CEPIS UPGRADE is the European Journal for the Informatics Professional, published bi-monthly at <http://cepis.org/upgrade>

Publisher
CEPIS UPGRADE is published by CEPIS (Council of European Professional Informatics Societies, <http://www.cepis.org/>), in cooperation with the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática,* <http://www.ati.es/>) and its journal Novática

CEPIS UPGRADE monographs are published jointly with Novática, that published them in Spanish (full version printed; summary, abstracts and some articles online

CEPIS UPGRADE was created in October 2000 by CEPIS and was first published by Novática and INFORMATIK/INFORMATIQUE, bimonthly journal of SVI/FSI (Swiss Federation of Professional Informatics Societies, <http://www.svifsi.ch/>)

CEPIS UPGRADE is the anchor point for UPENET (UPGRADE European NETwork), the network of CEPIS member societies' publications, that currently includes the following ones:
- inforewiew, magazine from the Serbian CEPIS society JISA
- Informatica, journal from the Slovenian CEPIS society SDI
- Informatik-Spektrum, journal published by Springer Verlag on behalf of the CEPIS societies GI, Germany, and SI, Switzerland
- ITNOW, magazine published by Oxford University Press on behalf of the British CEPIS society BCS
- Mondo Digitale, digital journal from the Italian CEPIS society AICA
- Novática, journal from the Spanish CEPIS society ATI
- OCG Journal, journal from the Austrian CEPIS society OCG
- Pliroforiki, journal from the Cyprus CEPIS society CCS
- Tölvumál, journal from the Icelandic CEPIS society ISIP

Editorial correspondence: Llorenç Pagés-Casas <pages@ati.es>
Advertising correspondence: <info@cepis.org>

Subscriptions
If you wish to subscribe to CEPIS UPGRADE please send an email to info@cepis.org with 'Subscribe to UPGRADE' as the subject of the email or follow the link 'Subscribe to UPGRADE' at <http://www.cepis.org/upgrade>

ISSN 1684-5285

Monograph of next issue (April 2011)
"Software Engineering for e-Learning Projects"
(The full schedule of **CEPIS UP**GRADE is available at our website)

CEPIS UPGRADE
The European Journal for the Informatics Professional
http://cepis.org/upgrade
Vol. XII, issue No. 1, February 2011

Monograph
Internet of Things
(published jointly with Novática*)
Guest Editors: *Germán Montoro-Manrique, Pablo Haya-Coll, and Dirk Schnelle-Walka*

UPENET (UPGRADE European NETwork)

CEPIS NEWS

* This monograph will be also published in Spanish (full version printed; summary, abstracts, and some articles online) by **Novática**, journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática*) at <http://www.ati.es/novatica/>.

The Mundo Method — An Enhanced Bottom-Up Approach for Engineering Ubiquitous Computing Systems

Daniel Schreiber, Erwin Aitenbichler, Marcus Ständer, Melanie Hartman, Syed Zahid Ali, and Max Mühlhäuser

Deploying ubiquitous computing systems into real world scenarios can realistically only be done in a bottom-up way, using smart building blocks. Our everyday environments are just too chaotic to allow top-down design of ubiquitous computing systems. Creating ubiquitous computing systems in a bottom-up manner has some inherent problems, which have not been successfully addressed in any existing approach, hence the scarcity of real-world ubiquitous computing systems. This article describes the Mundo Method for designing and implementing ubiquitous computing systems, which addresses two of these problems: structuring, i.e., separating the spontaneously emerging system into meaningful ensembles and orchestration, i.e., providing meaningful behaviour for an ensemble. The Mundo Method heavily relies on the MundoCore communication middleware, which is especially suited for ubiquitous computing applications.

Keywords: Bottom-Up Development, Methodology, Middleware, Smart Products, Ubiquitous Computing.

1 Introduction

There is an abundance of ubiquitous computing system infrastructures described in literature (see [1] for a survey). With them comes an almost equally large number of approaches for developing ubiquitous computing systems. Within these, we distinguish two main paradigms, which we call *top-down* and *bottom-up*.

- The **top-down** paradigm is characterized by the assumption that the complete system is under control of the engineer and that all components are known when a system implementation is designed.
- The **bottom-up** paradigm assumes that instead of being designed by an engineer as a whole, ubiquitous computing systems result from spontaneous integration of independent components.

To illustrate the difference between these two paradigms, we consider the example scenario of a kitchen that should be equipped with a ubiquitous computing system, i.e., the kitchen should become *smart*, similar to the ones described in [2][3][4]).

Following the top-down paradigm, the system design would start by careful observation of user needs, thereby eliciting requirements for the system behaviour. The implementation is then designed assuming complete knowledge over all aspects of the room, and assuming control

Authors

Daniel Schreiber works as a Doctoral Researcher at the Telecooperation lab *Technische Universität Darmstadt*, Germany. Currently he is working on user interfaces for ubiquitous computing systems in the Smart-Products EU project. <schreiber@tk.informatik.tu-darmstadt.de>.

Erwin Aitenbichler received his MSc in Computer Science from Johannes Kepler University in Linz, Austria, and his PhD in Computer Science from *Technische Universität Darmstadt*, Germany. Currently he is a Post-doctoral Researcher in the Telecooperation Lab in Darmstadt. His research interests are Smart Environments and Ubiquitous Computing. Erwin is a member of the ACM. <erwin@tk.informatik.tu-darmstadt.de>.

