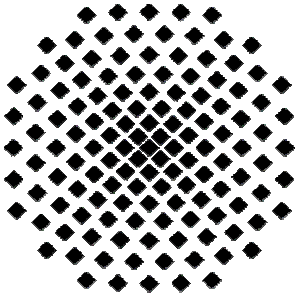


Universität Stuttgart

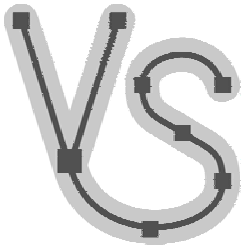
Studiengang:	**Softwaretechnik**
Betreuer:	**Martin Bauer**
Prüfer:	**Prof. Rothermel**
Beginn am:	**1.4.2003**
Beendet am:	**30.9.2003**
CR-Klassifikation:	**H.2.4, C.2.4., H.3.3.**

Diplomarbeit Nr. 2075

A Spatial Modelling Infrastructure for Ubicomp Applications in Home Environments

Othmar Lehmann

Fakultät Elektrotechnik, Informatik,
Informationstechnik
Universität Stuttgart
Institut für Institut für Parallele und Verteilte Systeme
Abteilung Verteilte Systeme
Universitätsstr. 38
70569 Stuttgart



KURZFASSUNG

Ubiquitous Computing ist ein Forschungsfeld, das in den vergangenen Jahren steigendes Interesse erfahren hat. Gerade in Wohnumgebungen hat Ubiquitous Computing das Potential, das tägliche Leben zu verändern. Räumliche Modelle sind ein nützliches Hilfsmittel des Ubiquitous Computing. Ubiquitous Computing in Wohnumgebungen stellt nun besondere Anforderungen an die dort benutzen räumlichen Modelle und die Infrastrukturen zu ihrer Verwaltung.

In dieser Diplomarbeit wird unabhängig von Nexus ein räumliches Modell und eine Infrastruktur zur Verwaltung dieses räumlichen Modells entwickelt, das speziell auf die Anforderungen für Wohnumgebungen zugeschnitten ist. Am Beispiel des Aware Home des Georgia Institute of Technology werden die Anforderungen an das räumliche Modell und die dazugehörige Infrastruktur untersucht. Diese Anforderungen bilden die Grundlage für eine pragmatische, leichtgewichtige und flexible Infrastruktur, den Aware Home Spatial Service (AHSS).

Um es dem AHSS zu ermöglichen, auch in größeren Einsatzszenarien zu funktionieren, die über ein einzelnes Haus hinaus gehen, wird AHSS dann in die Nexus Plattform eingebunden. Weiterhin werden Vorschläge gemacht, wie das Nexus Augmented World Model erweitert werden kann, um Ubiquitous Computing in Wohnumgebungen besser zu unterstützen.

ABSTRACT

Ubiquitous computing is a field that has drawn increasing attention of the research community in recent years. Ubiquitous computing in home environments has the potential of influencing our everyday life to a great degree. Spatial Models are a useful tool for ubiquitous computing. Ubiquitous computing in home environments has special requirements for the spatial models and the infrastructures to manage these models.

The goal of this thesis is to create a spatial model and an infrastructure to manage this spatial model which is specifically tailored for ubiquitous computing in home environments. The requirements for this model and the management infrastructure are examined, using the Aware Home of the Georgia Institute of Technology as an example. Based on these requirements, a lightweight, flexible and pragmatic infrastructure is created, the Aware Home Spatial Service (AHSS).

To facilitate the use of the AHSS in larger usage scenarios which go beyond a single house, AHSS then is integrated into the Nexus platform. Also a proposal is presented to extend the Nexus Augmented World Model to better support ubiquitous computing in home environments.

ACKNOWLEDGEMENTS

I would like to thank Gregory Abowd, who made me part of his research group and gave me a lot of freedom to pursue my own ideas. Being part of this great group was a very rewarding experience, and the things I learned are invaluable.

Also, I would like to thank Christian Becker, without whom this thesis would never have been written and Matthias Grossmann for clarifying the depths of AWML.

I am especially grateful to Daniela Nicklas and Martin Bauer for their great support while supervising this thesis. They even supported me during the weekend.

And last but not least, I want to thank the whole Nexus group for their advice and fruitful discussion.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

There's no sense in being precise about something when you don't even know what you're talking about.
– John von Neumann

1.1 PROJECT DESCRIPTION

The objective of this diploma thesis is at first to develop a spatial model and a spatial server to manage a local spatial model of a smart home environment. It has to manage static and dynamic objects in the home environment, thus supporting context-aware indoor applications. This specialized infrastructure is independent of Nexus.

Once this infrastructure is finished, it has to be examined how this special-purpose system can be integrated into a more general infrastructure like Nexus. Therefore, the Nexus Augmented World Model has to be extended to include objects from the indoor application domain.

It furthermore has to be examined which roles such a special purpose server can take in the Nexus platform (e.g. as a spatial model server, a location server or as a value-added service). The necessary interfaces then have to be implemented to make the specialized infrastructure work as a part of the Nexus platform.

1.2 MOTIVATION AND OBJECTIVES

The field of ubiquitous computing in smart environments has recently seen a rise in popularity. Ubiquitous computing moves the computer from the desktop into the environment of the user. So computers, and ultimately computing applications, become part of the environments – and are also situated within them.
One consequence of this development is that many ubiquitous computing applications need context information. This context information can be naturally structured along the dimension of space. So they can profit from a spatial model of their environment.
In desktop computing, the location of the computer and of the user is of relatively low importance. Typically, location information in these contexts is only needed in form of a time zone to properly set the system clock and maybe a telephone area code in order to configure dial-up networking. But when computers move into the environment, spatial information becomes more important for a variety of reasons.

Spatial Information is a major part of the context information of ubiquitous computing applications. Context information is information that can be utilized to characterize the situation of an entity. It will be described in detail in Section 2.1.3. Spatial information can also be used to structure context models. This will be explained in Section 2.1.4. This way, spatial information can form the index to more general context models.

Spatial information is necessary for many types of interactions with the real world. Sensors and actuators are situated within an environment. Most of the time it is necessary to have certain information about the location of these “real world

resources". Consider a very simple "illumination control" application that switches on the lights whenever a user enters a room. The application has various presence detectors distributed throughout the house, and also controls various actuators which are attached to the house network that can switch lights on and off. Now it receives a message from "detector6673" that someone has entered a room. So the application has to know that for example "actuator3354" controls the lights in that room. Or maybe there are multiple actuators to activate. This information is information about the spatial structure of the environment.

This spatial information, and the spatial structure it describes, is a common concept for all the applications that work within the environment. A multitude of applications can serve a given environment – from light control to communication applications. They can be from very different application domains. They can be from different programmers. They can use different technologies. But there is one thing that they have in common: The space they are working in.

While spatial information is crucial for many applications in ubiquitous computing, it is expensive to collect and to maintain. It is specific for every deployment scenario of a ubicomp system. This means that for each location that a ubicomp application is installed in (e.g. different houses) it has to be collected again. Floor plans, plans of sensor deployment, information about the sensors, plans of electrical installations etc. have to be used to get the information. Then the information has to be provided to each program that needs it, which in turn could expect this information in a different format. This severely impedes easy deployment of standardized ubiquitous computing applications. It is not possible to purchase an off-the shelf ubicomp application and to simply deploy it in your environment, because it always needs to be tailored for the spatial information of your environment.

