the *Devices Virtualization layer*, is located between the bundles of the OSGi Framework and Level **A** of the services accessible from the user. Its objective is to provide an abstraction of the devices of level **F**, by generalizing their behaviour independently from their identity (or type), nature and communication protocol, and hiding the complexity of the reciprocal communications. In particular, two different devices have different identity expressed from a set of attributes like: *name*, *serial*, *version*, *model*, *manufacturer*, etc. Two devices with different nature are logically connected to two distinct physical concepts. Nevertheless, two different devices with distinct type and nature may share the same actuation mechanism. For example, a networked lamp is a device different from an alarm. The lamp is logically connected to the *electric light* concept and may change the state of the environment where it is installed by providing, or not providing, light on the basis of the switch *on/off* actuation mechanism. The alarm is logically connected to the *sound* concept and may change the state of the environment hosting it by providing or not providing noise in accordance with its *open/close* actuation mechanism. The lamp and alarm are devices of different identity, nature and semantic, but share an actuation mechanism with the same working procedure. So, it is possible to extract a functional view permitting a first classification of the devices grouping them in two families: *Sensors*, capturing information from the networked devices and/or the environments, and producing events; *Actuators*, consuming events and, triggering actions on the networked devices in the considered environments. Sensors and Actuators can be still specialized in other objects. For example, the networked rolling shutter has a mechanism of actuation different from that of the networked lamp and alarm. It cannot be defined by two values but considering a set of valid values. For instance, the rolling shutter may have five possible valid values, *absent*, *low*, *medium*, *high*, *highest*, modelling five different positions and brightness degrees. Besides the Sensors and Actuators, complex devices exist in the living environments. They are the result of the composition of more elementary devices. For instance, a camera is defined as a complex device with different elementary actuation mechanism related to different functionalities, as later described. Figure 2 exhibits a view of the device interface hierarchy. It shows that the specialization of the generic devices of type Sensor and Actuator is reasonable. For example, the networked lamp is a device of

Actuator type, which can be described by a *BinaryActuator* interface, able to assume only two valid values. While the rolling shutter is a device of Actuator type describable by a *SetValuesActuator* interface being able to assume different discrete defined values. Furthermore, a device with values inside a given continuous range can be characterized by a *RangeValuesActuator* interface. Besides those discussed, further specialization levels can be identified. In addition, Figure 2 highlights that Interface *Device* is characterized by methods adding/removing the *EventListener* objects and used from clients for registering/un-registering a listener in *Device*. Thus, clients can be notified in a push way of changes in the state of the devices for taking their decisions. Listener and event interface hierarchies are also defined. Moreover, interface *Device* is characterized by getting/ setting methods for accessing and/or manipulating the *identity* of a considered device. The identity information is maintained in the logical layer and its handling is a first step toward the modelling of devices that considers the semantic aspect.

The interface hierarchy shown in Figure 2 is not complete. It permits the realization of reusable software components. Furthermore, the Devices Virtualization layer is still valid, even when the hierarchy is extended for including new devices, independently from their complexity.

Finally, level **A** groups the layers oriented to minimize the work of the end-user. In fact, they allow *DiK* to adapt a personal living environment to the needs of common people and/or situations and to simplify the human interaction. Level **A** includes three layers named *Logical, Services* and *User Interface* (*UI*).

Logical layer manages and maintains the information regarding the logical *internal* characterization of each networked electronic device and the optional logical *external* characterization. The internal characterization of a device is defined by its datasheet, while the external one is described by the *logical connections* between the considered device and the physical concepts it can affect. The physical concepts are attributes characterizing the environment that is *external* to the device. For example, a networked rolling shutter is a device internally characterized by the *raising* behaviour.