Marcus Ständer holds a master degree in Computer Science and works as a Research Associate in the Telecooperation Lab at the *Technische Universität Darmstadt*, Germany. His research focuses on context-aware smart environments. <staender@tk.informatik.tu-darmstadt.de>.

Melanie Hartmann is a Post-Doctoral researcher in the Telecooperatoin Lab at *Technische Universität Darmstadt*, Germany. She holds a Masters degree and a PhD in Computer Science from *Technische Universität Darmstadt*. Her research interests are artificial intelligence methods for user interfaces and context aware systems. <melanie@tk.informatik.tu-darmstadt.de>.

Syed Zahid Ali works as a Doctoral Researcher at the Telecooperation lab *Technische Universität Darmstadt*, Germany. He is interested in middleware and communication aspects of ubiquitous computing systems. <zahid@tk.informatik.tu-darmstadt.de>.

Max Mühlhäuser is a Full Professor of Computer Science at *Technische Universität Darmstadt*, Germany, where he leads the Telecooperation Lab. He received his Doctorate in Informatics from the *Universität Karlsruhe*, Germany, and founded a research centre for Digital Equipment. Since 1989, he has worked either as a professor or visiting professor at Universities in Germany, Austria, France, Canada, and the US. Max has published over 200 articles and co-authored and edited books about e-learning, distributed and multimedia software engineering, and ubiquitous computing. <max@tk.informatik.tu-darmstadt.de>.

Figure 1: The Four Steps of the Mundo Method for Engineering Ubiquitous Computing Systems.

over all details of the system. Of course, the engineer would try to avoid re-inventing the wheel and leverage existing components (e.g., a smart blender), established architectures (e.g., goal based [5]) and patterns (e.g., tuplespace for communication [6]) for the implementation where possible.

If the kitchen system is built following the bottom-up paradigm, there would be no designer of the overall system in the first place. Instead, different self-contained items are deployed into the kitchen. They are expected to start interacting with each other in a spontaneous way, the behaviour of the ubiquitous computing system emerges from the single parts.

The advantage of the top-down paradigm is that it can result in well designed, useful systems. However, the pure top-down approach can hardly be applied in practice outside the laboratory, as realistic environments are inevitably constructed in a bottom-up way [7]. Therefore, the bottom-up approach is more applicable in practice but, there is no guarantee that the high-level requirements of the users are met by a system created in this way.

In this paper we present a method for engineering ubiquitous computing system, called the Mundo Method. It adheres to the bottom-up paradigm and provides solutions to its two key problems, which have been unsolved so far: (i) how to structure bottom-up created systems into meaningful ensembles and (ii) how to orchestrate these ensembles so that the system behaviour reflects the requirements of the user.

In the scenario of the smart kitchen structuring means logically separating the components in the kitchen from those in the living room. Orchestrating the kitchen ensemble could mean integrating the oven with a scale so that the former automatically selects the appropriate settings based on the weight of a roast.

We present a more detailed overview of the advantages and problems of the bottom-up paradigm in Section 2. This section provides an overview of the proposed four-step method. The following four sections correspond to the four steps of the Mundo Method. Section 3 explains our approach towards building the individual components from which ubiquitous computing systems are composed. We provide a versatile hardware and software setup for turning devices without ubiquitous computing features into components for ubiquitous computing systems. Section 4 describes how these building blocks are deployed and automatically setup communication with each other. The following two sections present our solutions for the two key problems found in bottom-up engineered systems. Section 5 explains how the spontaneously connected devices can be structured according to different organizational principles. In a classical top-down solution this would have been done by the designer up-front. Section 6 deals with the orchestration of these ensembles, i.e., how to define and deploy behaviour in ensembles.

2 The Mundo Method for Engineering Ubiquitous Computing Systems

We propose a four-step method for engineering ubiquitous computing systems, shown in Figure 1. The four steps are:

build: Implement the physical and software components for ubiquitous computing systems.

deploy: Deploy these components into an environment, where they establish communication links.

structure: Define which components should interact as ensembles.

orchestrate: Define higher-level behaviour and functionality of the system.

This is the opposite of a top-down paradigm, which would start with a definition of the overall functionality and structure of the system before building and deploying the components.

As stated above, the bottom-up paradigm better reflects everyday practices. This is its most obvious advantage. We shape, construct and engineer our non-ubiquitous computing systems mostly in a bottom-up way, e.g., by buying new utensils and appliances for the kitchen. The utensils and appliances are relatively autonomous blocks. Instead of introducing functional components, e.g., for "logging", found in other approaches for building for ubiquitous computing systems, we propose to utilize these existing components as much as possible.

Moving the focus in the system decomposition away from the software engineer to the end-user resembles the change from components to services in the service oriented architecture, which provide functionality meaningful to the business user. Because the components provide meaningful functionality to the end-user, we use the term *product* to describe them. A product is a pre-confectionated package of functionality, which can be reused easily. Thus, another advantage of the bottom-up paradigm is that it results in a high degree of reuse.