Also, spatial information is subject to change. Sensors and actuators are added, removed or upgraded. Doors get blocked, furniture gets moved around. These changes in the spatial information need to be propagated to all applications that rely on this information. If this information is scattered throughout the different applications, and represented in different formats, then keeping this information current and consistent becomes a very challenging task.

One way to address these challenges is to introduce a standardized way to represent spatial information, and to store it in a specialized infrastructure. That way, the spatial information of a ubicomp environment only has to be acquired once, and all applications can use this information. If this spatial information is then stored in a central database, it is easier to keep it consistent between the different applications. This also allows ubicomp applications to use this centralized infrastructure to exchange information that is related to space. So if an application computes information about a spatial entity, it could in fact attach it to the representation of this spatial entity in the infrastructure that stores spatial information. Then every other application could access that information.

The goal of this thesis is to develop a way to represent spatial information and to create an infrastructure to store this information, so that applications can use it. Then this system should be integrated into a larger context – the Nexus platform.

1.3 APPROACH AND OVERVIEW

The goal of this thesis is to create a lightweight, flexible and pragmatic spatial model and infrastructure that facilitates spatial modeling for ubiquitous computing scenarios in home environments.

The first part of this thesis deals with the development of the specialized spatial modeling infrastructure for smart home environments. So after this introduction in Chapter 1, Chapter 2 lays out the environment that influenced the spatial modeling infrastructure created here. It introduces the ubiquitous computing domain that this infrastructure wants to support, shows what characterizes the special case of ubiquitous computing in home environments, and then introduces the Aware Home, the research environment this infrastructure was created for. A discussion of the concepts of spatial modeling and of relevant related work concludes Chapter 2.

Chapter 3 describes the requirements for the specialized infrastructure, the Aware Home Spatial Service (AHSS). It presents some scenarios and then derives requirements for a spatial modeling infrastructure.

Chapter 4 describes the design of AHSS. It describes the spatial model that the infrastructure is based on and the architecture of the infrastructure that provides the spatial model to the applications.

Chapter 5 discusses some of the technologies that have been used to implement the AHSS system, and elaborates on why they have been chosen.

Chapter 6 describes the experiences with the AHSS system and provides an evaluation.

The following Chapter 7 then deals with the integration of AHSS into the Nexus platform. It gives an overview of the Nexus platform architecture and discusses how the different approaches of Nexus and AHSS are reconcilable. A description of the interfaces that AHSS had to implement in order to function as part of a Nexus federation follows next. Then a description of the implementation of the integration of AHSS into Nexus is given, and the experiences gained from this integration are discussed.

The following Chapter 8 describes extensions to the Nexus augmented world model in order to better support ubiquitous computing applications in home environments.

CHAPTER 2

THE ENVIRONMENT

The art of programming is the art of organizing complexity.
– Edsger W. Dijkstra

This Section describes the environment this thesis is situated in. Section 2.1 introduces some basic terms that will be used in this thesis. Section 2.2 then describes the Aware Home and Nexus, two projects that are the environment this thesis was created in. Section 2.3 then concludes with the discussion of relevant related work.

2.1 DEFINITIONS

This chapter introduces major terms that are crucial to this thesis.

2.1.1 UBIQUITOUS COMPUTING

The term “Ubiquitous Computing” was forged by Marc Weiser while working at the Xerox Palo Alto Research Center. From 1988 to 1993, Weiser and his group created a variety of applications that laid the foundation for the idea of ubiquitous computing. In 1991, Weiser published a landmark paper [WEISER 1991] which made the concept of ubiquitous computing popular.

With the current paradigm of desktop computing, one user usually uses one computer. Ubiquitous computing, also called pervasive computing, is a movement away from the one user – one computer paradigm. In [WEISER 1991], Weiser states that “The most profound technologies are those that disappear.” Ubiquitous computing aims at removing the computer as a recognizable object, and making it a ubiquitous part of our environment. Computers are no longer only objects that sit on desks and demand our attention. Instead, they are embedded into the environment and offer their services in an intuitive way. Weiser compares this to the concept of reading. The “technology” of reading has become a part of our daily life we do not think about anymore. Writing is used in books and newspapers. But it is also pervasive on labels, traffic signs or directly on products. In our daily life, we do not explicitly think about the “technology” of literacy anymore, we just perceive the information that is written down. Writing has become truly pervasive.

On the other hand, current computers are not really pervasive. If I need the services of a computer, then I have to go to the computer that sits on my desktop and switch it on. Then I have to start up the right program. An example may be:
“Click Start, then find the right program group. The Web Browser is located in the “Internet” program group. Now find the right web site to get the timetable for a train. Fill out the form indicating start station, end station and the time.”

While interacting with the computer, you have to use his terms and his paradigms. The computer still remains a monolithic object that sits on your desk. It remains separate and in a way foreign to your natural surroundings. It does not integrate seamlessly into your daily life.

Ubiquitous computing aims to move the computer away from the desktop and out into the world. But ubiquitous computing is not just a question of making computers portable. So even a perfectly portable laptop that I can take anywhere does not make computing really ubiquitous, as long as the computer remains a discrete artifact. In analogy to the “literacy” example from above: A laptop would be one important book that I own and carry around where everything is written down. This does not even come close to the real ubiquity of writing in our environment, where we see billboards on the side of the road, warnings on traffic signs, or timetables at bus stops.

So pervasive computing devices are not personal computers as we think of them right now. They are instead specialized devices of various scales that are embedded into almost every part of the environment. They can be mobile or embedded, integrated into clothing, tools, cars, appliances, or consumer goods. They can be as small as a stamp and be attached to one of my books to identify it, and as large as a 5 feet by 5 feet wall display. The goal of ubiquitous computing is to create a system that is pervasively and unobtrusively embedded in the environment, completely connected, intuitive, effortlessly portable, and constantly available.

Ubiquitous computing should not be confused with virtual reality. In a way, ubiquitous computing is the complete antithesis to it. Virtual reality aims at creating an artificial world for the user. It tries to completely immerse the user into the world of the computer by using special goggles, gloves and a variety of special equipment. Ubiquitous computing on the other hand tries to make the computer part of the real world. Major challenges of virtual reality lie in processing power and algorithms to make the artificial world come to life. The main challenges of ubiquitous computing are more the very difficult integration of the human factors, computer science, engineering, and social sciences.

Over the last years, ubiquitous computing has drawn an increasing amount of interest from researchers and from the industry. A major reason is that technology has advanced to a point where the vision of ubiquitous computing becomes more and more viable. Computing power is available at low prices, and advances in mobile computing have made applications possible that were only keen visions a few years ago. It has become possible to integrate sensors into everyday objects. Adding communication capabilities to these objects is the next logical step which will bring ubiquitous computing to the next level.

