Tangible Applications for Regular Objects: An End-User Model for Pervasive Computing at Home

Spyros Lalis[1], Jarosław Domaszewicz[2], Aleksander Pruszkowski[2], Tomasz Paczesny[2], Mikko Ala-Louko[3], Markus Taumberger[3], Giorgis Georgakoudis[1], Kostas Lekkas[1]

[1] CERETETH & University of Thessaly Volos, Greece {lalis, ggeorgak, kolekkas}@inf.uth.gr

[2] Institute of Telecommunications Warsaw University of Technology Warsaw, Poland {domaszew, apruszko, t.paczesny}@tele.pw.edu.pl

[3] VTT Technical Research Centre of Finland Oulu, Finland {mikko.ala-louko, markus.taumberger}@vtt.fi

***Abstract*—This paper describes an end-user model for a domestic pervasive computing platform formed by regular home objects. The platform does not rely on pre-planned infrastructure; instead, it exploits objects that are already available in the home and exposes their joint sensing, actuating and computing capabilities to home automation applications. We advocate an incremental process of the platform formation and introduce tangible, object-like artifacts for representing important platform functions. One of those artifacts, the application pill, is a tiny object with a minimal user interface, used to carry the application, as well as to start and stop its execution and provide hints about its operational status. We also emphasize streamlining the user's interaction with the platform. The user engages any UI-capable object of his choice to configure applications, while applications issue notifications and alerts exploiting whichever available objects can be used for that purpose. Finally, the paper briefly describes an actual implementation of the presented end-user model.**

*Keywords—**Sensor and actuator networks, ubiquitous and pervasive computing, smart homes, system and application management, user interaction, tangible interfaces.***

I. Introduction

The continuous technological developments in the area of embedded computing and networking make it possible to digitally augment regular home objects with computing, sensing, actuation and communication capabilities, making them not only smart but also capable of cooperation with each other. In the near future, the household is likely to be populated with a host of such objects, ranging from usual appliances like a refrigerator, an electric kettle, or a TV, to infrastructural elements like doors, windows, and lamps, down to small devices such as temperature sensors, smoke detectors, and motion sensors.

Significant potential for advanced functionality can be created by transforming a collection of digitally-augmented regular objects into an open pervasive computing platform that allows home automation applications to exploit the different sensing and actuation capabilities of participating objects in a combined way. For instance, one application could employ temperature sensors and smoke detectors to infer the presence of fire. Another application could save on the electricity bill by controlling the operation of lights and appliances based on the user's activity and demand-response offers of the electric utility. Yet another application could double check that a window is not left open unintentionally while the thermostat setpoint for the heater located in the same room is above the outside temperature.

A multi-object computing platform, as described above, can be implemented by letting the nodes embedded in objects expose the local sensing and actuating capabilities in a suitable way, as well as communicate with each other to provide other middleware-level services to the applications. However, the underlying software and hardware is only a part of the challenge. An equally important aspect is to consider how the end-user perceives and interacts with such a platform. By no means should such a platform be yet another user-attention hungry technology, introducing complex or awkward processes of installation, configuration and administration. This is absolutely crucial if one wishes for it to be embraced by the general public.

This paper describes an end-user model for multi-object computing platforms based on regular digitally-augmented home objects. In the spirit of ubiquitous computing [1], our work is based on the premise that the platform should require the end-user to expend as little mental energy (and be bothered with explicit manual input and intervention) as possible. The main contributions of the end-user model are as follows. First, we advocate a low-profile, incremental process of platform formation. Second, we introduce tangible, object-like artifacts, such as the *community key* and the *application pill*, to represent important platform entities and functions. Third, we streamline the conventional user interaction with the platform, for the cases that cannot be handled using these special objects. The paper also describes a concrete implementation which is being pursued in the POBICOS project [2]. Notably, the presented end-user model is largely platform-independent and could be realized using different combinations of networking, hardware and software technologies.

The rest of the paper is organized as follows. Section II gives an indicative scenario and lists the main elements of the envisioned multi-object pervasive computing platform.