Products that can be used as building blocks of a ubiquitous computing system we characterize as smart products. Smart is a rather vague term and needs to be defined. Smart is often used to describe reasoning or artificial intelligence features [8]. In contrast, in this article we define as follows: A smart product is a self-sustaining component of a ubiqui-

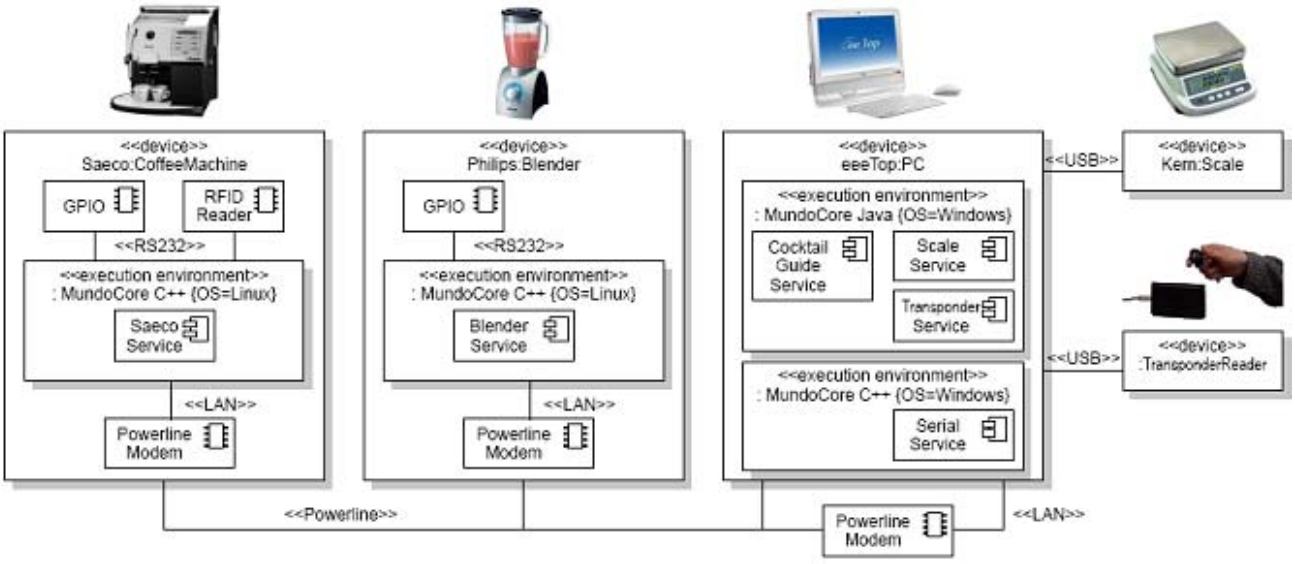

Figure 2: Architecture of the Smart Kitchen System.

tous computing system providing functionality to the system, which is meaningful to the end-user and which is able to communicate with other smart products.

2.1 Overcoming the Limitations of Bottom-Up Approaches

A ubiquitous computing system should be more than the sum of its components. The Mundo Method addresses this problem on three different levels. At the networking level, all smart products constituting a system must be enabled to communicate with each other. This problem can be addressed by suitable communication middleware. We provide the MundoCore middleware, which is specifically targeted towards the needs of ubiquitous computing systems.

However, besides low-level communication a system is characterized by:

- (higher-Level) structure, and
- (higher-Level) behaviour.

Thereby, *higher-level* refers to the structure and behaviour of multiple smart products. Experience has shown that

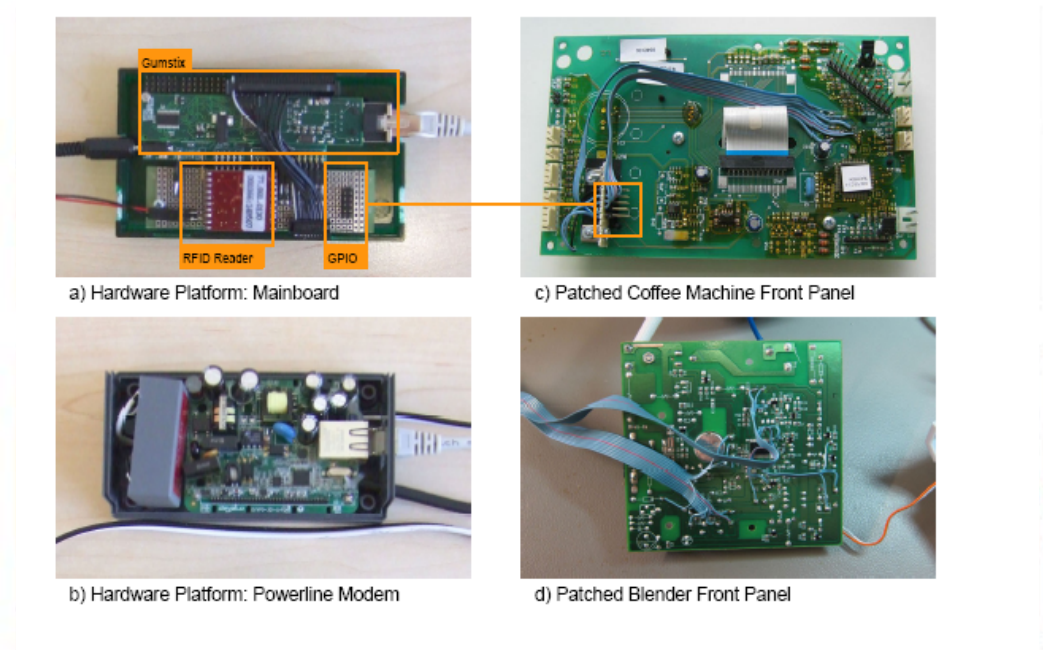

Figure 3: Smart Products Hardware Platform.

neither structure nor behaviour can be made to emerge automatically, e.g., by resorting to a common ontology for structuring the world-knowledge or planning. We propose therefore a more practical solution to these problems as part of the Mundo Method.

