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

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

UPGRADE monographs are also published in Spanish (full version printed; summary, abstracts and some articles online) by Novática

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/>)

UPGRADE is the anchor point for UPENET (UPGRADE European NETwork), the network of CEPIS member societies' publications, that currently includes the following ones:
- 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
- Pro Dialog, journal from the Polish CEPIS society PTI-PIPS

Editorial correspondence: Llorenç Pagés-Casas <pages@ati.es>
Advertising correspondence: <novatica@ati.es>

UPGRADE Newslist available at
<http://www.upgrade-cepis.org/pages/editinfo.html#newslist>

ISSN 1684-5285

Monograph of next issue (October 2007)
" Advanced Information Systems Project Management "
(The full schedule of **UP**GRADE is available at our website)

CEPIS UPGRADE
The European Journal for the Informatics Professional
http://www.upgrade-cepis.org
Vol. VIII, issue No. 4, August 2007

* 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/>.

Designing and Implementing Smart Spaces

Erwin Aitenbichler, Fernando Lyardet, and Max Mühlhäuser

The Mundo project at the Telecooperation Group is concerned with general models and architectures for ubiquitous computing. The Mundo Smart Environments system provides the necessary core services and tools to build applications for such environments. While the development of single services is common practice, the matter of how to coordinate services and how to make service bundles behave in a "smart" manner is still a research issue. We present a software development process for the systematic development of smart space applications. This process is supported by a set of common services and tools for modelling, inspection, debugging, testing, and rapid prototyping. We describe how these tools are applied in certain phases to support the process.

Keywords: Context Awareness, Pervasive Computing, Rapid Prototyping, Smart Space, Smart Environment, Ubiquitous Computing.

1 Introduction

Smart spaces or smart environments are living or work areas where computer technology is integrated into the building infrastructure. Typically, the goal is to ease interaction with the computer system or to make work processes more efficient. Smart spaces are inherently distributed systems with many heterogeneous computing nodes. State-of-the-art applications are based on service-oriented architectures (SOA). While it is common knowledge how to engineer single application services, the problem of how to efficiently coordinate services and how to make service bundles "smart" is largely unresolved.

In this article we focus on the context-based coordination of services on the infrastructure side. To do so, we describe a middleware layer containing a set of services useful for any smart space application. It supports the three important areas of communication, context, and coordination.

Communication: A smart space contains a multitude of heterogeneous devices. The integration of all these systems is always achieved at the software level. Thus, at a basic level, a distributed runtime system is the glue that binds together all the software services running on different platforms, operating systems, or programming languages. In addition, it must support spontaneous networking with devices brought into the environment by users.

Context: In smart spaces, interactions no longer occur in one place - they take place at changing locations or in a freely moving context. For that reason, the location of people and objects in the space play an important role. Locations of stationary objects can be modelled in a world model, while locations of mobile objects can be determined with sensors. Beside location context, various other sensors (tilt, acceleration, or weight sensors, microphones, etc.) or services (calendars, schedules, room reservation system, etc.) can provide valuable information about users and their tasks. The raw data obtained from sensors must be processed and transformed into context information that is useful for applications.

Coordination: After programming application functionality in a set of services, the logic of the overall application must either be programmed in some programming language or designed with suitable tools in a model-based approach. In this area, Rapid Application Development (RAD) tools are highly beneficial. Firstly, they allow applications to be prototyped rapidly, which is important in ubiquitous computing research. Secondly, technicians or end-users with-

Authors

Erwin Aitenbichler received his M.Sc. in Computer Science from *Johannes Kepler Universität* in Linz, Austria and his Ph.D. in Computer Science from *Technische Universität Darmstadt*, Germany. He is currently a post-doctoral researcher in the Telecooperation Group in the Department of Computer Science at the *Technische Universität Darmstadt*. His research interests are smart environments, and ubiquitous computing. Erwin is a member of the ACM. <erwin@tk.informatik.tu-darmstadt.de>.