Section III introduces the end-user model, with focus on the special tangible artifacts and the aspect of user interaction. Finally, Section IV outlines the implementation in the POBICOS project, and Section V discusses related work.

II. VISION

Our vision of how pervasive computing could be accomplished in the home based on regular objects is illustrated via the following scenario:

Maria and Peter decide to buy some new appliances. While browsing the stores they notice that some items have a “community-enabled” sticker. A salesperson explains that this is a new technology which makes it possible for regular objects to cooperate. The couple decides to buy a kitchen stove and a TV. They are also given a special community-enabled “key” object for free. At home, after reading the (surprisingly short) manual, they bring the key object close to the TV and press a button to register it with the platform. The process is repeated for the stove. As nothing fancy happens, the couple quickly forgets about this technology.

Weeks later Peter buys a cook book. He notices that it comes with a small community-enabled object labeled as “the new home safety application pill by CoolApps Ltd”. He registers the pill object following the usual process, and then pushes a button on the pill to start the application.

One day Peter is baking a cake. He goes to the balcony to get some fresh air and stays there for a while. Suddenly, he hears a rather unusual alarm tone coming from the bedroom. As Peter enters the house, he sees a message on the TV screen informing him about a problem with the stove. He rushes back to the kitchen and is relieved to see that the stove turned itself off just before his cake was about to turn into coal. Peter recalls that some time ago Maria bought a new community-enabled enabled alarm clock for their bedroom and, fortunately, registered it with the platform.

Peter recalls that, according to the manual, the home safety application comes with some pre-set parameters that can be modified to customize its behavior. Peter uses the TV to browse these settings, and decides to change the default policy for alerts to enable the engagement of voice messages.

This simple scenario captures, to a large extent, several key elements of our vision. These are described in more detail in the following.

A. *Unplanned, incremental formation from regular objects*

The user forms the multi-object computing platform in an incremental way, by adding objects to it. This can be done at any point in time and without thinking about the objects’ digital augmentation, a particular platform configuration, or a specific application. The user buys objects in order to employ them according to their natural functionality (a lamp is bought and placed at a particular location to light that area), not because they can contribute to the platform. Most often, the user is not even aware of the capabilities the object may provide to the platform. Contrary to a system that is engineered for a specific purpose, there is no a priori specified arrangement or reliance on infrastructure.

B. *Open, multi-application platform*

The user can add new and remove existing applications at any point in time. Multiple applications may co-exist and run concurrently, subject to the resource constraints of the objects that make up the platform. Like in conventional systems, applications are typically developed by third parties that are not affiliated with object manufacturers.

C. *Tangible artifacts for straightforward administration*

Special, object-like, physical artifacts are used to embody important platform entities and functions which the user should be aware of and to which the user should have immediate access. For instance the “key” is required to add and remove objects to/from the platform and the “application pill” is used to start/stop the execution of a particular application. Making special entities and functions tangible and representing each of them with a different physical object relieves mental ambiguities (as to which object should be used to perform a function) and simplifies interaction (a dedicated, single-function object can have a tuned interface compared to a general-purpose object that is loaded, perhaps even over-loaded, with several different functions).

D. *Streamlined user interaction*

Ideally, user interaction occurs solely via the tangible artifacts introduced for platform formation and application management. However, in practice, additional interaction is often needed: (i) to let applications notify or alert the user; (ii) to let the user configure applications. The former is a one-way communication towards the user; the expected user reaction is to act in the real world, not to interact with the platform or application. In the latter case, the user, not the platform, is in charge of the interaction, i.e., the user chooses when to engage in the interaction and which object to use to do the setup. Importantly, in both cases, the platform is self-contained, relying on the native interaction capabilities of regular objects that are already available in the home. There is no reliance on computer-like objects such as a PC, a PDA or a mobile phone. While such objects are allowed to participate in the platform, they are not required to support user interaction.