However, as we provide well defined interfaces to handle these problems in our approach, it can be easily adapted to use other methods. In the remainder of the paper, we will explain how the smart kitchen system shown in Figure 2 has been built using the Mundo Method. The goal was to build a system which is able to support the user in cooking a recipe with step by step instructions.

3 Building Smart Products

In order to integrate everyday electronic appliances into ubiquitous computing systems, these appliances must have built-in data interfaces. Unfortunately, e.g., kitchen appliances with integrated network interfaces are barely available on the market today.

The main reason for this is that a network software stack still requires considerable CPU and memory resources. However, appliances often only have very small 8-bit microcontrollers (e.g., the coffee machine described below) or no microcontroller at all (e.g., the blender described below). Using 8-bit microcontrollers is cheap (approx. $1), because they contain a full system-on-chip (SOC), while more powerful CPUs currently require external RAM and flash memory. This easily increases the overall system price by a factor of 10 or more. Fortunately, SOCs become more powerful from year to year, and as soon as complex networking software can be deployed on a single SOC, all kinds of appliances will come with built-in networking functionalities.

To emulate the capabilities of such future appliances with built-in network interfaces, we have designed the *Smart Products Hardware Platform* (SPHP).

3.1 Smart Products Hardware Platform

To turn an ordinary product into a smart product, we tap into the operating elements of the device, i.e., buttons, switches, dials, sensors, lights, displays, etc. Based on our earlier hardware modifications of off-the-shelf appliances [9], we have recognized that these modifications often have very similar requirements. Hence, we have designed the SPHP as a common basis for such modifications. The SPHP consists of:

> “A ubiquitous computing system should be more than the sum of its components”

> “Real world ubiquitous computing systems can realistically only be done in a bottom-up way”

- A computer-on-module, based on the Gumstix Verdex platform and the Linux operating system. It runs the *device service*, which encapsulates the functionality of the appliance as a software service. This allows monitoring and control of the device over the network (Figure 3a).
- A custom general purpose I/O (GPIO) board with an Atmel 8-bit microcontroller to run low-level, real-time control tasks. This board provides digital and analog in- and outputs. Optionally, the input and output lines can be isolated by optocouplers or isolation amplifiers. In addition, an OEM RFID reader module can be plugged onto this board (Figure 3a).
- A power line modem, based on the Devolo dLAN technology. It enables data communication over the power line with up to 14 Mbit/s (Figure 3b).

Using power line communication for smart products seems to be an elegant solution, for several reasons. Wired Ethernet cannot be used, because kitchens and similar environments do not have a network infrastructure. USB or RS232 would require a nearby PC and special attention needs to be paid to galvanic isolation. Wireless networks are difficult to set up by end-users, while power line communication offers zero-configuration.

3.2 Smart Kitchen Appliances

In the following, we describe how components for a smart kitchen scenario as mentioned in the introduction are created using this platform.

Smart Coffee Machine: This appliance is based on an ordinary off-the-shelf espresso machine. When used out of the box, the user can choose between espresso, small coffee, large coffee, and hot water. 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.

To attach the SPHP, we modified the front panel circuit board (Figure 3c). This allows us to detect key presses, simulate key presses, 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 of how much fluid has run through the pump; one pulse on this signal corresponds to 0.4 milliliters. An RFID reader is used to identify cups placed under the coffee dispenser. With all this information, we can detect the machine state and user actions to automatically start processes or proceed in a workflow, e.g., for descaling the machine.

> “To emulate the capabilities of future appliances with built-in network interfaces, we have designed the Smart Products Hardware Platform (SPHP)”

Smart Blender: The modifications to the blender are similar to those described above (Figure 3d). However, the original blender does not use a microcontroller and does not even have a galvanically isolated power supply. Hence, all digital signal lines have to go through optocouplers and the analog signal lines have to go through isolation amplifiers.

Our Smart Kitchen System (Figure 2) also uses the following two devices, which are not based on the SPHP and are simply connected to a PC:

Smart Scale: This scale from Kern has a built-in RS-232 data interface. It is connected to the PC through an USB adapter cable.

Transponder Reader: This is a custom-built reader for SimonVoss electronic door keys. Such door keys are used by several departments of our university. Building on this large deployment base (approx. 4,000 users), we use this technology for identifying users in our applications.

The SPHP turns ordinary appliances into network-capable smart products. All further aspects of system integration are now a concern of software.

4 Deployment and Networking

The individual components as described in the previous section are deployed in a kitchen environment. To become a ubiquitous computing system, the individual components need to communicate with each other. However, in a bottom-up approach, we cannot rely on hard-wired communication links. The components must establish a communication infrastructure autonomously. Just like they use the power supply network provided in the kitchen they may connect to the LAN. We call a group of communicating smart products a *smart products ensemble*.

We use the MundoCore [10] middleware to realize communication between smart products. It is an open-source communication middleware for developing, deploying and managing highly dynamic distributed systems composed of services. In the following, we first describe the basic concepts and elements of MundoCore-based systems. The *nodes* and *services* involved in the smart kitchen scenario are shown in Figure 2.