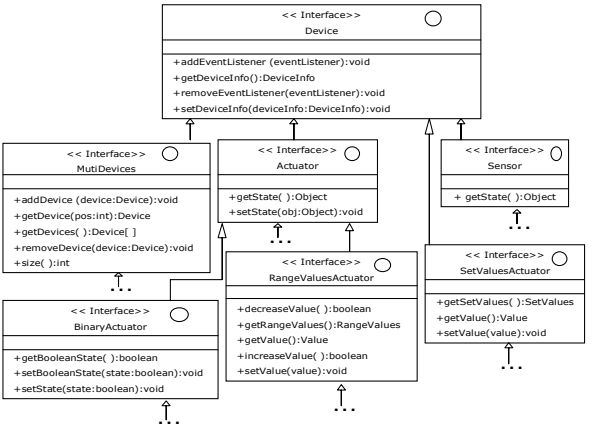

Fig. 2. A simplified view of device interface hierarchy

This behaviour allows the rolling shutter to be (un)rolled at a given grade. In this way, it allows one to change the state of brightness of a given environment. Therefore a logical connection exists between the cited device and the *solar light* physical concept, which represents its external characterization.

Services layer aggregates functionalities exported from single networked devices for providing services that are able to promote comfort, safety, security, initial minimization of human intervention and improved lifestyle for residential end-users. For instance, if an *illumination control service* exists in a house, it could promote comfort in terms of luminosity, while reducing associated cost for producing light in the area where the householder is located. This service may use: any localization sensor (e.g., presence sensor, RFID identification) for recognizing human presence in a given area; a luminosity sensor for knowing if a given luminosity threshold is achieved; and light actuators, like dimmer or on/off lamp, for reaching the light condition requested. Aggregating the functionality of networked curtains, rolling shutters and lamps allow the reduction of the associated cost for reaching a certain luminosity level, corresponding to the householder wished level of comfort. The control services use standard control mechanisms with loop control. However, in the context of home automation networks with slow action to effect, the control services were enhanced by using neural network for learning the relation between wishes (e.g., light condition), context (e.g., sensors, time, occupants), and possible actuations on the different actuator devices, that are located in the area where the service control takes place and are connected with the interesting physical aspect (e.g., devices connected with the illumination aspect). The use of a neural network allows the control services to achieve more rapidly their objective on slow networks and/or slow action/effect relation. In addition the Service layer includes a group of intelligent services permitting to support different levels of intelligence: context-aware, automated, reactive, adaptive. Whatever intelligence type might be, it requires the measurement and collection of data, as well as the extraction, aggregation and abstraction of information. The progress made in hardware technology allows storing very large amounts of data, while abstracting valuable information is still a very difficult task. This task is more difficult when applied to data collected when the people interact with devices and services in the living environments. A high degree of randomness in the real human life is source of high complexity.

Despite the high degree of randomness, it is possible to identify patterns in the person's life [3]. Patterns may represent regular repetitive interactions of the people with the networked devices. People have habits that are usually sampled in time and inter-connected with the other people' habits through various constraints, which are dependent on the current role and activities that people have when they use the devices and services of the actual environment. A person's life can be "sampled" on the basis of the hours, days, week days or week-ends, seasons, and so on. Human living environments can be "sampled" based on the location, room or areas, where federations of devices and persons are defined. The repetition of these patterns may have a high or low frequency according to the variability of the person's life. These facts suggest that person's life in human living environments can be automatically "photographed" and patterns captured, processed and transformed in rules for enabling control systems and autonomously acting, while remaining unobtrusive, in addressing people's needs by requesting user's feedback.

One important component of the intelligent services group is a rule engine named Jess [8] that allow the execution of rules describing relations between events and actions. The rules may be created by smart environment users, or be automatically generated by a learning system that was developed on the basis of the WEKA (Waikato Environment for Knowledge Analysis) tool [14]. This tool provides a suite of facilities for applying data mining techniques to large data sets for supporting various tasks including classification, market basket analysis (MBA or association rules), prediction. Currently, MBA algorithms are used for analyzing end-user patterns.

User Interface layer allows a transparent access to heterogeneous networked devices installed in living environments from interface AWT/Swing, Web and mobile.

4 An Example

Figure 3 depicts an example of virtualization. It refers to an Axis Video Camera with Pan/Tilt and Zoom functionalities [1]. The figure is organized in three blocks going from **a)** to **c)**.