III. END-USER MODEL

Along the lines of Section II, we propose an end-user model that describes, in a more formal and structured way, how the user perceives and interacts with the platform. The model consists of (i) basic terminology, (ii) special objects the user must be aware of, and (iii) a generic interaction pattern for the more conventional aspects of user interaction with the platform. The model is presented in a canonical way, striving for a clean separation of entities, roles and functionalities. Relevant use cases are described with reference to the scenario given in Section II.

A. *Basic terminology*

1) *Community-enabled object:* a regular object that provides sensing, actuation, and computing capabilities to the platform. Community-enabled objects can be marked,

e.g., with a sticker, so that the user can distinguish them from objects that are not community-enabled.

2) *Object community:* a collection of community-enabled objects in a home participating in the same platform (Figure 1). An object community is formed by adding and removing community-enabled objects in an explicit yet dynamic fashion. It represents a well-defined scope in terms of security vis-à-vis objects that are not part of the community, as well as in terms of the operational range of applications that run in the community.

B. *Tangible artifacts*

1) *The community key object:* The key object is used to add and remove other objects to/from the community (it also generates and transfers security-related keys and credentials in the background). It is the first object that must be acquired and it is mandatory to form an object community and to control its membership. The prototypical user interface for the key object is (Figure 2a): (i) a keypad for entering the name and PIN for an object community; (ii) two buttons, for triggering the addition and removal of objects; (iii) a LED for indicating the status and result of the last action; and (iv) close range communication ability with other objects for exchanging data in a safe manner without requiring a shared secret.

Use case: Initializing the key object

Peter and Maria switch on the community key for the first time. The LED on the key turns red. They enter a name and a PIN of their choice for their object community. The LED turns green, indicating it is ready to be used.

Use case: Adding an object to the community

Maria switches on the key and enters the name and PIN of the object community. The LED on the key turns green. She brings the key close to a newly purchased, community-enabled alarm clock and presses the “add” button. The LED on the key blinks for a few seconds and then turns green. Maria successfully added an object to the community.

Use case: Removing an object from the community

Peter turns on the key and enters the name and PIN. He brings the key close to the alarm clock and presses the “remove” button. The LED on the key blinks for a few seconds and then turns green. Peter successfully removed the alarm clock from the community.

.

2) *The application pill object:* Each applications is packaged in a distinct community-enabled object, called the application pill. The pill serves as a deployment and control vehicle for the application: it is used to start/stop application execution in the community and provides basic status information about the operation of the application. The user conceptually identifies the pill which the application itself; in other words, for the user, the pill *is* the application.

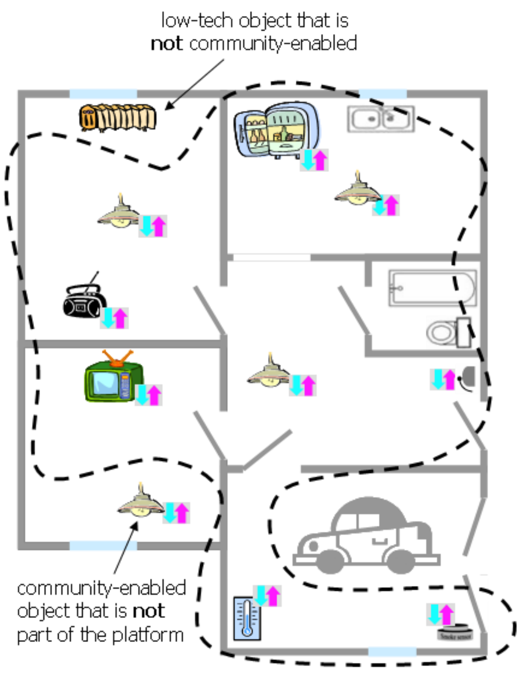


Figure 1. An indicative object community: bidirectional arrows indicate community-enabled objects, the dashed line around objects indicates the boundary of the community.

