Engineering Intuitive and Self-Explanatory Smart Products

Erwin Aitenbichler, Fernando Lyardet, Gerhard Austaller,
Jussi Kangasharju, Max Mühlhäuser
Telecooperation Group, Department of Computer Science
Darmstadt University of Technology, Germany
{erwin,fernando,gerhard,jussi,max}@tk.informatik.tu-darmstadt.de

ABSTRACT

One of the main challenges in ubiquitous computing is making users interact with computing appliances in an easy and natural manner. In this paper we discuss how to turn ordinary devices into *Smart Products* that are more intuitive to use and are self-explanatory. We present a general architecture and a distributed runtime environment for building such Smart Products and discuss a number of user interaction issues. As an example, we describe our smart coffee machine and its validation through systematic user testing.

Categories and Subject Descriptors

C.3.h [**Computer Systems**]: Ubiquitous Computing; D.2.11 [**Software Engineering**]: Architectures; H.5.2 [**Information Systems**]: User Interfaces

General Terms

Smart Products

Keywords

Smart Products, Ubiquitous Computing, User-centered design

1. INTRODUCTION

The vision of ubiquitous computing, as stated by Mark Weiser, is that the computers become so commonplace and so interwoven with our environment, that they practically disappear and become invisible [15]. One of the main challenges in building ubiquitous computing applications is how can the user interact with the invisible computer. The interaction should be easy and natural, yet allow for a sufficiently rich set of actions to be taken.

A further challenge arises when the systems should be usable by end users, that is, people who are not specialists in interaction research. Such people, on the other hand, expect easy and natural interaction and are not willing to take additional steps to perform seemingly simple actions.

SAC'07 March 11-15, 2007, Seoul, Korea

In this paper, we explore the challenge of building useful *and* usable everyday applications, destined to be usable by everyone. We present a platform that supports building such smart products and as a practical application of our research, we describe a smart coffee machine.

2. RELATED WORK

Research in smart appliances and environments has emphasized two major issues lately: i) activity detection, mainly used as context for system adaptation and user interaction [7]; ii) generic sentient computing infrastructures that collect and provide information [2]. Many previously reported systems and infrastructures are based on the instrumentation of consumer-oriented locations such as offices [3], homes [4], and bathrooms [5]; they aim at detecting the activities of occupants [4, 10] and the interplay between users and mobile devices [12].

Several projects have explored the computerization of everyday objects, investigating the granularity of information and the type of information that can be obtained. One approach is the pervasive embedding of computing capabilities as in Things That Think [14], the MediaCup [9], or the InfoDoors [13], where simple objects gather and transmit sensor data; the distributed infrastructure derives location and other context information from these data for use by applications. Another aspect that differentiate Smart Objects is the *embedded knowledge* they carry. This property has also been explored in both business and consumer scenarios. A business example is asset tracking [8]: enhanced sensing and perception provides for, e.g., autonomous monitoring of the physical integrity of goods or detection of hazardous physical proximity [6]. On the consumer side, for instance, hard drives that can check their own health and predict when they will fail, personal training assistants that adjust the pace according to the user's pulse, or smartphones that can be voice controlled with expandable PDA features. Another interesting example is the ABS system commonly available in cars that integrates data from various sensors to help the driver apply the breaks more effectively. The ABS combines its knowledge about driving conditions and reacts upon user's pedal feedback, adjusting the actual breaks behavior to void skids. The user does not have to ever learn about all the intricate relationships between the sensors and their subsystems in order to take advantage of this advanced functionality: just press a pedal.

Mr. Java, a project at MIT [11], has the closest relation to the coffee machine example reported here, at least at first sight: both investigate household appliances with

connectivity and enhanced information processing capabilities, and both integrate user identification and behavior customization. The following three characteristics distinguish our project from Mr. Java and the other references cited above:

1. In contrast to the usual design - evaluate - publish approach, we iterated over many design-evaluation cycles; thereby, we leveraged the adaptivity of our smart products platform, which we consider an outstanding feature; we concluded that any particular social environment needs careful customization in order to reach user satisfaction.