Block **b)** shows that the Java *AxisVideoCamera* class is implemented as a specialization of the *MultiDevices* class. In particular, it is composed of the following parts: six *RangeValuesActuators*, which are specializations of the *Actuator* class and virtualize the Pan, Tilt, Zoom, Iris, Focus and Frame/sec functionalities; one *SetValuesActuator*, which is a specialization of the *Actuator* class and virtualizes the preset position functionality; two BinaryActuators, which are specializations of the Actuator class and virtualize the Auto iris and Auto focus functionalities; one *VideoSensor,* virtualizing the video functionality as a specialization of the *Sensor* class. The specific implementation of *VideoSensor* for the Axis Video Camera includes the implementation of the Java Media framework DataSource [7] for the encapsulation of the MJPEG format provided by the Web server included in the Internet video camera. All the implemented classes include the functional code needed for the communication between the specific Actuators and Sensors and the physical Axis Camera, in accordance to the contract between the device implementation and their clients. Further, they exhibit suitable interfaces, exemplified by the **c)** block, to the client objects. The implementation of the considered Axis Camera uses the same actuation mechanisms adopted in other devices, such as the networked lamp, alarm and rolling shutter, but with a different semantic specification.

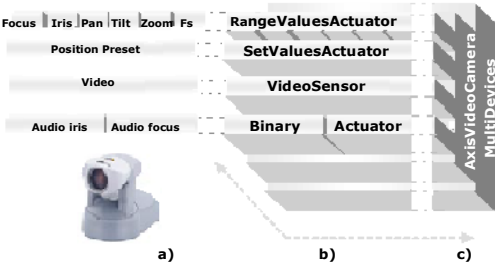

Fig. 3. Camera virtualization: a) mask of functionalities; b) implemented classes; c) interfaces

Therefore, a new complex device, different from the Axis camera, can be obtained simply changing the mask of the functionality shown in block **a)**. This is possible thanks to the device virtualization process. In particular, the implementation of the defined classes are generic and provide a generic implementation of the methods for getting and setting the state of a device, for the (un)registering of listeners and events notification, related to the modification of the state of a device. Therefore, the device virtualization process simplifies the reuse of the generic parts of the devices and the mechanisms notifying change events to the listeners. When the implemented classes and their interfaces are introduced in the framework of the architectural design shown in Figure 1, it is possible to get and modify the video camera state trough any kind of user interfaces. For example, the Axis Video camera provides an http network protocol interface. Getting and modifying the video camera state (e.g., Rotating the video camera in PAN/TILT or Zooming), could be performed through the http interface, and connecting to the URL http://*videocamerahost*/axis-cgi/com/ptz.cgi?autofocus=on, sets the state of the *BinaryActuator* regarding the AutoFocus to ON. The Video source is acquired in a MJPEG format from an http connection to the networked video camera (e.g., http://*videocamerahost*/axis-cgi/mjpg/video.cgi). This video source is encapsulated inside a Java Media Framework DataSource for facilitating its integration with the video/audio streaming and the visualization utilities offered by the Java Media Framework. The video source is transmitted by using the Real Time Streaming protocol for permitting its visualization through unicast or multicast connection and in on-demand way. The device virtualization process also simplifies the implementation of the device remotization for letting it be accessible in a remote way by using a RMI interface. The actual protocol between the RMI client and server is defined through the Jini Extensible Remote Invocation [13] that permits the use of protocols different from the Java specific one, named JRMP.

The described implementation was tested with frame rate of more than 30 frame/sec through Real Time Streaming protocol and replicated with a D-Link DCS 2100+ Wireless Internet Audio/Video Camera, providing the video and audio without the Pan/Tilt and Zoom functionalities.