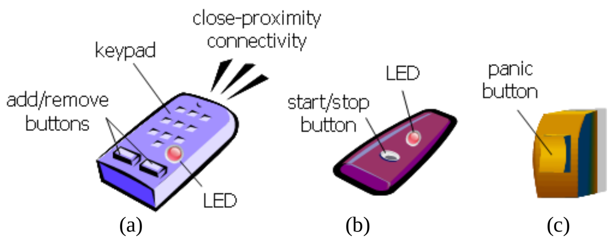


Figure 2. Tangible community artifacts: (a) key object, (b) application pill object, (c) panic button object.

Just like any other object, a pill must be added to the object community via the key before starting the application. The prototypical interface for the pill is (Figure 2b): (i) a push-button or switch to start/stop the application; and (ii) a LED for indicating the status of the application. One can imagine application pills being sold in stores and kiosks or given out for free bundled with products related to the application. At home, an application pill can be placed at location that allows for a casual periodic monitoring of its status LED.

Use case: Starting an application

Peter gets a home safety application pill object. He adds the pill to the community following the usual process. He then pushes the pill button to start the application. The LED on the pill blinks for several seconds and eventually turns green, indicating that the application is running.

Use case: Stopping an application

Peter wishes to stop the home safety application. He picks the application pill and depresses its button. The LED on the pill starts blinking. After a few seconds it turns off, indicating that the application has been stopped.

3) The panic button object: This object is used to forcefully terminate all applications running in the community at the push of a single button (Figure 2c). This could be required in case applications start behaving erratically or if the user feels uneasy about the overall platform behavior. The panic object can be likened to the master power switch in the electricity panel of a house or the reset button of a personal computer. As already mentioned, each application can be stopped by depressing the button of the respective pill. However, searching for and interacting with each individual pill can be quite stressful if the user is in a hurry. What's more important, stopping an application via the pill corresponds to a soft shutdown, under the control of the application program, which is clrealy undesirable when the application is malfunctioning.

Use case: Killing all applications

Maria notices an obscure object behavior without being able to infer what causes the problem. She quickly walks to the hallway and presses the button of the panic object attached on the main electricity panel. Soon, the weird behavior stops and the LEDs of all application pill objects turn off. Being more relaxed, Maria arranges for the technician to drop by the next day in order to get a closer look at the problem.

C. More convetional user interaction

A typical community will include several objects that do not have considerable user interface capabilities. In fact, objects like a window, a lamp or a motion detector do not have any proper user interface at all. On the other hand, objects like a TV or a digital frame can support (very) rich user interaction. Our approach is to rely on objects with advanced UI capabilities for configuring applications, while at the same time letting applications engage even simple to notify or alert the user. The key elements of our user interaction scheme are as follows.

1) Notifications & alerts: Applications may occasionally need to request the user's attention; this is achieved through notifications and alerts. The difference between the two is that notifications convey a verbal message whereas alerts do not carry such information (the user is responsible for finding out the cause of the alert). Notably, the actual form of notifications and alerts depends on the object that provides this function, each object supporting a different, perhaps complementary, flavor. Even simple objects like a lamp or a doorbell can contribute in this respect, especially for alerts, e.g., by blinking and respectively ringing at a certain alarming pattern. Of course, more complex objects with audiovisual capabilities, such as a TV or a radio, can be employed to make notifications via text or voice messages.

Use case: Being alerted/notified by an application

Peter's alarm clock in the bedroom and his wristwatch start beeping intensely (alerts). He walks into the living room and notices a message flashing on the TV screen warning him about a possible hazard with the stove (a notification).

2) Application setup: Once started via the pill, an application will ideally run without any interaction and rarely issue alerts or notifications. However, in the general case, some setup will be required, e.g., to change default thresholds or specify user preferences. For this purpose, each object that has a sufficiently powerful user interface is expected to allow the user to browse the list of applications running in the object community, and inspect or modify their settings. The setup process follows the native interaction style and look-and-feel of the object that provides the application setup function (consider differences between a remote-driven TV and a mobile phone with a touch-screen). Importantly, the setup can be accomplished with any UI-capable object, and the user can freely choose the one that suits him best.