2. In addition to supporting simple event-action rules, we emphasize complex tasks and procedures; we consider it a major benefit if users can be guided through such procedures as opposed to reading detested manuals - in addition, this benefit helps vendors to leverage feature-rich appliances; in the past, users often did not invest the necessary effort to explore advanced features.

3. From our experiences, we distilled a general architectural approach for turning an everyday item into a ubiquitous appliance, in such a manner that average, non-technical users find the device easy and natural to use. With the three distinct characteristics above, the present paper clearly differentiates from the yet another ubiquitous appliance category.

3. SMART PRODUCTS ARCHITECTURE

Smart Products are real-world objects, devices or software services bundled with knowledge about themselves, others, and their embedding. This knowledge has been separated in layers according to the level of abstraction they address: device capabilities, functionality, integrity, user services, and connectivity. In Figure 1 we present a conceptual reference architecture showing this separation of concerns to allow the integration of different vendors providing their own technology. Such scenario is particularly critical at the device level, since changes to embedded systems must be kept to a minimum to keep their cost viable. Adopting a SOA approach allows devices to be extended in their functionality and user adaptation capabilities with minimal embedded infrastructure requirements. Extensions to a Smart Product may involve external hardware, software, or both.

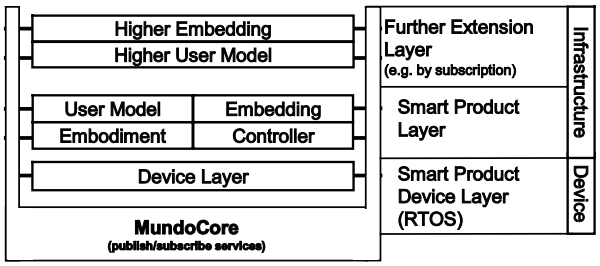

Figure 1: Smart Product Conceptual Architecture

The first layer is the Smart Product Device Layer. In embedded systems, this is where the runtime operating software or firmware lies. The processing power available at this level operates the actuators, sensors, I/O, and the user interface (typically LCD displays, status LEDs and buttons). The knowledge embedded in this layer defines a set of valid events, states, and a set of Event-Condition-Action (ECA) rules that govern the transitions between states. These rules determine the functionality of the device, implement current *smart behavior*, and ensure the operating conditions required to preserve the hardware integrity and safe operation.

The second layer is the Smart Product Layer, which consists of four main parts. First, the *Controller* that sometimes resides within the embedded device, coordinates processes bridging the physical-world behavior and software-based functionality. Second, the *Embodiment* that externalizes the knowledge of what a device consists of, what are its parts and their interrelationship. This knowledge also specifies the processes controlling the functionality. For instance, the process of making a coffee, or more complex, de-scaling a coffee machine. The *Embedding* bridges the physical world with software, enabling self-explanatory functionality. The *Embedding*, in a sense, *decorates* the processes described by the embodiment for adaptation using the information provided by the fourth constituent part: the *User Model*. The *User Model* provides information regarding the user that operates the device. Using other information sources and sensors from the infrastructure, the *User Model* module can recognize a person and related data such as preferences. When no external contextual information about the user is available, this module gathers the user input and compares it to user expertise patterns to better match user level to device functionality and assistance. The *Embedding* gathers the information from the *User Model* and determines the actual steps and explanations required for a user to perform a task.

Finally, other services can further extend a smart product functionality, for instance by subscription to the Smart Product manufacturer or third-party providers.

4. RUNTIME ENVIRONMENT

We will now describe the software architecture of our smart products runtime system (Figure 2).