Fernando Lyardet obtained his MSc in informatics from the *Universidad Nacional de La Plata* (UNLP), Argentina, and is currently a research assistant at the Telecooperation group, *Technische Universität Darmstadt*, Germany. His research interests include smart environments, smart products, and ubiquitous computing. Fernando is also member of the ACM and the AAAI. <fernando@tk.informatik.tu-darmstadt.de>.

Max Mühlhäuser is a Full Professor of Computer Science at *Technische Universität Darmstadt*, Germany. He received his Doctorate in Informatics from the *Universität Karlsruhe* 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>.

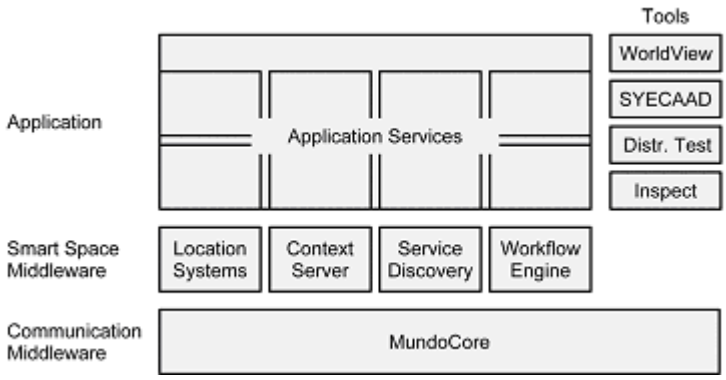

Figure 1: Mundo Smart Environments Software Architecture.

out programming skills are able to customize high-level application functionality.

This article describes the *Mundo Smart Environments* system. The work is part of the *Mundo* project [1] which is concerned with general models and architectures for ubiquitous computing. We present a software architecture for smart environments, the necessary core services, and development tools which provide solutions for the three areas described. We then describe a software development process for smart space applications and show how our tools are used in this process.

Our article is structured as follows. In Section 2 we give an overview of our Mundo Smart Environments software architecture. Based on this platform, applications are developed according to the process described in Section 3. We show how our tools are used to support certain tasks in this process. The application example in Section 4 demonstrates the application of this process. Related work is described in Section 5 and we finally conclude the article in Section 6.

2 Smart Environments Platform

The overall structure of our smart environment applications is based on a service-oriented architecture as shown in Figure 1. The communication middleware *MundoCore* [2] provides the common software basis and enables communication between the computers in the distributed system. Classical distributed systems middleware supports aspects such as naming, remote procedure calls, and distributed object computing. In ubiquitous computing, there are several new kinds of services that are required by almost any application. It is therefore useful to move this functionality into the middleware layer. Common services for smart environments include location tracking, context processing, service discovery, and workflow support. The middleware and its services will be explained in greater depth in the following sections.

2.1 MundoCore

MundoCore is the lowest middleware layer of our smart environments platform. It is responsible for all communication-related aspects and was specifically designed for the needs of services in the higher layers. The original aim of distributed object computing middleware was to enable the cooperation of objects that are independent of devices, operating systems, and programming languages. Over time, personal computer and server platforms became more powerful and had no problems running large, monolithic, and not well-optimized middleware software. Ubiquitous computing introduces a wide spectrum of new computing platforms, vastly different in terms of size, mobility and usability. The lower end of this spectrum is marked by computers embedded into everyday objects and small sensor nodes that often have only very limited processing capabilities.

MundoCore is based on 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. With its small footprint it can also be run on many embedded systems. The advantage of C++ is that it is easy to access hardware, low-level operating system APIs, and program efficient audio and video processing services. MundoCore C++ is therefore primarily used for low-level services and for services where performance is critical. Most higher-level services are programmed in Java, because of the higher productivity. The various versions of MundoCore are compatible at protocol level.