2.1.2 Ubiquitous Computing in Home Environments

People spend the majority of their life at home, so bringing ubiquitous computing to the home domain would have considerable impact. The idea is to add intelligence to home environments to support the inhabitants with their daily tasks. A home environment in this context means the property where inhabitants lead their private life. For this thesis, the following properties of home environments were identified, which alleviate some of the challenges of general ubiquitous computing:

Limited size: Home environments are limited in size. They typically encompass only a single building, more often only parts of a building (e.g. an apartment).

Well-defined static boundaries: The boundaries of a home and its basic layout are relatively static. Homes typically do not move around, and do not change their size dynamically.

Unified administrative domain: Homes typically consist of a single administrative domain. That means that the whole home "belongs" to the same entity, which has the authority to set up the systems in the home. This may not be true if an apartment is being sublet, or if multiple parties share an apartment. But since all parties are in very close contact, and presumably have contractual relations with each other, it can be assumed that a common administration of the home domain can be arranged. If this proves to be impossible, then the area can be split into multiple home domains to reach a unified administrative domain for each home domain.

Limited number of inhabitants: Homes typically have a limited number of inhabitants who regularly frequent the premises. There are also guests who may only visit the home once, but they are limited in number. The inhabitants of the home can be assumed to have a relationship to the administrator of the home domain, either personal or by contractual agreement. This is different to for example a railway station, where large numbers of unknown people frequent the premises, and where the set of users is very dynamic. Also, in a railway station the occupants of the station typically do not have ties to the administrator of the domain that are as close as in a home environment.

Constant availability of network connectivity: Because of the limited scale of home environments and recent advances in networking technology, it can be assumed that network connectivity can be made available easily throughout the home. Structured network wiring throughout homes are typically found in professional domains like office buildings. But it is becoming more and more common in home environments. Also technologies to use phone lines or power lines as network connections (homePNA [HOMEPNA 2003], digital powerline e.g. [DIGITAL POWERLINE] or home automation systems like X10 [X10 2003] to name a few). Even if wired networking is not feasible, advances in low-cost wireless technology like wireless LAN (IEEE 802.11x) or Bluetooth have made ubiquitous wireless network connectivity available to the mass market. In these settings, home-internal network communications are typically cost neutral, i.e. they are not billed by transfer volume.

Highly augmentable: Home environments are limited in size, but they are frequently occupied. So it is worthwhile to highly augment them. They provide a tightly controlled environment where environmental conditions are predictable and vandalism typically is no problem. Also the responsibilities for the augmentations are clear because of the unified administration domain, and no billing issues for the use of the augmentations arise, because they typically are owned by the people who use them. So it is possible to embed a wealth of intelligence into the environment. The intelligence can reside more in the home, and does not have to be carried around in the form of personal computing devices like PDAs or wearable computers.

2.1.3 CONTEXT

In ubiquitous computing, context information plays an important role. There are several different definitions of context, e.g. [DEY 1999], [CHEN 2000]. The definition presented here is largely based on the definition in [BAUER ET AL 2002]

Definition: Context

Context is information that can be utilized to characterize the situation of an entity. Entities are people, places and objects that are considered relevant for the behavior of applications. Entities themselves are also considered to be part of their context.

On that basis it can be distinguished between two types of context: primary context and secondary context. The primary context of an entity consists of identity, location and time. Secondary context can then be derived by taking further aspects of a given entity into account, e.g. the current state / activity of an entity. Thus, the primary context can serve as an index to identify entities. The secondary context can be derived from the given characteristics of such an entity. In general, a combination of either identity and time or location and time is needed to access context information in a meaningful way. However, time can also be defined implicitly as the current point in time. Using the primary context as an index corresponds very closely to the way we intuitively structure the world around us. For example, the spatial structure of the world around us provides an important basis for the way we think about and act within our environment. Also, in reality information is often presented that way, e.g. on signposts. A context model is a uniform model that includes the parts of the context that are relevant for a set of applications.

2.1.4 SPATIAL MODELING

We have seen in Section 1.2 that it is useful to model spatial information in a standardized fashion and to share it between applications. As we have seen in the previous Section, spatial information can also serve as a primary index for context information. Now the question arises how spatial information can be modeled uniformly.

A variety of different location models exist, which can be classified as topographical, topological, or hybrid models. Topographical models use geometry to model space. They model the spatial entities as geometric shapes that are placed within a coordinate system. The relations between the entities, e.g. which entities are next to each other, are implicitly defined by their location. Topological models describe the relations between spatial objects explicitly without localizing them in a coordinate system. Hybrid models combine the localization of the spatial entities within a coordinate system with the explicit modeling of their relations. The complexity of the models differs widely, e.g. regarding the level of detail and the available functionality. Common tasks are querying for objects within a distinct area as well as determining the nearest object with respect to a given position. Hence, the context model

requires a spatial structure. Since context models take the spatial structure of the real world as their basis, they can also be called world models.

In Section 2.1.3, we have seen that spatial information is a type of primary context that can serve as an index to identify entities in the context. It is possible to structure a context model by the dimension of space. A spatial model can be enriched to contain further context information. So in fact an enhanced location model can be regarded as a context model. When the term “context model” is used in this work, it means an enhanced spatial model that also contains other context information.

2.2 PROJECTS

In this chapter, projects are described that form the framework this thesis is built upon.

2.2.1 THE AWARE HOME

The Aware Home Research Initiative (AHRI) [KIDD 1999] is an interdisciplinary research endeavor at the Georgia Institute of Technology in Atlanta, Georgia, USA. Its research agenda states: “How can a home enhance the quality of life of the residents if the home is aware of the activities of the occupants?”. Within this general question, the research agenda spans four complimentary themes:

Develop the technological building blocks: The construction of the necessary building blocks to allow the house to sense what its inhabitants are doing, and to act upon it.

Design the interactive experience: Design the way the inhabitants interact with the house. This means exploring applications to support the inhabitants, and designing the user interfaces used for interaction with these applications.

Explore Software construction: This addresses the challenges of creating software in the home environment using the available resources.

Consider the socio-legal implications of awareness: The collection of information about the activities of the inhabitants of the home creates a variety of concerns about privacy and autonomy.

The heart of the Aware Home Research Initiative is the Residential Laboratory of the Georgia Tech Broadband Institute, also called the Aware Home. It is a three-story, 5040 square foot (468 square meters) home that was completed in May 2000 and serves as a living laboratory. From the outside, (see Figure 2-1 on page 21) it looks like a normal home.

FIGURE 2-1: External view of the Aware Home

As shown in Figure 2-2 on page 22, the first and the second floor are laid out as normal apartments, complete with kitchen, living room, bathroom, bedrooms and walk-in closets.

FIGURE 2-2: **Layout of the Aware Home**

The basement houses meeting facilities, hardware labs and machine rooms (see Figure 2-3 on page 22).

FIGURE 2-3: **The computer vision cluster in the basement of the Aware Home**

The Aware Home is a highly augmented environment. It features multiple-angle camera coverage to support the image understanding research, a microphone and speaker infrastructure, display devices, various home-automation actuators and a

variety of other sensors. It also features an RFID location tracking system (see Figure 2-4 on page 23).

FIGURE 2-4: **RFID antenna mat in the Aware Home**

The Aware Home serves as a development platform and test bed for ubiquitous computing applications in home environments. The fact that the house looks like a normal home makes it easier to conduct user studies, and to evaluate the applications in their "natural" environment.