4.1 Node

A MundoCore node is an execution container that contains the MundoCore runtime environment and an arbitrary number of services. Each node corresponds to an operating system process. It is possible to run any number of nodes on a single computer. Nodes are autonomous and use peer-to-peer communication.

MundoCore versions are available for Java and C++. A Java-based node hosts Java services, while a C++ based node hosts C++ services. Because both versions use the same communication protocol, services can arbitrarily talk to each other, no matter in which language they are programmed. Hence, the developer has the freedom of choice as to whether an application service is implemented in Java or C++. In general, higher-level services, such as workflows and user interfaces are easier to develop in Java, while C++ is more efficient and provides much better access to the hardware.

In the kitchen scenario (Figure 2), every smart product contains one C++ node on the embedded Gumstix computer. This allows good access to the hardware. Java is practically not usable on this resource-constrained platform. The interaction PC hosts two nodes: one based on C++ and one based on Java. Here, C++ services are used to access the USB devices, while the main application services are implemented in Java.

MundoCore is well-suited for the bottom-up construction of ubiquitous computing systems as it supports the automatic discovery of neighbour nodes in the same subnet. This is realized by the following concepts:

- Broadcast and Multicast discovery: By using IP broadcast or multicast packets, nodes advertise their presence in the network. This discovery mechanism works inside the local subnet or the multicast scope.
- Join via the *primary port*: MundoCore guarantees that if at least a single node is running on a computer, then a node is reachable on the primary port. If the node holding the primary port terminates, then another node will try to allocate the primary port. This concept allows nodes in external networks to join a MundoCore overlay network when a single computer running a node is known.

Each MundoCore node runs a publish/subscribe broker. All brokers in the overlay network are also automatically interconnected. Most of the communications mechanisms that are used by services, such as eventing and remote method calls are based on this publish/subscribe system.

4.2 Service

The network discovery happens at the layer of nodes that discover each other as described above. Insides a node, smart product functionality is made available as services.

> “Appliances must have built-in data interfaces in order to be part of ubiquitous computing systems”

For example the smart coffee maker functionality is implemented as a MundoCore service, called SaecoService.

A service has defined input and output ports which are specified in the service interface. Ports are interconnected using *channels* provided by the publish/subscribe system. They can provide unicast or multicast connections.

4.3 Communication

MundoCore provides support for the three major traffic classes: events, request/reply, and media streaming.

The publish/subscribe abstraction is well-suited for distributing events. For example, when the SaecoService detects an RFID tag on the cup holder surface of the coffee machine, it publishes an RFID Event to the channel saeco.rfid. All services interested in this event can subscribe to the channel saeco.rfid and will receive all such events.

Request/reply interactions are supported via remote method calls, which are also performed over publish/subscribe in most cases. This enables a good decoupling of the services and provides *location transparency* as well as *execution transparency*. Hence, service consumers as well as service providers can be moved in the network, without requiring any adaptations on the side of the respective communication partner.

Consequently, when developing a smart product, one does not have to hard-wire connections to other products it depends upon, but these can be dynamically satisfied at runtime.

4.4 Service Discovery

Besides the implicit location of objects via channel names, MundoCore also provides service discovery functionality. A service can explicitly search the MundoCore environment for services fulfilling certain criteria, e.g., implementing an interface. Once discovered, a one-to-one connection for remote method calls is established.

5 Structuring Smart Products Ensembles

In a top-down designed system, all possible communication paths between products are modelled statically in the design phase. This will inevitably lead to a closed system. In contrast to that, we want to facilitate ad-hoc interactions between products that were not designed in beforehand with our bottom-up approach. Communication links are automatically formed using node discovery and products can search for required functionalities in the environment using *service discovery*.

However, we cannot permit complete freedom in these automatic discovery processes. For example, if products in the kitchen need to cooperate to realize some functionality, the products in the living room should not be considered for cooperation, as they are too far away. Hence, setting up smart product ensembles consists of two steps:

1. Structure the available smart products using scopes.
2. Within the boundaries defined by scopes, the system can automatically discover and compose services.

Communication scopes have been proposed as an adequate means for organizing and structuring distributed systems [11][12]. They emerge naturally from technical implementations, e.g., subnet or multicast groups introduce communication scopes into a system. We generalize the idea of communication scopes and propose a model for communication scopes in a middleware, called *zones*, which can be used to implement different real-world structures, even in parallel. Thereby, the *zones* model provides a common abstraction for different kinds of structures, also those that