MundoCore's internal architecture addresses the need for different transport and invocation protocols, automatic peer discovery, peer-to-peer overlays, different communication abstractions, and proper language bindings. An effective API must provide programmers with functions that are appropriate to their specific application scenarios. Distributed object computing is a widely accepted and easy-to-use programming model. In context-aware and other information-driven systems, publish/subscribe is a better abstraction for distributing events because it supports multicasting and decouples data producers from data consumers. Having the right abstractions provided by middleware leads to lower application development time and a reduction in the

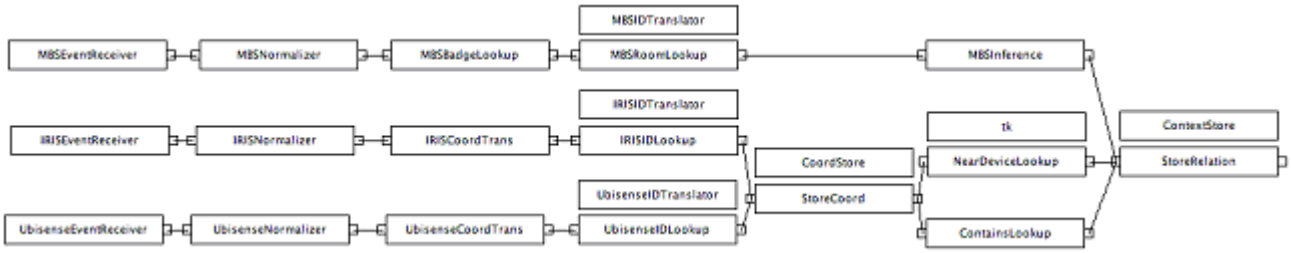

Figure 2: Context Server Widget Configuration for Locating Users.

size of application code.

In addition to events, remote method calls can also be implemented based on the publish/subscribe system. In this way, services are decoupled from one another and gain access and execution transparency. When a client sends requests to a service, the publish/subscribe abstraction provides a functionality similar to a simple naming service. Because services subscribe their interfaces at the place of execution, services can be started anywhere in the system. This allows us to define the places of execution of services at deployment time in a flexible way or to migrate services at runtime dynamically.

2.2 Context Server

The Mundo Context Server [3] is responsible for transforming readings gathered from sensors into information that is meaningful to applications. The context server provides the following functionality:

- Interpreting data received from sensors and transforming this data into a common representation.
- Maintaining a geometric world model of the smart environment and supporting geometric operations and queries.
- Inferring "higher-level" context from "lower-level" context.
- Notifying applications when certain context properties change.
- Storing histories of sensed and inferred context and supporting queries in those histories.

The server is based on the notion of widgets to process context information. The data produced by sensors is now very heterogeneous. First, the server transforms the gathered data into a small number of common data representations. For example, the different data formats and events from RFID readers, transponder readers, or infrared badge systems are transformed into a common *ID data type* and a set of common *reader* events. Similarly, positions and orientations from 3D tracking systems are transformed into a common coordinate system. Up to this point, the IDs stand for tags, transponders, or badges. The next step is to map these IDs to the IDs of persons or objects carrying those tags.

To derive higher-level context information, the context server uses modelled context, i.e., a detailed geometric (2D or 3D) world model. A world model is the virtual counterpart of the real environment targeted by the application and is basically a detailed geometric model containing walls, furniture, and other objects. The model is augmented with a metadata layer describing objects and regions of interest. With the help of this modelled context, the server is now able to infer higher-level context information from location systems that provide 3D coordinates and orientations, such as which room a user is in, which objects are near the user, or which object the user is currently looking at. Consequently, the information delivered to applications is already abstracted from the underlying sensors and describe changes in higher-level context.

For example, consider the widget configuration shown in Figure 2. It is used to derive certain relations between users and objects based on three different location tracking systems: the Mundo badge system, an IR optical tracking system, and the Ubisense tracking system. The badge system directly provides symbolic location information. Thus, only a mapping from system-specific badge and room IDs to global IDs is necessary.

Data from the 3D tracking systems must also be interpreted using the world model. All derived relations are written into a tuple space. With the widget configuration shown in Figure 2, the situation in Figure 4, and given that users can be tracked by at least one of the three location systems, the context server would derive the relations for room A112 shown in Table 1.