Use case: Configuring an application

Peter decides to inspect the settings of the home safety application using the TV. He presses the "community function" button on the remote and browses the application list shown on the TV screen. He selects the home safety application, reviews its settings and decides to change the default policy for alerts. When the change is confirmed, Peter presses the "community function" button on the remote and the setup window disappears from the screen.

IV. IMPLEMENTATION

The presented platform concept and end-user model is currently being implemented in the POBICOS project [2]. Several ideas and features of POBICOS have their roots in ROVERS [3], which is a predecessor of this work. This section gives an overview of the POBICOS platform and briefly describes the system-level mechanisms used to achieve the end-user functionality described in the previous sections.

A. POBICOS platform overview

The POBICOS platform follows a middleware approach whereby each object supports a standard API. Objects may feature different middleware extensions depending on their sensing, actuating and computing capabilities. The application programming model is based on mobile code units, called micro-agents, which execute on top of a VM environment [4]. Each application typically consists of several cooperating micro-agents that spread in the community to exploit the capabilities of objects (Figure 3).

The POBICOS middleware is implemented on top of TinyOS v2 for the Imote2 from Crossbow using an eZ430-RF2480 ZigBee subsystem from Texas Instruments for the wireless communication between nodes. Regular objects are prototyped using Imotes. A generic adapter box with an Imote and a power level converter (Figure 4a) is used to POBICOS-enable objects and external systems via RS232.

B. *Adding and removing objects*

The key object, implemented on an Imote, maintains a registry with the addresses of all objects that are part of the community. The registry is updated when an object is added to or removed from the community. Registry updates can be propagated to the community in an asynchronous fashion by several objects (not just the key). To avoid inconsistencies, the key assigns to each update a monotonically increasing version number, enabling objects to detect duplicates and take into account membership changes in the right order.

The close-proximity communication between the key and the object being added/removed is implemented using the short-range mode of the 802.15.4 radio on the Imote (in principle, any near-field communication technology can be used for this purpose). When the add/remove button is pressed, the key establishes a connection with any object that is close-enough to respond, retrieves the object's address and performs the requested interaction (updating the registry as needed). Provided the range is small-enough, this guarantees that the proper object will be addressed but also that no other object can eavesdrop on the conversation.

Objects that are part of the same community encode and decode the messages exchanged between them over ZigBee using a community-wide encryption key. This is generated by the key object based on the name and PIN chosen by the user, and is transmitted to each object as part of the addition process. More details about the security approach and respective key and registry management protocols in POBICOS can be found in [5].

C. *Starting and stopping applications*

The application pill object is also implemented using an Imote. It contains the entire application code bundle, i.e., the binaries of all micro-agents of the application. The bundle is loaded on the Imote from a PC via the serial port. Pressing the application pill button leads to the instantiation of the micro-agents on the local or remote nodes (under the control of the application program). Depressing the button causes the micro-agents to be removed.

D. *Notification and alerts*

The POBICOS middleware features special instructions for notifying and alerting the user. Both types of instructions range from a high abstraction level such as "alert using whatever means possible" to more specific levels like "alert visually" or "alert by siren sound". Some objects support the notification and alert instructions in a manner that is compatible with their natural/native functionality. For example, in the current prototype an alarm can be raised by a variety of objects such as a TV (controlled via a POBICOS-enabled set-top-box; Figure 4c), a lamp or a beeper (both controlled via a POBICOS-enabled power plug; Figure 4b).

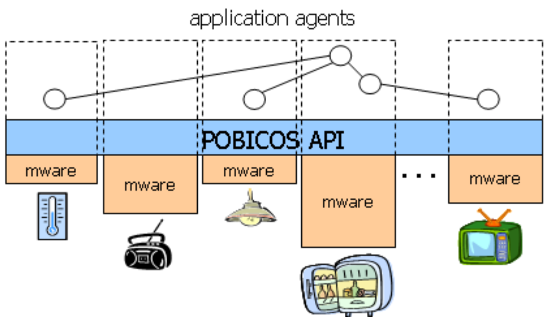


Figure 3. POBICOS platform concept: objects feature different middleware extensions based on their capabilities, application micro-agents are placed on available objects to exploit them.