Experiences from the Aware Home served as the inspiration and the Aware Home served as a test bed for this thesis. The requirements derived from this environment were the foundation for the design of the infrastructure developed in this thesis.

2.2.2 NEXUS

Nexus is a project of the University of Stuttgart [HOHL ET AL. 1999]. It is funded by the Deutsche Forschungsgemeinschaft through the Sonderforschungsbereich 627, "Umgebungsmodelle für mobile kontextbezogene Systeme" (Spatial Models for Context Aware Systems"). Nexus focuses on the definition and standardization of world models for mobile and ubiquitous computing applications. In this context, it deals with communication, information management, methods for model representation and sensor data integration. More precisely, Nexus deals with

- *Modeling- and Extensibility Concepts:* What should world models look like, and how can they be kept extensible
- *Federated Model Management:* The integration of heterogeneous world models through the use of federation concepts
- *Integration of Time Concepts:* How can time be represented in the model
- *Generic Integration of Sensor Data:* What sensor data is useful for the applications, and how can it be integrated into the model

- *Consistency Concepts:* How can the information in the model be kept consistent
- *Model-based Communication:* Development of communication concepts based on world models
- *Security and Privacy Concepts:* Exploration of the security and privacy concerns of global world models
- *Automatic Acquisition of Model Data:* How can model data be acquired and the model be created automatically
- *Methods for Presentation of and Interaction with the Model:* What can the interaction with the world model look like, e.g. on mobile devices
- *Social Aspects:* Exploration of acceptability issues of global world models

To cover this breadth of topics, Nexus is an interdisciplinary research effort. It includes computer science, geographic information systems, databases and other technical fields, but also social sciences.

Nexus defines a standardized but extensible world model and an infrastructure framework to manage the information of the world model. Nexus uses a comparatively heavyweight infrastructure based approach. The architecture is designed with special regard to scalability, e.g. by using federation concepts. The comparatively heavyweight approach makes deployment of Nexus comparatively complex. However Nexus is flexible enough to accommodate specialized infrastructures.

Although indoor ubicomp applications in home environments are in the scope of the Nexus project, so far it has mainly focused on mobile computing applications. Nexus will be covered in great detail in Section 7.2.

2.2.3 THE NEXUS LOCATION SERVICE

The Nexus Location Service [LEONHARDI & ROTHERMEL 2001] is a Nexus service that collects and manages the location information of mobile entities independent from a special application or sensor system. It is intended to be available globally. Therefore it is designed with much attention towards scalability. The Location Service stores the location information in a distributed architecture. Because of the dynamic nature of location information, it is stored in main memory databases. The Location Service utilizes highly efficient dead-reckoning protocols to transmit the location information.

The Location Service supports a variety of queries, i.e. queries for “All persons at a specific location” or “Location of a specific person”. It also supports the notion of events, like persons entering or leaving an area.

The Location Service is part of the Nexus Platform and is discussed in more detail in Section 7.3.1.

2.2.4 NEXUS INDOOR SPATIAL SERVER

In 2002, a diploma thesis [GRZAN 2002] developed a specialized indoor Spatial Model Server for the Nexus platform to support location based indoor applications. This Spatial Model Server (SpaSe) only supported a very limited subset of the Nexus world model, and did use a main memory representation to store the world model.

The main challenge for practical deployment of this system is that it does not provide any extensibility mechanisms, which makes it impossible to tailor it to specific indoor ubiquitous computing usage scenarios.

2.3 RELATED WORK

This section surveys some related work in the realm of spatial modeling. The list of related work surveyed here is not comprehensive. There are many other projects that attempt to create infrastructures for ubiquitous computing, and many more which work on applications in this realm. Most of them have to deal with spatial information in one way or the other. In many of these projects, the spatial models remain non-explicit and / or very domain specific (i.e. the location of users in the QoSDream system [COULORIS ET AL], which only deals with the location of users, without providing generic spatial modeling facilities).

The following sections survey the spatial modeling aspects of some projects which to some degree provide generic spatial modeling facilities and influenced this thesis.

2.3.1 EASY LIVING

Easy living is a project of Microsoft Research by Steve Shaver, Barry Brumit et al. [BRUMIT ET AL 2000] It aims at creating an architecture and technologies for Ubiquitous Computing. It is focused on an indoor environment – specifically on a living room. To a great degree, it is concerned with user interfacing.
The main foci of the project are:

- *Ubicomp UI concepts and applications:* Easy Living aims to explore new UI concepts and applications in the ubicomp realm.
- *Distributed systems architecture:* Easy living is based on its own Middleware called InConcert
- *Sensing :* EasyLiving explores computer vision technologies for location sensing

- *World modeling:* Easy Living uses the EasyLiving Geometric Model (EZLGM)

So Easy Living uses an explicit spatial model to store its spatial information. This model is stored within a standard relational database system without any spatial extensions. Since the scope of EasyLiving is only the living room, the spatial model is limited in its flexibility. It does not support mechanisms for extension of the spatial model. The model is a topographic model and is based on geometric relations between the objects. Figure 2-5 on page 26 shows an example of the contents of an EasyLiving Geometric Model.

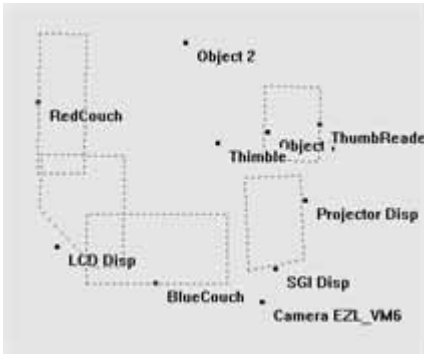


FIGURE 2-5: **Example of contents of the EasyLiving Geometric Model**

2.3.2 CANU

The CANU project [CANU 2003] focuses on the challenges posed by mobile ad-hoc networks (MANETs) in the realm of ubiquitous computing. Its major fields of research are:

- *Information Dissemination:* Dissemination of information in MANETs by using epidemic algorithms [KHELIL ET. AL 2002]
- *Routing:* New routing schemes for dynamic MANETs which exploit the movement characteristics of users in the MANET [BECKER ET AL 2001]
- *Application architectures:* Software architectures to cope with the constant change in MANET environments. [BECKER ET. AL 2002]

CANU defines a topologic spatial model that is tailored especially to the requirements in MANETs [BAUER ET AL 2002]. The CANU spatial model served as an inspiration for the topologic aspects of the spatial model developed in this thesis.

2.4 SUMMARY

There are already a variety of projects which to some degree deal with spatial modeling. But none of these projects provide a lightweight, flexible and extensible infrastructure that is tailored to the requirements of ubiquitous computing applications in home environments.

CHAPTER 3

REQUIREMENTS

Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher.

(Vollkommenheit entsteht offensichtlich nicht dann, wenn man nichts mehr hinzuzufügen hat, sondern wenn man nichts mehr wegnehmen kann.)
– Antoine de Saint-Exupéry, Terres des Hommes

Ubiquitous computing in home environments poses a variety of requirements to the infrastructure that supports it. For almost all areas of software development, the elicitation of requirements is a crucial task. This also holds true for the creation of new software infrastructures like the Aware Home Spatial Service, which is developed in this thesis.