Applications can query the tuplespace or subscribe to changes. A query operation returns all relations matching a specified pattern. When an application subscribes to the tuplespace, it will be notified each time a new or updated tuple matching the specified pattern is written to the space.

There are two major reasons why context processing is implemented as part of the middleware, separate from applications. Firstly, this service is required by many ubiquitous computing applications and if it is part of a common middleware it can be easily reused. Secondly, context processing is a long-running task and may need to record information over a long time.

Applications may run only for a short time but may need to access context information from the past, collected over a long period of time. Accessing information from the past requires this information to have been recorded beforehand. Consequently, the context server must also be configured so that it records all the information that will be needed later.

User:Erwin	in	Room:A112
User:Erwin	near	Device:PolyvisionBoard
User:Andreas	in	Room:A112

Table 1: User Relations for Room 112 as Derived by the Context Server.

3 Development Process

Figure 3 shows the different phases of the software development process. It is based on the waterfall model and has been refined for certain tasks. In many phases existing tools are sufficient, e.g., an IDE for Java programming, or JUnit to perform unit tests. For several other phases we introduce additional tools to support the development process. In the following section we describe how these tools support the different phases of software development.

3.1 Analysis Phase

In the analysis phase, it is first necessary to identify which context information the application can benefit from. Because the use of context information is highly application-specific, it is hard or impossible to create a universal context model which covers every possible requirement. Instead, our aim is to define processes describing how to select sensors, configure processing, and how to make this context information accessible to applications.

On the basis of information requirements, the necessary sensors can be selected. Especially for location tracking systems, the required resolution and accuracy plays a major role in this selection process.

Next, the interface to the application is defined. This includes message format and the kinds of subscriptions and queries needed by the application.

Finally, the context server is configured such that it transforms the raw sensor readings into what the application needs. Because context information will only be available for later retrieval when it is recorded in time, database widgets must be configured accordingly. It must be decided which information has to be stored and for how long.

3.2 Design Phase

WorldView is a versatile tool with functions to support the modelling, implementation, and testing phases. In the modelling phase, WorldView is used to create a spatial model of the smart space. It supports 2D models as well as detailed geometric 3D models. In the map window, the application shows the floor layout and provides an overview of the available resources and their locations. Resources include tags and sensors of different location systems, wall displays, and smart doorplates.

WorldView provides an easy way to define "regions of interest". These are regions of arbitrary shape which can be annotated with metadata. Regions can be uploaded to the context server as part of the world model and accessed by context widgets in order to derive higher-level context. Location-aware applications can then specify subscriptions based on semantic location instead of having to work with coordinates of the underlying location tracking system. In this way, WorldView provides a simple way to define the regions of interest that should trigger spatial events.

3.3 Implementation Phase

A lot of research into ubiquitous computing is being conducted around possible application scenarios that can be put to daily use. Many of these scenarios are quite simple and straightforward to implement. However, many of these applications never come to fruition, because the whole process from development to deployment is still very complex. To write a new application, the developer typically has to start his or her IDE, install all required libraries, write and test code and then deploy the application on a server. For that reason, we wanted to make this development process as quick and easy as possible. One aim was to enable a wide variety of people with some basic technical background, but not necessarily with knowledge of a programming language, to create and customize applications. Simple-structured applications can be directly developed with the tools provided, while more complex applications can be prototyped.

The *SYstem for Easy Context Aware Application Development* (SYECAAD) [4] facilitates the rapid development of context-aware applications. Simple applications can be completely developed with this tool and more complex systems can be prototyped. Applications are built using a graphic-oriented block model. The basic building blocks are called *functional units*. Functional units have input and output pins and are interconnected to form *functional assemblies*.