When a micro-agent invokes an abstract instruction at runtime, it is mapped to a more specific instruction supported by its host. Thus, applications using abstract notification and alert instructions can exploit a wide range of objects, which may provide different specific instructions; this obviously comes at the price of having less control on the way the user will be actually notified/alerted. It is up to the programmer to decide what the meaningful tradeoff is for each occasion.

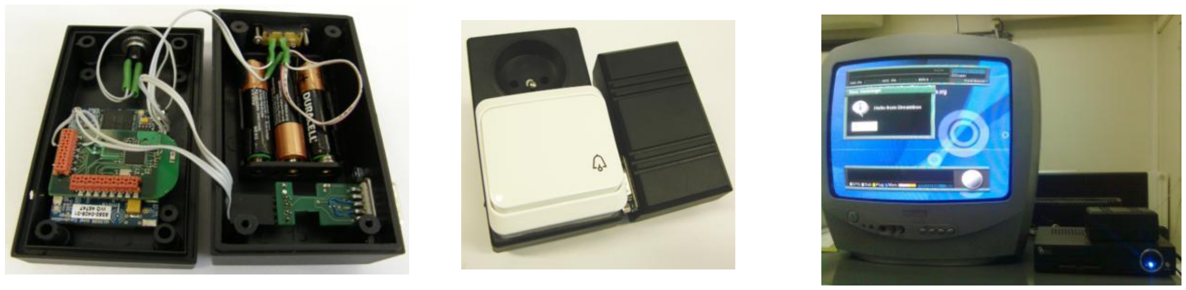
Last but not least, the POBICOS middleware provides a primitive for instantiating multiple copies of micro-agent on as many objects support the instruction(s) invoked by it. This allows an application to engage several objects at once for the purpose of alerts and notifications, thereby increasing the probability of catching user attention. It is important to note that this does not require any additional effort on behalf of the programmer.

E. *Application setup*

The setup functionality is implemented based on (i) a distributed protocol for fetching/updating the configuration settings of all currently running applications, and (ii) a user interface front-end for browsing and changing these settings. The first component is part of the middleware core running on the Imote. The second component is optional and needs to be developed separately for each object, depending on its UI capabilities. At this point, a front-end is available for the PC which communicates with the first middleware component on the Imote via the serial port. Proper UI front-ends are under development for a TV set-top-box and a mobile phone.

V. RELATED WORK

In our model the user defines the operational and security scope of the community by adding and removing objects via the key object in a conscious and explicit way. The issue of knowing which devices belong to the same system scope also arises in mobile ad-hoc systems that allow wearable and portable devices to dynamically participate in a personal area network, e.g., the 2WEAR system [6] and the Spartan BodyNet [7]. These systems assume that devices have been a priori assigned a unique id indicating their owner and already



(a) (b) (c)

Figure 4. POBICOS-enabling real objects: (a) the RS232 adapter; (b) the power socket; (c) the TV set-top-box.

hold security keys that can be used to encrypt data and to perform a challenge-response scheme. In general, ad-hoc wireless technologies provide network-level association mechanisms based on a shared secret but do not specify how a device ends-up with this information. ZigBee implements its own security scheme but the transmission of keys from the coordinator to a new device that joins the network occurs via an ordinary open message exchange over wireless.

Significant research has been done on many aspects of user interaction in smart spaces/environments, e.g., [8] [9]. Of particular importance are alternative methods of input and control, e.g., see [10] for controlling devices via hand-based gestures, or [11] for supporting voice-based interaction with appliances. Our end-user model does rely on advanced UI. In fact, it is designed to exploit regular objects that are likely to be part of a household anyway, via modes and modalities the user is already familiar with. Moreover, it does not focus only on UI-capable objects but allows even simple objects to be engaged for notifying/alerting the user.