In this thesis, a scenario-based approach for the elicitation of requirements is chosen that is based on typical usage scenarios. This approach is similar to the approach used to elicit the requirements for the world model of the Nexus project, as described in [MEßMER 2000] and [USE CASE 2000]. In Section 3.1, the scenarios that have driven the requirements for this infrastructure are presented.

There are different types of requirements for a software infrastructure that supports ubiquitous computing applications in home environments. First of all, there are the requirements that are mandated by the applications that have to be supported by the infrastructure. These requirements are elicited in Section 3.2.

Other requirements come from the (organizational or technical) environment that the infrastructure should be deployed in. For this infrastructure, the main usage scenario is the Aware Home research project. These environment-driven requirements are described in Section 3.3.

3.1 SCENARIOS

The scenarios presented here are typical usage scenarios from which the requirements for the system can be derived.

In the following sections, three scenarios are described. They all take place in a smart home. A home that can sense what its inhabitants are doing, adapt to them and support them with their tasks. All the applications described are not custom-made for the specific house. Instead, they are commercial off-the-shelf software that you will be able to buy and then just load into the house system.

3.1.1 THE SMART INTERCOM

In the smart home, there is a smart communication system – the Smart Intercom. Being voice activated, it allows the inhabitants to reach every other person in the house – regardless of where he or she is located. So for example while being in the basement, it is possible to say ”House, I want to talk to Thomas”. The house then would localize Thomas, check if it is acceptable to contact him at the moment or if he is occupied otherwise, and then create an audio connection to him using the closest available audio interface device. This could be the phone on his desk or the speakers and microphones in the room.

3.1.2 THE SMART DOORBELL

Now a smart doorbell should be added to the house systems. Depending on whom a visitor is here to see, the Smart Bell finds the respective person and notifies her that a visitor is waiting at the door. Then it routes a video feed from the entrance area to the wall display closest to that position. Now it can be decided whether the visitor should be let in. It is also possible to identify the visitor and to register him with the intercom system and other systems of the house.

3.1.3 SMART ALARM SYSTEM

Also, a smart alarm system should be added to the house systems. Because the house knows the identities and the locations of its inhabitants, it can continuously monitor whether unauthorized persons are in the house and notify the authorities appropriately. Then it would provide the authorities with information about the situation within your house – e.g. the layout of your house, where the inhabitants of the house are and where the intruders are located.

3.2 APPLICATION-DRIVEN REQUIREMENTS

INFORMATION MODEL: The applications in the scenarios need certain information to fulfill their tasks. Mobile objects, such as users with their identity and position, and stationary objects, such as furniture or input/output devices, are relevant for context-aware applications. These objects have to be managed with respect to their spatial environment, e.g. the floor plan of a smart house or the layout of a city center. Additional attributes of these objects, such as user preferences or a functionality description, can be used by applications. The information model should be easily extensible, since new applications are likely to require additional or specialized objects.

INFORMATION ACCESS: Applications access the information in the information model through queries, predicates and functions operating on the information model. Queries are used to access objects via their identity or their current location, e.g. a range or a neighborhood. The interpretation of a range or a neighborhood requires some spatial knowledge which has to be provided by the underlying information model. Spatial predicates are constantly evaluated and signal distinct correlations between mobile and stationary objects to an application, e.g. a user entering a room or a certain user meeting another user. Applications can modify information in the information model via updates.

CONSISTENCY: The information about people, house and resources, e.g. as gathered through sensor systems in the infrastructure, has to be maintained and kept consis-

tent among all applications. If real world objects or resources are moved or removed, the information has to be updated.

ABSTRACTION FROM RESOURCES: The variety of possible information sources and services, such as sensors, actuators, and user interfaces, should be transparent to applications. A layer of abstraction should be provided to facilitate the easy change of resources, such as upgrading a positioning system. Newly integrated resources should be made available to applications.

INTEROPERABILITY: One of the main requirements for the infrastructure is interoperability in three dimensions. The first dimension concerns interoperability between resources, e.g. the audio speakers should be able to work with the TV to display the video feed of the Smart Doorbell. The second dimension concerns applications that communicate using information or resources: e.g. the Smart Doorbell inserts a new ”person” into the house and the Smart Intercom should be able to use this information. Finally, the third dimension concerns the interoperability between applications: e.g. the Smart Doorbell notifies the Smart Intercom that it has to use the audio speakers to signal into the room.

3.3 ENVIRONMENT- DRIVEN REQUIREMENTS

Requirements for the software infrastructure not only come from the application domain that should be supported. Software infrastructures always have to fit into a technological and organizational environment. And these environments also impose requirements onto the infrastructure.

PLATFORMS AND PROGRAMMING LANGUAGE: A very fundamental question regarding the technological environment of an infrastructure is the question which platforms and programming languages have to be supported. The best infrastructure is useless when it only works with c++ applications on PCs, whereas the applications that want to use it run Java on handheld computers.

In the Aware Home, the main programming language for new applications is Java on Windows 98/2000/XP and Linux 2.4.x. But many of the sensing and signal processing applications are still written in C and C++. This is necessary because these programming languages allow direct access to the hardware and also provide better performance. Apart from PCs, handheld devices are in use. They mostly run Microsoft’s Pocket PC operating system and are usually programmed in Java.

This environment leads to the following requirement: The software infrastructure should support the PC platforms under Windows 98/2000/XP and Linux, and also PocketPC platforms. It should be easily accessible for Java applications. C and C++ applications should also be able to use it, although ease of use has lower priority because of the smaller user base.

NETWORKING ENVIRONMENT: The networking environment in the Aware Home is mainly based on Ethernet. There are network connections in every room. Furthermore, the whole house is covered by wireless LAN access points. The network is based on TCP / IP.

This leads to the requirement that the software infrastructure should be based on TCP / IP. Intermittent network connectivity is not an issue that is of concern.

EASE OF USE: In the Aware Home, applications have been developed for quite some time now without the support of the Aware Home Spatial Service. This means that there is a huge amount of know-how available on how to develop this kind of applications. A new infrastructure now brings change, and requires people to learn doing things in new ways. So in order to facilitate wide spread use of the AHSS, using the new infrastructure should be as easy as possible. Also, one request from the application developers was to hide as much of the characteristics of the network as possible (e.g. network addresses and port numbers), in order to ease application development.

LIMITED BUDGET: The budget of this project was very limited. So it is not possible to use expensive third-party software. Also this project did not have access to academic licensing programs of large vendors like Microsoft or IBM.

FLEXIBILITY: The research environment in the Aware Home is very dynamic. New applications are developed frequently, and the requirements for infrastructures and spatial models change frequently. So a new infrastructure should not obstruct this development by pressing the application developers into a too inflexible framework. Direct interaction and agreement between the developers who are all in one lab is preferable to inflexible formal standards that produce a high administrative overhead to maintain them over the frequent changes.

CHAPTER 4

SPECIFICATION AND DESIGN

Software design is hard, and we need all the help we can get.
– Bjarne Stroustrup

This Section gives an overview of elements of the specification and design of the system. Section 4.1 describes the core of the AHSS, the spatial model. Section 4.2 and its subsections then introduce the elements of the system architecture, the spatial database, the spatial server and the client services.