Functional units come in three different flavours: sen-

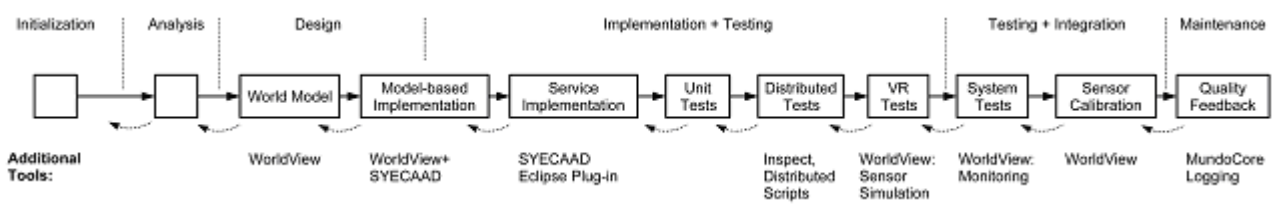

Figure 3: Tool Support for the Different Phases of Software Development.

sors, operations, and actors. A sensor unit either receives data directly from a sensor or pre-processed data from the context server. Operations perform logical or arithmetic operations, implement dictionaries, render HTML pages, etc. On the output side, actors can control the smart environment or send feedback to users by publishing MundoCore events or by invoking methods on arbitrary remote services. This way, an application can, say, control smart power plugs, control data projectors, send emails, send SMS to mobile phones, send instant messenger messages, or display information on electronic doorplates.

Functional assemblies are a modularization concept. Assembles can be installed, removed, restarted, and edited independently. In practice, the logic for each room in a building would be modelled in a separate assembly. A special feature of SYECAAD is that it can compute the output states of functional units based both on an event-based and a state-based model. Each time the value of a sensor changes, all operations depending on this value must re-evaluate their state. In an event-based model, changes are propagated as messages throughout the system. However, when a functional unit is started, output values are not available until all inputs have propagated change messages. This is very impractical during system development. In a state-based model, the state of all units is evaluated at a fixed interval. This approach is not efficient, but all output states are available immediately. The hybrid model used in SYECAAD permits an efficient execution and still allows changes to be made in the running system almost without losing state.

SYECAAD uses a client/server architecture. The server hosts the applications. Clients connect to the server and allow running applications to be controlled, edited, and tested. The *Application Logic Editor* in *WorldView* is such a client to edit applications (Figure 4). This system significantly simplifies application development. It is no longer necessary to set up a development environment, because all the application logic is centrally stored on the application server. The development environment is a client application that connects to this server. In this way, an application can be loaded from the server, displayed, edited, and deployed with the click of a button.

If the standard sensor, operation, and actor blocks are not sufficient, the system can be extended by programming new blocks in Java. A plug-in for the Eclipse IDE supports the programmer with code templates and help documents.

3.4 Testing Phase

Implementations of abstract data types and smaller units of frameworks can be successfully tested with *unit tests*. However, to verify the correct behaviour of a distributed system, it is also essential to run integration tests across multiple computers.

To conduct such tests, we have implemented a *Distributed Script Interpreter*. Script server processes are started on multiple hosts in the network. The script servers and the master script interpreter are based on MundoCore for communication. This allows them to automatically discover each other on startup. To run a test, the name of an XML script file is passed to the master interpreter. The master interpreter then distributes the subtasks to the server processes in the network and finally collects all test results.

The *MundoCore Inspect* tool is a low-level tool that directly builds on the communication middleware. It can connect to an arbitrary remote node and manage the hosted services. The program makes it possible to view the routing tables, list the import and export tables of message brokers, monitor the messages sent over a channel, view service interfaces, dynamically call remote methods with a generic client, view service configuration information, and reconfigure services.

With *VR-Tests* it is possible to test applications from the desktop. *WorldView* can be used to simulate certain tracking systems. In this case, the user can move around the symbols on the map and *WorldView* generates the same kind of events the actual tracking system would. This significantly reduces application development times because users do not have to get up from their workplace and move physical objects around every time they want to test their applications.

3.5 Deployment and Maintenance

When the results from VR Tests are satisfactory, then the application can be tested together with the real sensors. In this phase, the *WorldView* application can be used to inspect the running system by visualizing the events from certain event sources, like tracking systems. If a tag is physically moved around, the position of the corresponding symbol in the map view is updated in real-time.