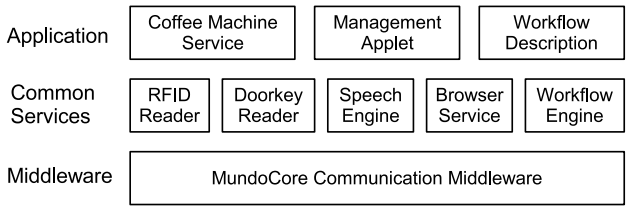

Figure 2: Software Architecture

The common basis for our SOA architecture is our own ubiquitous computing middleware MundoCore [1]. MundoCore has a microkernel design, supports dynamic reconfiguration, and provides a common set of APIs for different programming languages (Java, C++, Python) on a wide range of different devices. The middleware implements a peer-to-peer publish/subscribe-system and an Object Request Broker which allows us to easily decouple services and spread them on several different devices transparently. Many of the services are of a general nature and they have also been used in other projects. For example, the RFID reader service is a Java-based service that interfaces with a reader device connected to a computer's USB port and emits an event each time an RFID tag is moved close to the reader. The Speech Engine service is written in C++ using Microsoft's Speech SDK and AT&T NaturalVoices. The Browser Service allows

to remote-control an Internet browser by means of remote method calls (RMCs).

Finally, our smart coffee machine example presented here requires some application-specific services. The Coffee Machine Service provides the hardware abstraction for the coffee machine. It allows to control the machine by means of RMCs and generates event notifications when the status of the machine changes. The high-level application logic is implemented as a workflow description. This allows us to design the user interactions in a model-driven way. The use of workflows and how the workflow engine is integrated with other services in the system is described in Section 7.

5. THE SMART COFFEE MACHINE

The Saeco coffee maker is a normal off-the-shelf coffee machine. When used out of the box, the user can choose between three kinds of coffee namely espresso, small coffee, and large coffee. There is a coffee bean container and a water tank attached to the machine. If either is empty, a small display on the machine prompts to refill them. The display also prompts to empty the coffee grounds container if it is full.

Our modifications allow us to control all the buttons remotely and determine the state of the machine (Figure 3). With this state information and additional RFID readers, we can detect user actions and automatically start processes or proceed in a workflow.

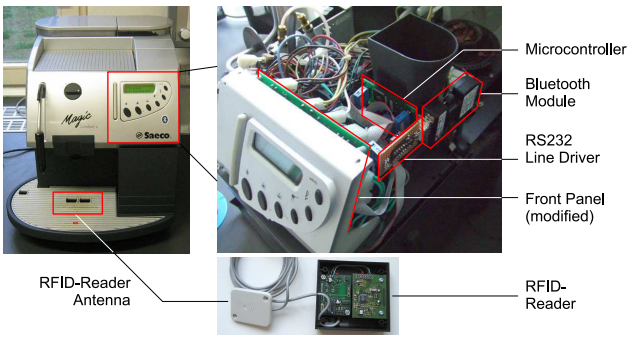

Figure 3: Hardware components added to the coffee machine

The hardware of the enhanced system consists of the coffee machine, two RFID readers to identify cups, one reader for reading the digital keys, and a PC running the control software. Figure 4 shows the hardware architecture and the individual components. The roles of the individual components are as follows.

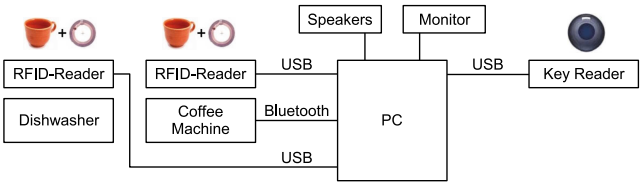

Figure 4: Hardware architecture

Coffee Machine: Because the machine does not come with a data interface, we modified the front panel circuit board and attached our own microcontroller to it. This allows us to detect keypresses, simulate keypresses, check if the water tank is empty and read the pump control signal. The latter indicates that the machine is actually dispensing fluid and gives a very accurate measure how much fluid has run through the pump; one pulse on this signal corresponds to 0.4 milliliters. The machine communicates with the rest of the system via a Bluetooth module.

RFID reader at Coffee Machine: The antenna of this reader is attached to the bottom of the coffee machine's cup holder surface. It is positioned such that the reader is able to identify cups as soon as they are placed below the coffee dispensing unit. The RFID tags are glued to the bottom of the cups. This allows the system to start brewing coffee as soon as a cup is put down.

RFID reader at Dishwasher: The second RFID reader is placed next to the dishwasher. User can swipe cups over this reader before they put them into the dishwasher.

