Development Tools for Mundo Smart Environments

Erwin Aitenbichler
Telecooperation Group
Darmstadt University of Technology
Hochschulstrasse 10
64289 Darmstadt, Germany
erwin@informatik.tu-darmstadt.de

1 Introduction

The *Mundo* project [2] at our group is concerned with general models and architectures for ubiquitous computing systems. The present paper describes the tools created in this project to support the development of applications for smart environments.

First, we give a brief description of the overall structure of such applications. The common software basis is formed by the communication middleware *MundoCore* [1]. It 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. The architectural model addresses the need for proper language bindings, different communication abstractions, peer-to-peer overlays, different transport protocols, different invocation protocols, and automatic peer discovery. Most importantly, MundoCore serves as an integration platform that allows to build systems out of heterogeneous *services*.

Applications consist of a set of common services and application-specific services. Common services for smart environments are offered, e.g., by our *Context Server* [3] and an application server [4]. The Context Server is responsible for transforming the readings received from sensors into information that is meaningful to applications. It builds on the notion of context widgets. To facilitate the development of applications for smart environments, we have created a number of tools in addition. In the following, we describe how these tools support the different phases of software development (Figure 1).

2 Modeling Phase

WorldView is a versatile tool with functions to support the modeling, implementation, and testing phases. In the modeling 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 the regions of interest that should trigger spatial events for location-aware applications. The created maps can be uploaded to the Context Server and then be accessed by context widgets in order to derive higher-level context information.

3 Development Phase

A lot of research in ubiquitous computing is 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 life, because the whole process from development to deployment is still very complex. To write a new application, the developer typically has to start 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 easy and fast as possible. An 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 alter applications. Simple-structured applications can be directly developed with the provided tools, while more complex applications can be prototyped.

SYECAAD: The SYstem for Easy Context Aware Application Development (SYECAAD) [4] facilitates the rapid development of context-aware applications. 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 flavors: sensors, operations, and actors. A sensor unit either receives data directly from a sensor or preprocessed data from the Context Server. Operations perform logic or arithmetic operations, implement dictionaries, render HTML pages, etc. On the output side, actors can control the smart environment or send feedback to users. Actors can publish MundoCore events or invoke methods. This way, an application can e.g., control smart power plugs, control data projectors, send emails, send SMSs, send instant messenger messages, or display information on electronic doorplates. An electronic doorplate is a 8-inch color TFT display with touchscreen that replaces an ordinary doorplate.

SYECAAD uses a client/server architecture. The server hosts applications. Clients connect to the server and permit to control, edit and test running applications. The *Application Logic Editor* in *WorldView* is such a client to edit applications. This system addresses all the development steps described above. The first step, namely setting up the development environment, is no longer necessary, 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.

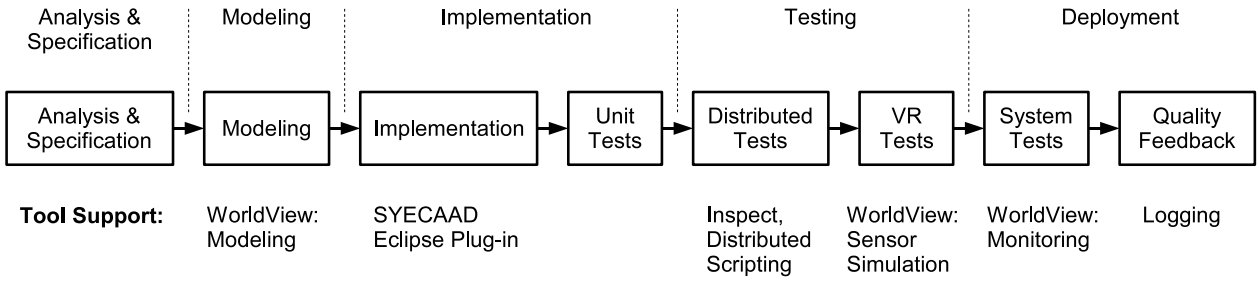

Figure 1: Tool support for different phases of software development (multiple iterations of steps possible)

Eclipse plug-in: 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.

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 behavior of a distributed system, it is also essential to run integration tests across multiple computers.

Distributed Tests: To conduct such tests, we implemented a *Distributed Script Interpreter*. Script server processes are started on multiple hosts in the network. The script servers and the master script interpreter also use MundoCore for communication. This enables 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. When the master interpreter encounters a `execRemote` instruction, it starts a new process on a remote script server and passes the text contained in the tag to the standard input of the remote process. The standard output of the remote process is passed back from the remote script server to the master interpreter and written to a log file. The shell return codes indicate if processes were successful running their sub-tests.

Inspect: The *MundoCore Inspect* tool can connect to an arbitrary remote node and manage the hosted services. The program allows to view the routing tables, list the imports and exports tables of message brokers, monitor the messages passed over a channel, view service interfaces, dynamically call remote methods with a generic client, view service configuration information and reconfigure services.

Sensor Simulation: To test applications, *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 tracking system would.

5 Deployment Phase

Monitoring: *WorldView* can also 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.

Quality Feedback: 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.

6 Summary

The development process of distributed, loosely-coupled, and context-aware applications raises the need for novel tools. Such tools to support the different phases of software development, i.e., modeling, implementation, testing, and deployment have been presented.

Bibliography

[1] Erwin Aitenbichler, Jussi Kangasharju, and Max Mühlhäuser. Experiences with MundoCore. In *Third IEEE Conference on Pervasive Computing and Communications (PerCom'05) Workshops*, pages 168–172. IEEE Computer Society, March 2005.

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

[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.