4.1 SPATIAL MODEL

The spatial model is the heart of the infrastructure. It determines how spatial information is modeled within the infrastructure. As explained in Section 2.1.4, a spatial model can serve as the foundation for a context model.

The AHSS spatial model defines and implements a standardized way to model context information structured along the dimension of space. This spatial model is topological, very similar to the one that has been proposed for the CANU system [BAUER ET AL 2002] that was adapted and extended by topographic aspects.

As shown in Figure 4-1 on page 36, the spatial model represents context information from the real world. All applications can use the same spatial model. The model consists of a graph, with locations as vertices and relations as edges. Locations describe relevant spatial objects. These could be rooms, furniture or even people. An AHSS Location has an ID that is unique within the spatial model, a type that is associated with it, some standard attributes and an optional spatial attribute. This spatial attribute can be a point, a line, a polygon or any other geometry type specified in the OpenGIS standard [OGC 99-049]. It represents the topographic aspect of the model. For example, it is possible to attach the shape of a room (i.e. its geometry) to the Location representing it. Spatial attributes provide a variety of spatial operations like union, intersection etc. Furthermore, Locations are extensible. Applications can add arbitrary attributes with application-specific semantics.

Locations are connected by relations. Relations allow the modeling of spatial relations explicitly. They have a type and a weight that can be used to model distances. These explicit spatial relations are necessary because in the real world, geometric distances are not the only relevant spatial relations. For instance, two rooms might be very close from a geometric point of view, but if there is a wall between them, or

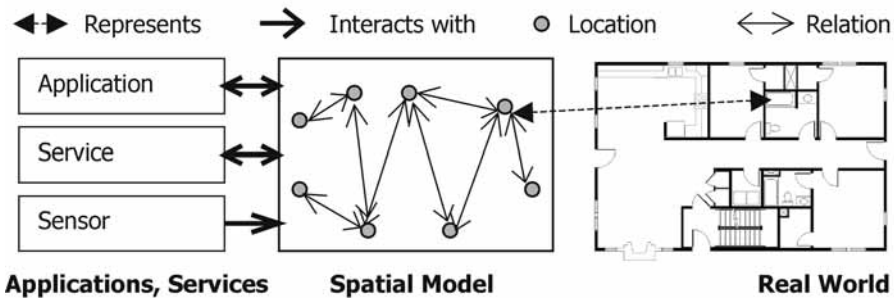


FIGURE 4-1: **AHSS conceptual view**

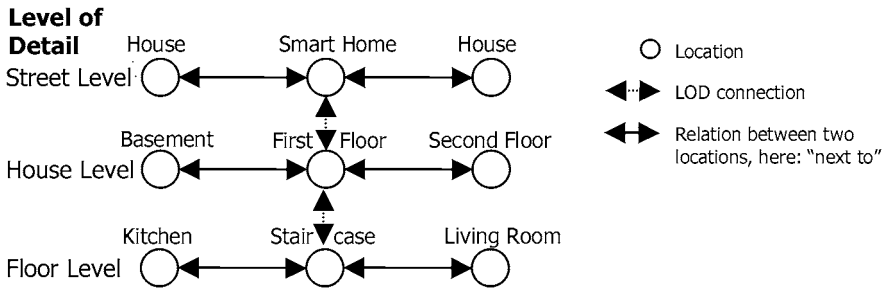


FIGURE 4-2: Levels of Detail in the AHSS spatial model

a door that can only be used in case of an emergency, they may in fact be very far apart.

Also, locations are organized within levels of detail. Levels of detail structure the spatial model hierarchically. This is important because not every application needs to deal with the model on the same level of detail. One application might only need the coordinates of a house, while another application might need more detailed information, like for example the layout of the rooms in the house. Figure 4-2 on page 37 illustrates this. On every Location, it is possible to generalize (get the equivalent Location on a lower LOD) or specialize (get the gateway Location on a higher LOD). For example, generalizing the kitchen would yield the first floor, specializing the first floor could yield the staircase.

AHSS also provides a basic event service. It is possible to specify events on Locations. These events are defined by a name and a list of subscribers. For example, an application is able to attach an event called “personEnters” to a “room” Location. Then another application can “fire” the event of this name, and all the subscribers are notified. An application that triggers an event can also specify payload that should be delivered to all the subscribers of this event. That way it is possible to notify all the subscribers that an event has occurred, and to supply additional information. In the previous example, this could be information about who has entered the room.

The semantics of AHSS events – like the semantics of many attributes of Locations – is entirely application specific and is not defined by the AHSS spatial model. So it is possible to define an event called “personEnters”. The same event could also be called “greenRabbit”, as long as all applications that subscribe to it and trigger it agree on its semantics. Also, the type of a Location could be “Room”, but it could also be “Dragon” – AHSS provides the basic primitives to model spatial and contextual information and provides an infrastructure to share and manage this model. It does not prescribe a specific way of modeling the world. So the application developers who use AHSS are free to tailor the way they model the world to their own needs, and can evolve their modeling standards as they see fit.

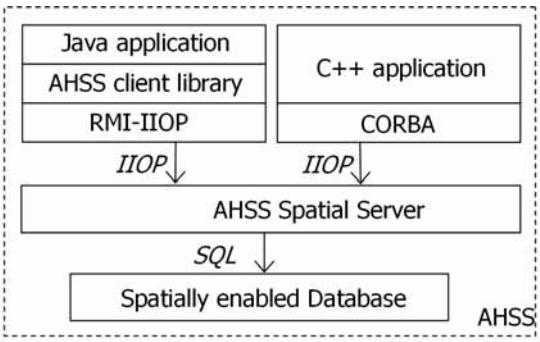


FIGURE 4-3: **Architecture of the AHSS**

4.2 ARCHITECTURAL OVERVIEW

If multiple applications are concurrently using the same spatial model, the model has to be kept consistent among those applications. The AHSS uses a centralized approach to accomplish this – the spatial server. This central spatial server stores the current spatial model. To improve performance, applications perform local caching of the spatial model. This centralized architecture allows us to keep the architecture of the system relatively simple. Figure 4-3 on page 38 shows an overview of the architecture. However, it should be noted that a centralized design is not required by the spatial model. Sharing and consistency of the model could also be provided by a decentralized infrastructure.

AHSS uses a 3-tier architecture approach. The spatially enabled database, as described in Section 4.2.1, is used to store the spatial information. The AHSS Spatial Server, which is explained in more detail in Section 4.2.2, is the central server that provides the AHSS services to the clients, while using the spatially enabled database. The client applications use the services of the spatial server and are supported by the AHSS client services library as described in Section 4.2.3.

4.2.1 SPATIAL DATABASE

The spatial server has to store a potentially large amount of information that has a spatial component in a persistent way. Handling large amounts of data and storing it persistently in an efficient and secure manner poses serious challenges. It therefore is advisable to use specialized databases to store this data.