Key reader: Our reader for the digital keys consists of a SimonsVoss Smart Relais, a microcontroller, and a USB interface. Like the electronic locks in our computer science building, the relais can be triggered with the digital keys, which are given to all employees and students. Every user owns a key, and there is a one-to-one relationship between users and keys, which makes these keys highly suitable for identification purposes. In addition, the keys can already serve as simple interaction devices. Because users are required to activate them explicitly by pressing a button, the reception of a key ID can be directly used to trigger actions.

PC: The PC hosts most of the services and is hidden in a cupboard. It gives users voice feedback via the speakers. Users can view their favorite webpages on the monitor. The monitor, keyboard, and mouse are optional and not required to use the core functions of the system.

6. INTERACTION DESIGN

In the interaction design of a smart product, we distinguish between simple and complex interactions and support them in two different ways.

Simple interactions are one-step functions that would be normally triggered with a button. These interactions should be natural to users. This should be even true if they use a product for the first time. The product should behave as the user intuitively expects. Such interactions are implemented by monitoring the user's actions with suitable sensors and typically do not involve graphical or voice user interfaces. An example for a simple interaction is that the user puts her coffee mug under the coffee dispenser and automatically gets her favorite coffee.

Complex interactions refer to multi-step procedures that require the user to have product-specific knowledge. Such interactions involve graphical or voice-based user interfaces and the system guides the user during these interactions. An example is de-scaling the coffee machine, which requires the user to perform several manual steps.

As many interactions as possible should be designed as simple interactions. To identify actions that are natural to users and to verify that designed interactions are intuitive, user studies must be performed at multiple stages in the product design. An important goal of our smart coffee machine project was to evaluate how average users (i.e., non-technical people) could interact with ubiquitous applications and appliances in a natural and intuitive way. The

implementation has gone through several phases and in each phase, we performed a user study in order to evaluate what things needed to be changed and how did the users feel about using the machine. In the following, we will present our experiences to illustrate the lessons learned from our user tests. Starting from the initial versions developed over 9 months ago, the machine has been in daily use in our group. Our group consists of 21 people. 5 people do not drink coffee at all and were excluded from the test. The 5 authors only participated in the first user test, therefore 11 users served as main test subjects.

6.1 Initial Implementation

In the initial design we marked coffee cups with RFID tags and when a user puts a coffee cup under the dispenser, she would automatically get coffee. We have cups in many different sizes, ranging from small espresso cups to large coffee mugs. The cup database stores the RFID tag IDs together with the type of coffee (espresso, small, or large) and the cup size in milliliters.

After interviewing the users to get their subjective impressions, we found out that users preferred the smart machine, because it requires no user attention to trigger the machine or to control the amount of coffee you want. This "un-attended nature" of the machine was cited as a benefit by all users.

Since the coffee machine is controlled by a service, the service also can log statistical data. In particular, the service knows how much coffee beans are left and for how long they will last. We programmed a notification service to send emails to all the people using the kitchen when beans were *close* to running out.

6.2 Error Conditions

In case of a problem such as running out of water, in addition to the standard error messages on the machine's display, the user also receives audio instructions on how to resolve the problem (i.e., "User X, please refill water."). After the user has fixed the problem, the machine automatically resumes the requested operation. All feedback from the system is also over voice, delivered through a Text-to-Speech engine.

With audio feedback, users changed their behavior when handling error situations with the machine. The most common error situations are a full coffee grounds container and the machine running out of water. We observed that in case of lack of audio feedback, the users would complain that the system was "broken", instead of looking for the problem themselves on the display as they did before. The machine behaved exactly as before (i.e., it showed the error message), but users did no longer read the error messages from the display of the machine. We concluded that this was the result first of the expected consistency in the machine's behavior, and second, to the fact that although listening is slower than reading, it requires less effort, what could make it a more appealing modality for consumers.

6.3 Billing

Next, we wanted to add automatic billing to the system. This requires to identify users by some means. The first implementation was based on everyday observations on how people handle the coffee machine and also by asking them how they would like to operate it. Because most users have their favorite coffee cup and mostly drink one kind of coffee, our conclusion was that associating a cup to a user is an administrative task that has to be done rarely and therefore does not have to be very comfortable.