The MundoCore middleware implements heap debugging, system resource tracking, deadlock detection, progress monitoring, and logging. Some bugs are not discovered until the system is tested with the real sensors. In this case, detailed logs provide important information for developers to fix the problem.

4 Application Example

We will describe three simple applications created with our tools in the following. The configuration for three rooms is shown in Figure 4.

Office: For the office room A109 we defined that if the light level is below a certain threshold and there is at least one person in the room, then the lights are turned on. Alternatively it is possible to use the manual light switch. This application is based on a light sensor and a location tracking system as inputs, and LON[1] -controllable lamps as outputs.

Coffee kitchen: The coffee kitchen A120 configuration describes that if at least two people stay in the kitchen for some time, then an instant message is sent to all people in

[1] Local Operating Network: a bus system for building automation

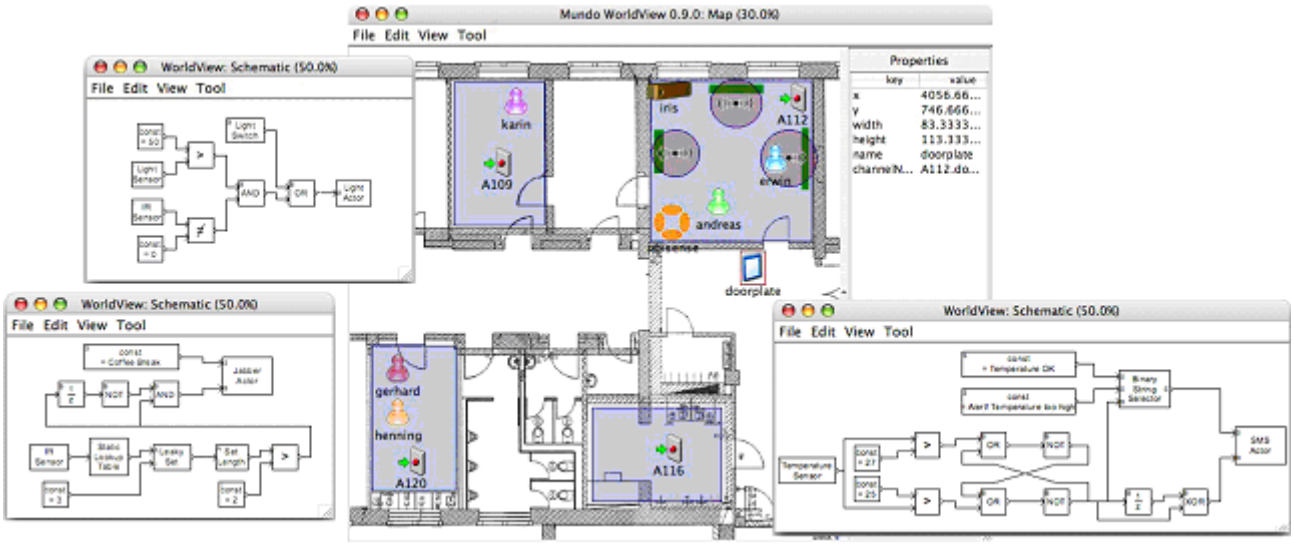

Figure 4: Spatial Model in WorldView and Application Logic for three Rooms.

offices nearby, inviting them for "socializing". This application is based on a location tracking system as input and a service implementing the Jabber protocol to send instant messages.

Server room: In our new computer science building the air conditioning system fails from time to time. The configuration for A116 defines that if the temperature exceeds 27 degrees, then the system administrator is notified by SMS. When the temperature drops below 25 degrees, another SMS is sent. This application is based on an SNMP thermometer as input and a service for sending short messages via an Internet website.

5 Related Work

Intelligent environments are examined in numerous projects, such as Gaia [5], Aura [6], Nexus [7] and iWork [8], and many toolkits and infrastructures have been constructed for the purposes of developing and supporting such applications. These technologies provide novel ideas to address different design concerns such as the interface [9] [10] [11], collaboration between heterogeneous devices [12], and context awareness [13]. Some tools for developing environment models are also available from commercially available LBS products such as Ubisense [14] and Elpas [15], but their intended audience is different. While Ubisense provides a C++ interface aimed at developers, the rule-based approach of Elpas allows end-user development through its rule editor.