The EasyLiving project [BRUMIT ET AL 2000] uses a regular relational database system to store its spatial model. The problem with regular relational databases is that

they do not natively support spatial data types. This is not necessarily a problem when the spatial information only has to be stored within the database without doing any queries based on it. But when querying a spatial model, spatial queries are very common. An example for a spatial query is a query where a spatial object (e.g. a Point) is provided, and the results of the query are all the spatial objects in the database (e.g. Polygons) which overlap with it.

A regular database which does not support spatial data types can still store spatial information, e.g. in form of a string or a binary object. But the database does not understand its semantics. So at best, the databases will be able to query this data based on identity of the stored representation. This means that e.g. if the spatial information is stored as its string representation, then it would be possible to use the identity of these strings to query for specific spatial objects. But the same spatial object (e.g. a Polygon) can have a variety of string representations. So it is not certain that a query for all database entries whose “position” attribute is a certain polygon would in fact return all those entries, because the same polygon could be described by different strings. Also, advanced queries like “containment” of one shape in another would not be possible within the database without implementing this functionality within the database e.g. as stored procedures.

So using a conventional relational database without support for spatial data types would necessitate the manual handling of all issues related to spatial queries. So all the advanced features of databases like indexing for faster retrieval would have to be replicated manually. Manual handling of all these issues can lead to performance issues and increased implementation complexity. Fortunately, there is a variety of databases available which directly support spatial data types. By using them, it is possible to get support for the spatial queries that are required in AHSS.

The selection of the concrete spatial database system is discussed in Section 5.2.

4.2.2 SPATIAL SERVER

The central server stores the current spatial model. The applications then synchronize their own spatial model with the model of the server to ensure consistency. Other systems whose main goal is scalability, e.g. Nexus, distinguish between static spatial information (that does not change regularly, like for example the room layout) and dynamic spatial information (that changes regularly, like for example the position of tracked people). Because AHSS is targeted at small to mid-size usage scenarios where the amount of dynamic information is small enough to be handled by the database, it does not make this distinction. This makes it possible to keep the system simpler and more lightweight.

The server interface is procedural. Interactions with the server are done by calling procedures.

The interface of the spatial server offers the following functionalities:

- *Maintain locations:* It is possible to create new locations in the spatial server, to update existing locations and to delete existing locations.
- *Query for locations by ID:* Locations can be queried for by location ID
- *Query for adjacent location:* The spatial server can be queried for locations that are explicitly related to a specified location and whose relation satisfies the specified restrictions. So it is possible to traverse the explicit relations on the server.
- *Query for locations by spatial query:* It is possible to query for all locations that overlap with a specified geometry
- *Query by location attributes:* Locations can be queried for by specifying restrictions for the attributes of the locations
- *Subscribe/unsubscribe to an event:* Applications can subscribe and unsubscribe to events that are defined on locations. Events are identified by a name.
- *Fire event:* Applications can fire events that are defined on a location. They have to specify the name of the event they want to fire, and can optionally specify "payload" for the event. Then all the applications that have subscribed to that event are notified, and the payload of the event is delivered to them.

4.2.3 CLIENT SERVICES

Ease of use for the application developer was a very important goal for this infrastructure. So in order to support the application developer, the Aware Home Spatial Service Client Services were created. Right now, they are only implemented for Java applications. They are a client library which encapsulates most of the specifics of the interaction with the Spatial Server. By using the Client Services, the application developer can create local Location objects, which manage their interaction with the server on their own. The application developer does not even have to deal with network addresses or the specifics of how to reach the server.

Some example code is the best way to illustrate how this works.

```
[1]  // Do some initializations
[2]  SpamodConnection.init("iiop://
     localhost:1050","SpatialServer");
[3]  SpatialServerRMIInterface spa=SpamodConnection.getServer();
[4]  SpatialGeometryFactory fact=new SpatialGeometryFactory();
[5]  // Create a group for the locations
[6]  ClientLocationGroup location_group=new
     ClientLocationGroup();
[7]  // Create the house
[8]  ClientLocation house=new ClientLocation(TYPE_HOUSE,"The
     house",fact.makeRectangle(940,0,0,1500),LOD_STREET,FOR_ATL,n
     ull,location_group);
[9]  // create each floor
[10] ClientLocation first=new ClientLocation(TYPE_FLOOR,"The
     first
```

```
     floor",fact.makeRectangle(940,0,0,1500),LOD_HOUSE,FOR_1ST,
     house,location_group);
[11] ClientLocation second=new ClientLocation(TYPE_FLOOR,"The
     second
     floor",fact.makeRectangle(940,0,0,1500),LOD_HOUSE,FOR_2ND,
     house,location_group);
[12] ClientLocation base=new ClientLocation(TYPE_FLOOR,"The
     basement",fact.makeRectangle(940,0,0,1500),LOD_HOUSE,FOR_BAS
     E, house,location_group);
[13] // link them
[14] house.setDownlink(first);
[15] first.addBidirectionalRelation(base,REL_REACH,300d);
[16] first.addBidirectionalRelation(second,REL_REACH,300d);
[17] house.updateOnServer();
[18] // Query
[19] Location[] quer=spa.geometryQuery(
     factory.makePoint2d(1,1),null);
[20] for (int i=0;i<quer.length;i++) {
[21] System.out.println("\nResult of spatial
     Query:"+quer[i].toString());       }
```

The initializations beginning at line 1 are needed to tell the client services the RMI address of the Spatial Server and to get a reference to it. It also creates a new factory for geometric objects. Beginning from line 5, new Locations are created. For easier handling, these Locations can be aggregated into groups. So in line 6, a location group is instantiated. Then various locations are instantiated and added to that group. While instantiating these locations, the geometry factory instantiated in line 4 is used. Then, beginning from line 14, the Relations between these Locations are set. Line 14 sets a LOD relation (i.e. a specialize/generalize relation). Lines 15 and 16 then set Relations of the type “REL_REACH” (reachability) with a weight (distance) of 300. Line 17 then updates the data of the locations in the location group on the Spatial Server. From that point, all applications that access the Spatial Server have access to these locations. Line 19 then queries the Spatial Server for all Locations that have a Geometry that overlaps with the point (1,1). Line 20ff then iterates through the results and prints them.

CHAPTER 5

IMPLEMENTATION ISSUES

"If you build it, they will come."
– Movie "Field of dreams"

5.1 MIDDLEWARE

The Aware Home Spatial Service is a distributed system. Client applications (or the Client Services part of AHSS in case of Java applications) have to communicate with the AHSS spatial server. One fundamental question is how this communication should be implemented. While socket programming can be used to implement the interoperability protocols manually, this is not a practical option. The interactions between client and server can be organized as simple remote procedure calls, and do not have any special requirements for the interaction. So it is reasonable to use existing middleware standards to implement these interactions. There are several available commercial and proven middleware standards. The following sections give a quick and greatly simplified overview of the available standards.

5.1.1 CORBA

CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate. CORBA is maintained by the Object Management Group (OMG). The members of the OMG are over 700 organizations which represent major software companies and research institutions.

The basic CORBA paradigm is that clients request a service (i.e. a method execution) from a remote object and receive the results of this execution. CORBA is a specification that basically specifies the interfaces a client has to implement in order to use remote services, and a server has to implement in order to be able to satisfy remote service requests.