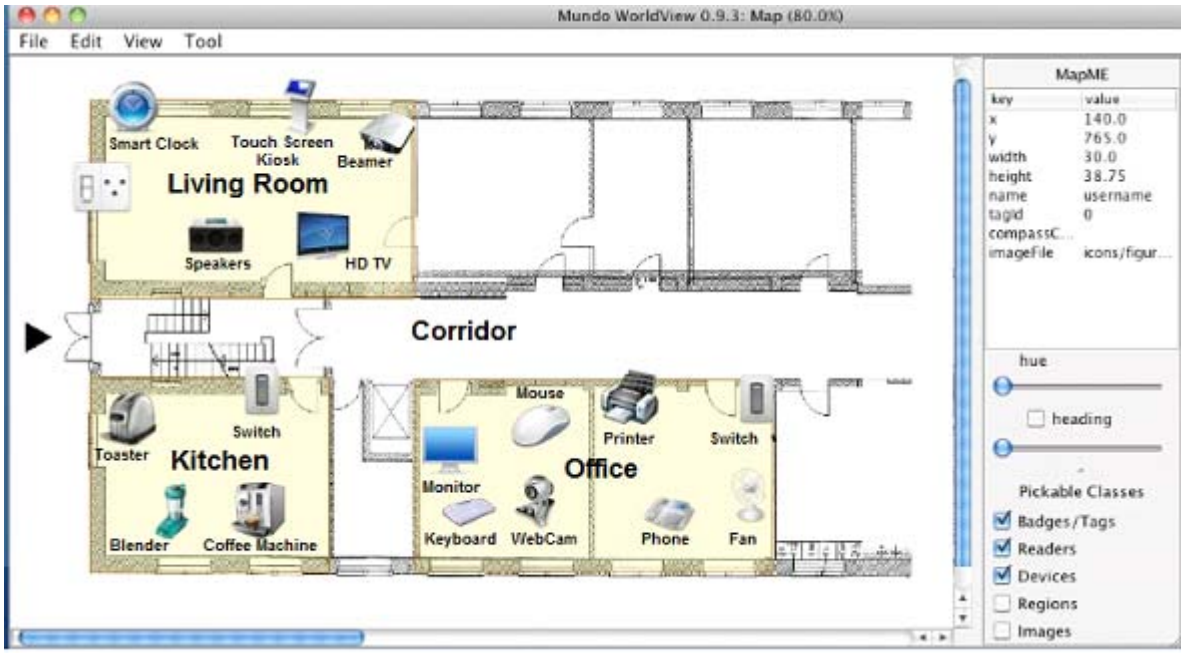

Figure 4: World View – Location-Based Zones.

do not directly emerge from the network level.

The technical communication scope is often a manifestation of an underlying structure in the real world. For example, in a ubiquitous computing system scopes can be based on location in the real world [13], organizational hierarchy, or trust relationships [14]. By taking the structure management out of applications and moving it into the middleware, applications can dynamically adapt to the changing underlying structure in the real world.

5.1 Publish/Subscribe Communication with Zones

We extend the classic publish/subscribe system model containing consumers, producers and brokers with *zones* and *zone arbiters*. A zone is a logical grouping of consumers and producers. For smart product ensembles, a zone is an organizational unit of nodes in the publish/subscribe system, in which events do not propagate beyond the zone in which they were generated. For example, smart products in the kitchen zone can only observe the events that are created by other nodes in the kitchen.

> “A zone is a logical grouping of consumers and producers”

To implement zones, the channel-based publish/subscribe model of MundoCore remains unchanged. Smart products can still subscribe and publish to channels, as described above. However, the message broker has been augmented to take the zone of the smart product into account. It will only dispatch messages to a product if it is a member of the required zone.

Every zone is identified by a URI, which may have a hierarchical structure, depending on the aspect the zone models. Currently, we model zones for the following aspects:

- Location: Smart products are joined into zones based on their location, e.g., smart products are joined into the kitchen zone if they are physically located in the kitchen.
- Organizational Structure: The zone structure can base on organizational hierarchy, e.g., the Telecooperation Lab is part of the Computer Science Department, which is in turn part of TU Darmstadt.
- Task: Ubiquitous computing systems can only work effectively when they have a model of the task or even the process the user is currently performing. Taskbased zones are automatically created and contain the products involved in the current task of the user.

5.2 Zone Arbiter

Zones are managed by a *zone arbiter* service, which is implemented in a distributed way, running on every MundoCore node. In a zone-enabled publish/subscribe system, the zone arbiter service provides the following operations to smart products:

create(X, Z) smart product X creates a new zone Z

join(X, Z) smart product X joins an existing zone Z

leave(X, Z) smart product X leaves zone Z

change(X, Zl, Zj) smart product X leaves zone Zl and joins Zj

destroy(X, Z) smart product X destroys zone Z

The zone arbiter may restrict smart products from creating and joining certain zones, based on the structure one wants to model with the zones. Currently, we have implemented basic support for zones based on location models, as described in the next section.

5.3 Location-Based Zones

As location is an obvious basis for structuring smart products ensembles, we explain the implementation of this case in more detail. Figure 4 shows the floor plan of the home containing the kitchen that should become smart. As one can see, several smart products have been deployed throughout the home. The goal of the structuring in this case is to separate the smart products in the kitchen from those in the living room.

We assume the home is equipped with a location tracking system which provides geometric coordinates to smart products. In addition to the geometric model, a symbolic location model is required that describes the structure of the home at the room level, i.e., living room, kitchen, dining room, etc.

The MundoCore middleware comprises a service for hosting such symbolic location called WorldModelStore. The model used by this service consists of named rectangular regions. The model can be edited using the WorldView tool (shown in Figure 4). This tool does not only allow offline editing of the symbolic location model, but also live inspection of the current setup.

Another service, the Context Server, also coming with MundoCore, performs the transformation between the geometric coordinates from the tracking service to the symbolic locations. The zone arbiter now continuously checks the location reported for each smart product, joining them into zones reflecting their current symbolic location. This takes into account changes in the location of the smart product, i.e., if a TV is carried from the kitchen into the living room its zone changes, as well as structural modifications of the symbolic location model. The latter allows the imposition and maintenance of a structure after smart products have been deployed in a bottom-up manner.