5 Conclusions and Future Work

This paper describes an extensible ubiquitous layered architectural design for smart living environments supporting different levels of intelligence. The technologies used for developing it, were already used with success in other projects in the ubiquitous computing context. The main difference respect to the previous usage consists of the existence of the **B** layer. It contains the Devices Virtualization layer and is oriented to decouple the **A** layers from layers below it. So, several technologies can be integrated for providing an architecture that is open to different makers and adequately supports the developers implementation tasks and decisions of the users that can feel free to buy and insert different new devices in their living environments and make them operative. Devices Virtualization layer aims at defining a framework for easily developing services, by decoupling the physical devices from the clients accessing them, and offering a middleware that permits the activation of a service, choosing a suitable user interface implementation with reference to the type of client accessing it. Further, this layer enables DiK to better survive to the changes due to the

technological progress. This aspect is very important when a software system with unstable requirements has to be developed. This is the case of the applications for living environments, where people's habits continuously change together with the physical devices to be used and integrated.

The need of a semantic characterization for networked devices was also highlighted, for addressing the dynamic discovery of devices and services, promoting comfort, safety, security, communication, and so on. This aspect is deepened in [2]. It required investigation in using ontology and specialized representation mechanisms of contextual information for ubiquitous systems. Finally, the Intelligence services were developed to achieve automatic generation of rules based on the finding of patterns in the interaction between users and devices/services in the smart living environment. Another Intelligence service regarded finding the relations for each physical aspect (e.g., light, temperature) between sensor level target and possible actuations considering constraints like cost saving. The Intelligence services use data-mining and neural networks techniques and apply them for achieving smart living environments without creating autonomous and non-manageable or understandable environment.

Future work will be considered in the field of embedded software in hardware devices with distributed infrastructure and intelligence. The aim is to support the cooperation between these devices to reach some comfort level based on the living environment occupants without needing of a semi-centralized architecture.

References

1. Axis Communications: Axis Networked Video Camera. http://www.axis.com/products/cam_213/
2. Bodhuin, T., Canfora, G., Preziosi, R., Tortorella, M.: Hiding complexity and heterogeneity of the physical world in smart living environments. Submitted. Available from the authors (2005)
3. Eagle, N., Pentland, A.: Reality Mining: Sensing Complex Social Systems, J. of Personal and Ubiquitous Computing. To appear (2005). http://reality.media.mit.edu/pdfs/realitymining.pdf
4. Fuertes, C. T.: Automation System Perception-First Step towards Perceptive Awareness Dissertation. Institute of Computer Technology, TU Wien (July 2003)
5. Gu, T., Pung, H.K., Zhang, D. Q.: Toward an OSGi-Based Infrastructure for Context Aware Applications, IEEE Pervasive Computing, Vol.3, No.4 (October-December 2004) 66-74
6. Helal, S.: Programming Pervasive Spaces, IEEE Pervasive Computing, Vol.4, No.1 (January-March 2005) 84-87.
7. JavaSoft: Java Media Framework. http://java.sun.com/products/java-media/jmf/index.jsp
8. Sandia National Laboratories: Java Expert System Shell. http://herzberg.ca.sandia.gov/jess
9. Open Service Gateway Initiative: The Open Service Gateway. http://www.osgi.org
10. Russ, G.: Situation-dependent behaviour in building automation. Dissertation, Institute of Computer Technology, TU Wien (2003)
11. Russ, G., Dietrich, D., Tamarit, C.: Situation Dependent Behaviour in Building Automation. Proceedings of Workshop EurAsia-ICT 2002, Advances in Information and Communication Technology, Shiraz, Iran (2002) 319-323
12. Schramm, P., Naroska, E., Resch, P., Platte, J. Linde, H. , Stromberg, G. and T. Sturm,: A Service Gateway for Networked Sensor Systems, IEEE Pervasive Computing, Vol.3, No.1 (January-March 2004) 66-74
13. Sommers, F.: Call on extensible RMI – An Introduction to JERI, JavaWorld. http://www.javaworld.com/javaworld/jw-12-2003/jw-1219-jiniology_p.html (2003)
14. Waikato Environment for Knowledge Analysis Project. http://www.cs.waikato.ac.nz/~ml/