The vision of ubiquitous computing [1] is for the system to provide the desired functionality without distracting the user. Our model is conceived along these lines, requiring user intervention only for application configuration, which happens under user control; the user decides when to start such an interaction and is free to pick any UI-capable device for this purpose. Notification and alarms are introduced as first-class aspects of domestic computing since they play a key role in raising user awareness.

Tangible interfaces and the importance of having special objects dedicated to special functions have received a lot of attention in the HCI domain, see [12] for an overview. In the spirit of the community key proposed in our model, [7] discusses the use of a lock-shaped object to enable privileged functionality in a wearable system, while [13] proposes a wristwatch as an authentication device for ubiquitous service access. Also, the concept of the application pill has some similarities with the 2WEAR application wallet [6] and the personal server [14]. The former carries the code/state of applications that exploit I/O peripherals found in the personal area network. The latter serves as a personal data drive that can connect to applications running on nearby PCs to access/process this data. The main difference is that each pill is dedicated to a single application and features a minimal UI for controlling and monitoring its execution hence the pill is in fact a tangible representation of the application itself.

ACKNOWLEDGMENT

This work is funded by the 7th Framework Program of the European Community, project POBICOS, FP7-ICT-223984.

REFERENCES

[1] M. Weiser, “The Computer of the 21st Century”, in Scientific American, 256(3), 1991, pp. 78-89.

[2] POBICOS project web site. http://www.ict-pobicos.eu/

[3] J. Domaszewicz , M. Roj, A. Pruszkowski, M. Golanski, and K. Kacperski, “ROVERS: Pervasive Computing Platform for Heterogeneous Sensor-Actuator Networks”, Proc. WoWMoM 2006, pp. 615-620.

[4] A. Pruszkowski, T. Paczesny, and J. Domaszewicz, “From C to VM-targeted Executables: Techniques for Heterogeneous Sensor/Actuator Networks”, Proc. WISES 2010, in press.

[5] P. Tarvainen, M. Ala-Louko, M. Jaakola, I. Uusitalo, S. Lalis, T. Paczesny, M. Taumberger, and P. Savolainen, “Towards a Lightweight Security Solution for User-Friendly Management of Distributed Sensor Networks”, Proc. ruSMART 2009, pp. 97-109.

[6] S. Lalis, A. Savidis, A. Karypidis, J. Gutknecht, and C. Stephanides, “Towards Dynamic and Cooperative Multi-Device Personal Computing”, in The Disappearing Computer, LNCS 4500, 2007, Springer, pp. 182-204.

[7] K. Fishkin, K. Partridge, and S. Chatterjee, “Wireless User Interface Components for Personal Area Networks”, in IEEE Pervasive Computing, 1(4), 2002, pp. 49-55.

[8] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, “EasyLiving: Technologies for Intelligent Environments”, Proc. HUC 2000, pp. 97-119.

[9] B. Johanson, A. Fox, and T. Winograd, “The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms”, in IEEE Pervasive Computing, 1(2), 2002, pp. 67-74.

[10] ASM. Rahman, M. Hossain, J. Parra, and A. El Saddik, “Motion-path based Gesture Interaction with Smart Home Services”, Proc. ACM MM 2009, pp. 761-764.

[11] T. Kostoulas, I. Mporas, T. Ganchev, N. Katsaounos, A. Lazaridis, S. Ntalampiras, and N. Fakotakis, “LOGOS: A Multimodal Dialogue System for Controlling Smart Appliances”, in New Directions in Intelligent Interactive Multimedia, SCI 142, 2008, Springer, pp. 585-594.

[12] K. Fishkin, “A Taxonomy for and Analysis of Tangible Interfaces”, in Personal and Ubiquitous Computing, 8(5), 2004, pp. 347-358.

[13] J. Al-Muhtadi, D. Mickunas, and R. Campbell, “Wearable Security Services”, Proc. ICDCS Workshops 2001, pp. 266-271.

[14] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light, “The Personal Server: Changing the Way we Think about Ubiquitous Computing”, Proc. Ubicomp 2002, pp. 194 - 209.