6 Orchestrating Smart Products Ensembles

To provide a substantial benefit to the end-user, the ubiquitous computing system must support higher-level behaviour or task knowledge which goes beyond invoking the functionality of a single product via a network interface. If we have to deal with more complex functionality, the required task knowledge can usually be specified by a set of steps that need to be performed. For example, the system should be able to guide the user through complex proce-

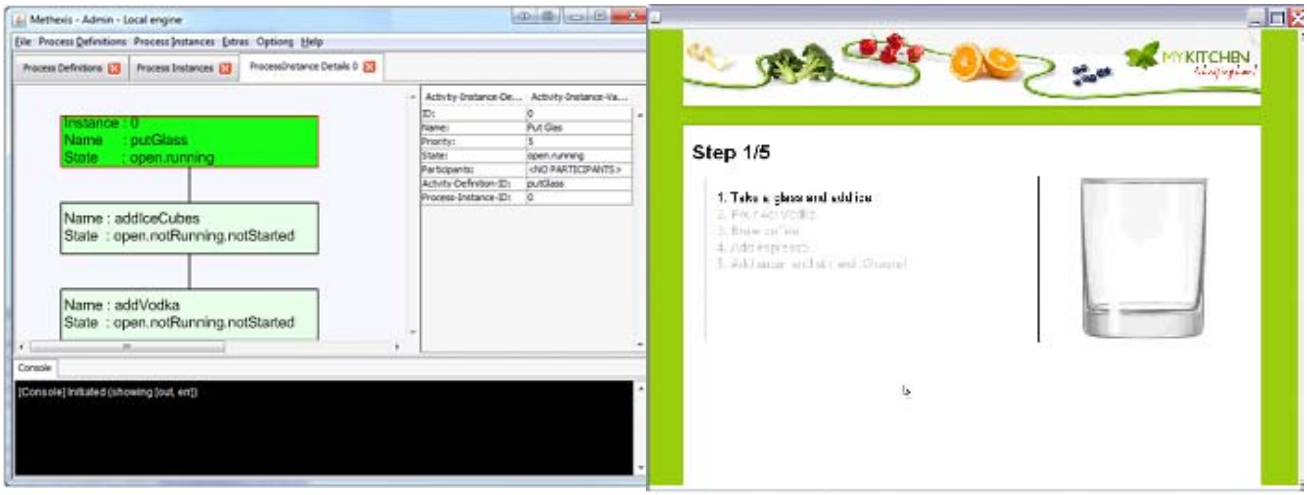

Figure 5: (a) Example Workflow and (b) the Corresponding User Interface.

dures, like the descaling of a coffee machine in the kitchen. In this case, the task knowledge can be built-in to the coffee machine.

However, in many cases, higher-level behaviour spans multiple products. A classical example being a printer and a scanner, which can together offer copy-machine functionality. In a top-down designed system, the copy-machine functionality would be modelled up-front. If the system is constructed in a bottom-up way, the question arises how is this higher-level behaviour is introduced to the system.

It could possibly come with a product, which provides task knowledge beyond its own functionality, e.g., a printer that has knowledge about copying. It is hard to generate such complex task knowledge in a completely automated way, e.g., through reasoning. For that reason, the end-user or developer should be able to specify and deploy task knowledge at runtime in an interactive way.

6.1 Context Aware Workflows

In order to execute tasks in a ubiquitous computing system, the products at first require a representation of the usually procedural knowledge necessary to carry out the task. In non-ubiquitous computing systems, workflows are used for a similar purpose [15][16][17]. Workflows are a formalized means to model higher-level procedural knowledge in a business organization. Based on the definition of the Workflow Management Coalition [18], we define a workflow as follows: "*A workflow represents a process and organizes it as a finite set of activities, containing information about their interrelations (called transitions). Activities contain information about involved organizations and users (called groups and participants), required input and output data, and tools which need to be applied*".

The activities and participants in ubiquitous computing systems differ from those found in traditional workflow systems. For example, they do not take the user interaction into account which is essential in ubiquitous computing systems. Further, ubiquitous computing settings pose special requirements on workflows. As they are used to model higher-level behaviour, spanning multiple smart products, they must also support activities that cannot be executed by the smart product running the workflow itself. In this case, the activity can be carried out by a product nearby, or as a last resort, the user might be asked to perform the activity.

We proposed a novel workflow language for ubiquitous computing systems called XPDL4USE [9]), which is based on the XPDL [20] workflow language. Using this language the activities in a workflow are described in a way that can be mapped to the service discovery of the underlying middleware. This way, an activity in one workflow may trigger a remote workflow on another smart product. For example, we consider a workflow for making a cocktail in which one activity requires crushed ice for preparing a cocktail. Further, we assume only ice cubes are available. The ice cubes can be processed into crushed ice by a blender. Therefore, this workflow activity can be performed with the help of the blender, and the activity should trigger a sub-workflow with the blender.

Figure 5 shows an example workflow for preparing a cocktail [21]. The currently active activity '*putGlass*' is highlighted (Figure 5 a), which needs to be performed by the user and is thus rendered as a user interface ('Take a glass and add ice', Figure 5 b).

6.2 User Interaction and Context in Workflows

The user needs to be approached, whenever an activity cannot be executed by any available smart product. In this case, the user must find a way to carry out the task.

Smart products often provide very limited means to interact with the user directly, e.g., a coffee machine usually only has a few buttons and a one or two line display. For that reason, the required interaction with a user should be reduced to a minimum.