Based on this hypothesis, the initial implementation had the following characteristics: Changing settings and associating cups to a user had to be done with a GUI application running on a nearby computer. In addition, cups got leases. This means that after 24 hours of not using the cup, the association was deleted and the cup got freed. In other words, you would have to associate the cup typically once a week.

Once a cup is associated, the user automatically gets her preferred coffee when she puts the associated cup under the coffee dispenser. In addition, the system automatically takes care of accounting and the user can have his favorite web site pop up on a nearby computer screen.

However, as we observed the initial users, we noticed that some of our assumptions were wrong, and that users even started to change their behavior in order to overcome some limitations and problems.

Firstly, it turned out that human beings are not creatures of habit but creatures of laziness. Instead of washing their cups and reusing them, they put them into the dishwasher and use clean cups. This conflicted with the assumption that associating cups once a week is acceptable. Associations had to be done much more often and users found it unacceptable to use a GUI application. Although the GUI was enhanced several times, it was simply not accepted.

Secondly, the introduction of a "date of expiry" for cups was not convincing to the users. They wanted to be able to explicitly de-associate the cup and get feedback that the cup was free.

We addressed the concerns raised in the first test by making the following modifications: To improve the association process, we introduced a "one-click association". Here, we use our digital door keys to associate cups to users. Because users were used to use the door key several times a day, there was no need to teach the users how to use the digital key. They just had to be told to press the key when asked by the coffee machine, as if they were opening a door.

The explicit de-association was handled by mounting a second RFID reader near the dishwasher. When people put their cups in the dishwasher, they can swipe the cup over the reader to break the association. More importantly, they get voice feedback that the association has been broken.

With this implementation, almost all system interactions could be done with tagged cups, the digital key and Text-to-Speech instead of using the GUI-based application.

6.4 User Tests

After the modifications, we performed two formal user tests, conducted by observing users using the coffee machine as well as free-form interviews. The overall outcome was extremely encouraging. Most users preferred to use the automated machine in spite of the procedure being slightly more complicated and the time to get coffee slightly longer. (Note that it takes 15–25 seconds to get the coffee, depending on the type of coffee, so an additional delay on the order of a second or two is usually not significant.)

The second set of tests focused on the other aspects of the coffee machine operation such as voice feedback, the association, selection of an alternate coffee, and the subjective user perception. The use of voice to provide feedback was highly

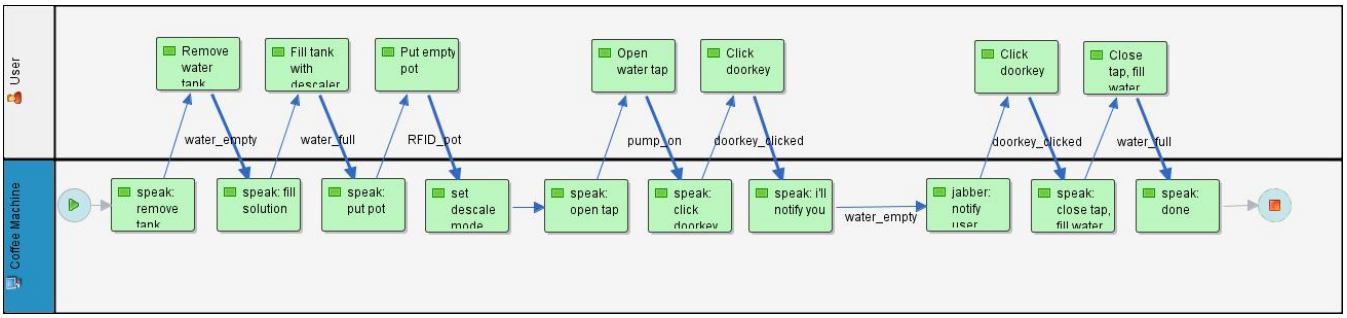

Figure 5: De-scaling process modeled in XPDL using XPEd

regarded as beneficial. However, many users (2/3) found it difficult to understand the voice output in some cases. We believe the reasons might be as follows: When a user hears a particular message for the first time, she is probably surprised by it and does not fully understand the message, hence leaving the impression that the messages are hard to understand. Sometimes the voice feedback comes when the machine is busy performing the requested action, and the machine is typically quite loud, hence the voice feedback is hard to understand. To alleviate this problem we implemented "subtitles". All voice output is displayed as text on the computer screen as well.