The ability of end-user development has been also pursued through various approaches. For instance, CAMP [16] provides a simplified natural language using the idea of "Magnetic Poetry", and Hull describes a system for providing media landscapes based on environment model and scripting [17]. Also, in the context of the Nexus project, a tool was developed for smart environments programming based on flow charts [7]. However, these projects cover only very specific application and design concerns.

Contrary to the previous examples, our system covers all phases of the development process with a flexible platform and reusable services and tools. The description of the process presented in this article is an important contribution to the standardization of the development of this kind of applications.

6 Summary

Creating context-aware applications for smart spaces raises the need for novel platforms and tools to support application development. At a basic level, our middleware MundoCore provides the necessary integration platform to enable communication between the heterogeneous devices in the environment. Our smart environments system supports a set of common services at the middleware layer. The context server interfaces with sensors, transforms sensor data to information that is meaningful to applications and provides applications with a query and event subscription interfaces. The *WorldView* tool supports modelling of smart environments and monitoring. The *SYECAAD* tool facilitates the rapid development of context-aware applications based on visual programming. We have described a software development process and how these tools are embedded into the process, thereby contributing to the systematic development of smart space applications.

References

[1] Andreas Hartl, Erwin Aitenbichler, Gerhard Austaller, Andreas Heinemann, Tobias Limberger, Elmar Braun, Max Mühlhäuser. Engineering Multimedia-Aware Personalized Ubiquitous Services. In IEEE Fourth International Symposium on Multimedia Software Engineering (MSE'02), pages 344–351, December 2002.

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

[3] Marek Meyer. Context Server: Location Context Support for Ubiquitous Computing. Master's thesis, Darmstadt University of Technology, January 2005.
[4] Jean Schütz. SYECAAD: Ein System zur einfachen Erzeugung kontextsensitiver Applikationen. Master's thesis, Technische Universität Darmstadt, 2005.
[5] Manuel Roman, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, Klara Nahrstedt. A Middleware Infrastructure for Active Spaces. Pervasive, 1(4):74–83, October 2002.
[6] David Garlan, Daniel P. Siewiorek, Asim Smailagic, Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. Pervasive, 1(2):22–31, April 2002.
[7] Torben Weis, Marcus Handte, Mirko Knoll, Christian Becker. Customizable Pervasive Applications. In PerCom 2006, 2006.
[8] Brad Johanson, Armando Fox, Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. Pervasive, 1(2):71–78, April 2002.
[9] Rafael Ballagas, Meredith Ringel, Maureen Stone, Jan Borchers. istuff: a physical user ce toolkit for ubiquitous computing environments. In CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 537–544, New York, NY, USA, 2003. ACM Press.
[10] Saul Greenberg, Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology, pages 209–218, New York, NY, USA, 2001. ACM Press.
[11] Scott R. Klemmer, Jack Li, James Lin, James A. Landay. Papier-mache: toolkit support for tangible input. In CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 399–406, New York, NY, USA, 2004. ACM Press.
[12] Peter Tandler. Software infrastructure for ubiquitous computing environments: Supporting synchronous collaboration with heterogeneous devices. In Ubicomp, pages 96–115, 2001.
[13] Daniel Salber, Anind K. Dey, Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In CHI, pages 434–441, 1999.
[14] Ubisense. The Smart Space Company. <http://www.ubisense.net/>, 2007. Last visited: 30.06.2007.
[15] Visonic Technologies. Elpas. <http://www. visonictech. com>, 2007. Last visited: 30.06.2007.
[16] Khai N. Truong, Elaine M. Huang, Gregory D. Abowd. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In Ubicomp 2004, 2004.
[17] Richard Hull, Ben Clayton, Tom Melamed. Rapid Authoring of Mediascapes. In Ubicomp 2004, 2004.