This can be done by automating actions and support of implicit interaction instead of requiring explicit user input. For that purpose, the component executing the workflows

must be aware of the current state of the ubiquitous computing system and the user's activities, i.e., they must be context aware.

This enables a smart product for example to detect when the user finished a step and thus to automatically proceed to the next step. For example, a water sensor can automatically detect whether the user has filled water in the water tank instead requiring the user's explicit acknowledge via a user interface. In this case, context triggers the next activity in the workflow.

7 Summary

In this paper, we proposed a novel method called the Mundo Method to build ubiquitous computing systems in a bottom-up way as we think this approach is more applicable in practice than the top-down approach. However, the current approaches for building ubiquitous computing systems in a bottom-up way have several problems, i.e., structuring the different components in a ubiquitous computing system and specifying higher-level behaviour which involves several components. We showed how these problems can be tackled with the Mundo Method and how its different steps can be realized in practice.

Acknowledgements

Part of this research was conducted within the SmartProducts project funded as part of the Seventh Framework Programme of the EU (grant number 231204).

References

[1] Christoph Endres, Andreas Butz, Asa MacWilliams. A survey of software infrastructures and frameworks for ubiquitous computing. Mob. Inf. Syst., 1: pp. 41-80, January 2005.

[2] Itiro Siio, Noyuri Mima, Ian Frank, Tetsuo Ono, Hillel Weintraub. Making recipes in the kitchen of the future. CHI '04 extended abstracts on Human factors in computing systems, pp. 1554, New York, NY, USA, 2004. ACM.

[3] Patrick Olivier, Guangyou Xu, Andrew Monk, Jesse Hoey. Ambient kitchen: designing situated services using a high fidelity prototyping environment. Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments, PETRA '09, pp. 47, New York, NY, USA, 2009. ACM.

[4] M. Schneider. The semantic cookbook: sharing cooking experiences in the smart kitchen. Intelligent Environments, 2007.

[5] Thomas Heider, Thomas Kirste. Supporting goal based interaction with dynamic intelligent environments. Frank van Harmelen (ed.), ECAI 2002, pp. 596-600, IOS Press.

[6] B. Johanson, A. Fox. The Event Heap: a coordination infrastructure for interactive workspaces. Mobile Computing Systems and Applications, 2002.

[7] Tom Rodden, Steve Benford. The evolution of buildings and implications for the design of ubiquitous domestic environments. Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 9-16, Ft. Lauderdale, Florida, USA, 2003. ACM.

[8] Max Mühlhäuser. Smart products: An introduction. Constructing Ambient Intelligence - AmI 2007 Workshops, Darmstadt, Germany, volume 11 of CCIS, pp. 158-164. Springer Verlag, Heidelberg etc., Alemania, Feb 2008.

[9] Erwin Aitenbichler, Fernando Lyardet, Gerhard Austaller, Jussi Kangasharju, Max Mühlhäuser. Engineering Intuitive and Self-Explanatory Smart Products. Proceedings of the 22nd Annual ACM Symposium on Applied Computing (SAC), pp. 1632-1637. ACM Press, 2007.

[10] Erwin Aitenbichler, Jussi Kangasharju, Max Mühlhäuser. MundoCore: A Light-weight Infrastructure for Pervasive Computing. Pervasive and Mobile Computing, 3(4): pp. 332–361, August 2007. doi:10.1016/j.pmcj.2007.04.002.

[11] Ludger Fiege, Miraa Mezini, Gero Mühl, Alejandro Buchmann. Visibility as central abstraction in event-based systems. ECOOP Workshop on Concrete Communication Abstractions of the Next 701 Distributed Object Systems, 2002.

[12] Stephen E. Deering, David R. Cheriton. Multicast routing in datagram internetworks and extended lans. ACM Trans. Comput. Syst., 8(2): pp. 85-110, 1990.

[13] Christian Becker, Frank Dürr. On location models for ubiquitous computing. Personal Ubiquitous Comput., 9(1): pp. 20-31, 2005.

[14] Edward C. Epp. Relationship management: Secure collaboration in a ubiquitous environment. IEEE Pervasive Computing, 2: pp. 62-71, 2003.

[15] Matthias Wieland, Daniela Nicklas, Frank Leymann. Managing technical processes using smart workflows. Towards a Service-Based Internet, pp. 287-298, 2008.

[16] C.P. Kunze, S. Zaplata, W. Lamersdorf. Mobile process description and execution. Lecture Notes in Computer Science, 4025: pp. 32, 2006.

[17] Martin Bauer, L Jendoubi, O Siemoneit. Smart FactoryMobile Computing in Production Environments. Proceedings of the Mobisys 2004 Workshop on Applications of Mobile Embedded Systems (WAMES 2004).

[18] Workflow Management Coalition. Terminology & Glossary. 1999.

[19] Philip Webster, Victoria Uren, Marcus Ständer. Shaken not Stirred : Mixing Semantics into XPDL. 5th International Workshop on Semantic Business Process Management, Extended Semantic Web Conference 2010.

[20] Workflow Management Coalition. XML Process Definition Language Specification (XPDL). Specification, 2008.

[21] Marcus Ständer, Melanie Hartmann, Max Mühlhäuser. Flexible and Context- Aware Workflow Execution using Ontologies. Enviado a: ACM SIGCHI Symposiumon Engineering Interactive Computing Systems 2011, 2011.