Another issue that arose was that users started to wonder how to make a different coffee than the default one without changing the default user settings. This situation happens when a user wants, for example, an espresso instead of the usual large coffee. Again, being able to change the setting with only a GUI was not acceptable. To cater for people wanting different coffees at different times, we implemented an "override button". Fortunately, there was an extra button on the coffee machine which did not have any function and we used it as the override button. When a user presses this button, she can choose the kind of coffee she wants instead of getting the coffee programmed for the cup.

6.5 Results

First and foremost, our experience underlines the importance of user-centered design. Ubiquitous applications often enter new and unexplored domains, hence it is hard or impossible to know beforehand how a system can best be used. Not many features from our initial design survived through the two user tests, even though the design seemed sound at that time. In particular, using the digital key as user identification was a positive experience. The lesson here would be that if users need to perform additional actions, they are more readily accepted if they are based on other items which they use in everyday life, even if the uses are different.

Second, only certain modalities may be mixed in a system with a multi-modal user interface. In our case, this was shown by users not checking the display of the machine. We solved this by making all the output from the system to come over audio. However, the users did not have problems with different modalities on the input and output channels. "Tactile" input and auditory output was not a problem to the users.

Third, users do not trust automatism, at least in every case. Even though a lease mechanism was implemented (and thoroughly debugged!), users insisted on having an explicit possibility to de-associate cups.

7. COMPLEX INTERACTIONS

In the following, we describe how our extensions to the machine can be used to help guide the user through complex procedures, in our case the cleaning (or de-scaling) of the machine. The machine needs to be cleaned regularly, since limestone builds up in the machine and it has to be removed. The process is relatively cumbersome and complicated, since it involves first filling the machine with the cleaning liquid, running the liquid through the machine using a certain mechanism that includes pushing the right buttons and opening and closing the water tube at the right time, waiting a relatively long time, and then finally flushing the machine before it is ready to be used again. Most people in our group are not aware how to do this process.

Our goal in implementing this task support is to show how our modifications can help users in complex tasks. We chose cleaning as the task, because it must be performed occasionally even though the process is described in the manual, most people never read the manual and even if they do read it, they must use the manual as a support every time they have to clean the machine. The cleaning process is also very suitable for being supported, since parts of it can be done automatically (controlled by the system) and parts of the process require manual user interaction. Furthermore, there are long waiting periods in the process. Our implementation shows how the system supports the user by letting the user know what to do next at any step. Any steps which can be performed automatically are done by the system and only when user intervention is required, does the machine ask the user for help. Also, when a particular step of the process will take a long time, the machine lets the user know this, so that the user does not need to wait by the machine. We have implemented a notification system which alerts the user via an instant message when the long step has been completed.

Internally, we describe this process as a workflow description (Figure 5). We use the XML Process Definition Language (XPDL) as data format, the JPEd graphical editor to edit workflows, and the OpenEmcee Microflow Engine to execute workflows. We have written a small Perl script to translate XPDL descriptions into OpenEmcee's proprietary XML format.

In the described system, all communications between services is based on MundoCore and uses channel-based publish/subscribe. We defined a generic activity and transition class for the workflow engine that interfaces with this publish/subscribe system.

Following our reference architecture, the workflow descriptions become part of the Embodiment Module and are executed by the Controller. In this particular example, the Embedding Module performs only a simple adaptation of the process by stating the user's name provided by the User Model.

7.1 Activities

XPDL permits to assign an arbitrary number of *extended attributes* to *activities* and *transitions* in the workflow. The attributes of an activity are used to describe the action that should take place. An action can be a message send operation or a remote method call. Method calls build on the dynamic invocation interface of MundoCore which allows to call any method of any remote service. For example, to output text via the Text-to-Speech engine, the following attributes are used:

```
channel   = "tts"
interface = "org.mundo.speech.synthesis.ITextToSpeech"
method    = "speak"
p0        = "Please remove water tank"
```

The channel property specifies the name of the channel to which the invocation request should be sent. The TTS service is subscribed to the channel `tts` either locally or somewhere in the network. The message distribution is handled by MundoCore and is fully transparent to the application. (It should be noted that channels are not necessarily global - MundoCore has *zone* and *group* concepts to limit the scope of channels.)

7.2 Transitions

State transitions can be triggered by arbitrary MundoCore events. For example, if the water tank becomes empty, the Coffee Machine service publishes a notification of type `org.mundo.service.saeco.WaterEvent` with the content `empty = true` to the channel `saeco.event`.

MundoCore supports notification filtering based on XQuery expressions. We use this mechanism to describe transition conditions in the workflow. To execute a transition as soon as the water tank is empty, the following extended attributes are specified for the transition:

```
channel = "saeco.event"
filter  = "for $o in $msg where
            $o[class='org.mundo.service.saeco.WaterEvent'] and
            $o/empty=true()"
```

The transition is executed as soon as the first notification matches this filter expression.

8. CONCLUSION

In this paper we have presented a general architecture and distributed runtime environment for building Smart Products. Turning a device into a Smart Product only requires minimal physical extensions to the device itself - basically a simple communication interface, such as Bluetooth, to retrieve and set the device's events and state. Through this communication interface, the surrounding ubicomp environment is able to integrate the device, improving its functionality and usability.

Using a coffee machine as an example, we have shown how to extend it into a Smart Product with additional functionality and adaptation capabilities using the proposed architecture. A major concern during this work has been the impact on the people using the device. The different user tests carried out provided valuable feedback and insights to keep interaction simple and natural. The tests also pointed out the requirement of a system support for guiding users through complex procedures. This functionality has been also developed and integrated in our reference architecture.

9. REFERENCES

[1] E. Aitenbichler. System Support for Ubiquitous Computing. Shaker, 2006.

[2] E. Aarts. Ambient Intelligence: A Multimedia Perspective. *IEEE Multimedia*, 11(1):12–19, Jan 2004.

[3] M. Addlesee, R. Curwen, S. Hodges, et al. Implementing a Sentient Computing System. *IEEE Computer*, 34(5):50–56, Aug 2001.

[4] B. Brumitt, B. Meyers, J. Krumm, et al. EasyLiving: Technologies for Intelligent Environments. In *Proc. of HUC 2000*, volume 1927 of *LNCS*, 12–27. Springer, September 2000.

[5] J. Chen, A. H. Kam, J. Zhang, et al. Bathroom Activity Monitoring Based on Sound. In *Proc. of Pervasive 2005*, volume 3468 of *LNCS*, May 2005.

[6] C. Decker, M. Beigl, A. Krohn, et al. eSeal - A System for Enhanced Electronic Assertion of Authenticity and Integrity. In *Proc. of Pervasive 2004*, volume 3001 of *LNCS*, 18–32. Springer, April 2004.

[7] A.K. Dey, D. Salber, and G.D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *HCI Journal*, 97–166, 2001.

[8] A. Fano and A. Gershman. The Future of Business Services in the Age of Ubiquitous Computing. *Comm of the ACM*, 45(12):83–87, December 2002.

[9] H.-W. Gellersen, M. Beigl, and H. Krull. The MediaCup: Awareness Technology embedded in an Everyday Object. In Proc. of HUC'99, volume 1707 of *LNCS*, 308–310, 1999.

[10] MIT Project Oxygen. http://oxygen.lcs.mit.edu/.

[11] Mr. Java Project. http://www.media.mit.edu/ci/projects/mrjava.html

[12] A. Schmidt, K. A. Aidoo, A. Takaluoma, et al. Advanced Interaction in Context. In *Proc. of HUC'99*, volume 1707 of *LNCS*, 12–27. Springer, September 1999.

[13] B. Shneiderman. *Leonardo's Laptop: Human Needs and the New Computing Technologies*. MIT Press, October 2002.

[14] Things That Think. http://ttt.media.mit.edu/.

[15] M. Weiser. The Computer for the 21^{st} Century. *Scientific American*, 265:66–75, 1991.