CORBA specifies the Object Request Broker (ORB) which transports the requests to the server and transmits the results back to the client. The client uses an object reference which is typed by an interface (which is described in IDL, the interface definition language). The ORB then uses this reference to locate the server and to process the requests. This creates location transparency – the object that provides the service can be located on the same machine as the client, but also on another machine on the network.

The CORBA specification provides mappings for many languages. Also the requests are language independent. That means that a request that was generated by a C++ program can be satisfied by a server that e.g. is written in Java. As described in Section 3.3, this is very important, because at the Aware Home, there are various platforms in use. Since CORBA 2.0, CORBA also specifies an interoperability protocol called IIOP, which allows ORBs from different manufacturers to interoperate. IIOP is a binary format which works very efficiently.

CORBA uses stubs and skeletons which can be generated from the IDL interface descriptions by special generators that are part of CORBA. This adds a lot of con-

venience, because the stubs can make a remote call look very similar to a local call for the application programmer.

CORBA is a very powerful specification for which implementations are available for a wide variety of platforms. This necessitates a comparatively high degree of complexity of the available CORBA implementations. So using CORBA requires a higher amount of expertise as e.g. RMI.

5.1.2 SOAP

SOAP is a standard that was proposed to the W3C by UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, Microsoft, and SAP. It (among other things) defines a way how RPCs (or more specifically Remote Method Invocations) can be encoded in XML. The requests are then transmitted using standard protocols like HTTP. As CORBA, SOAP is platform and language independent. This means that a C client could use services offered by a Java server. SOAP is one of the core technologies of Microsoft's .NET platform and is becoming more and more popular. At the time this thesis was created, SOAP is available for Java in multiple implementations, the major one being the SOAP implementation by Apache.

The implementation of services with Apache SOAP is relatively easy. It only requires to implement the class that implements the service and to specify deployment information in an XML document. The rest is handled by Apache SOAP. But calls from a client to the server are (compared to other middleware like CORBA or RMI) relatively complicated. Instead of naturally calling a method on a remote object, it is necessary to manually build complex request objects and handle much of the process of sending the request manually. There are also some unresolved compatibility issues with various SOAP implementations.

Also, SOAP transmits the requests and the responses as XML. This introduces a lot of overhead into the transmission process, which can create performance problems.

5.1.3 RMI

RMI is the middleware that is integrated into Java since JDK 1.1. It supports (it may not come as a surprise) remote method invocation between separate Java Virtual Machines which can be located anywhere on the network. Java RMI is a very elegant solution. It integrates almost seamlessly into Java, and there is almost no difference between the syntax of a local and a remote call. Also, RMI interfaces do not have to be specified in a specialized language like IDL, instead they are specified as regular Java interfaces. RMI is also interoperable between all platforms Java runs on. Unfortunately, it is not very interoperable between different languages, because it is only available for Java. RMI uses its own interoperability protocol called JRMP

to transfer the requests over the network, which can in turn use a variety of transport protocols (TCP, UDP etc).

5.1.4 RMI/IIOP

RMI/IIOP is a technology developed by Sun and IBM, which was introduced into Java with version 1.3. It allows RMI to use IIOP as an interoperability protocol instead of the proprietary JRMP. This allows Java clients to access RMI/IIOP servers with the same ease as with regular RMI. But now it also allows non-Java clients to use a CORBA-ORB to access RMI/IIOP servers. Also, Java-clients can use RMI/IIOP to access CORBA servers. So as long as clients are also implemented in Java, they can use the very convenient RMI to access the server, while non-Java clients can still access the server through CORBA.

There are some issues, though. When using RMI over IIOP, some minor RMI features become unavailable, like remote garbage collection. Also, there has to be a description of the service interfaces in IDL, so that CORBA clients can access them. But neither of these issues is critical.

5.1.5 Conclusion

All of the surveyed middleware systems have their advantages and their disadvantages. SOAP is an interesting technology. It is relatively new, but it is heavily pushed by major players like Microsoft, and it provides the necessary functionality. Its major drawback was that at the time at which the middleware-decision had to be made, SOAP still was comparatively immature and suffered from inefficiency issues. Also, at the time of the decision, the major SOAP implementation for Java was Apache SOAP, which was very complicated and not very intuitive to use for the application programmer. For years, middleware systems have been providing much more convenience for the developers than the brand new SOAP technology as implemented by Apache SOAP did at the time of the decision. Whereas with RMI it hardly makes a difference if you call a local method or a remote method in RMI, Apache SOAP required the developer to go through a complicated process of manually generating the request that is then send over the network. SOAP also did not have enough advantages over the other systems to even the score, so SOAP was not selected.

AHSS now uses RMI/IIOP as a middleware platform. This combines the elegance and simplicity of RMI with the language independence of CORBA. It allows ease of use in the “common case” (when client and server are implemented in Java), while still providing interoperability with all important platforms.

5.2 SPATIAL DATABASE

One of the central parts of the AHSS system is the spatial database management system (DBMS) in the backend. A spatial DBMS differs from a regular relational database system in that it also supports spatial data types like points and polygons. These data types are not specified in the SQL standards that almost all current relational DBMS support by default (e.g. in SQL99). So right now, not every mainstream database system supports spatial features.

The OpenGIS consortium has created a variety of specifications on how to represent spatial information in databases and how to query it [GML 1.0] [OGC 99-049]. For mainstream commercial databases, spatial support is available either by default (e.g. Oracle 9i already supports basic spatial capabilities) or as an option (The IBM Spatial Extender adds spatial functionality to the IBM DB2 database).

Unfortunately, these commercial databases have prohibitive price levels that limit their availability to a larger user base.

One might argue that many commercial DBMS offer special academic programs that provide discounts for academic use or even waive the license fee entirely for academic users. So for example, the department of computer science at the University of Stuttgart would have been allowed to use DB2 for free. But these programs are usually only available at specific institutions and not at others. E.g. the Georgia Tech CS department was not entitled to use this IBM software. Or these programs can include huge amounts of paperwork to get them going, while requiring approval from top-levels of the institute – which is a serious disadvantage. Also, these programs limit the use of the database systems to certain academic settings – which can be a problem for the evaluation of AHSS in a multitude of settings.

Since no commercial DBMS was available without taking the risk of violating intellectual property rights of the respective manufacturer, an alternative had to be found. The DBMS had to be freely available, support spatial data, and it had to be (to a certain extent) mature. Performance and a wealth of features were less important, because of the prototype status of this AHSS implementation.

For this implementation of AHSS, PostGIS [POSTGIS 2003] was selected. It was developed by Refractions Research Inc. and is available as open source. PostGIS is an add-on to PostgreSQL, a well-known open-source object-relational database management system.

PostGIS supports the OpenGIS "Simple Features Specification for SQL" [OGC 99-049]. Although it is still in pre-1.0 status and does (by far) not reach the wealth of features of the commercial systems, it still provides the functionality needed to implement AHSS, is more lightweight than full-grown commercial DBMS like Oracle, and is available for free for everyone. The drawbacks are that it does not provide optimized performance and is also a bit tricky to deploy, because it has to be manually compiled with the PostgreSQL source.

CHAPTER 6

EXPERIENCES

More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity.
– W.A. Wulf, A Case Against the GOTO

6.1 EXAMPLE APPLICATIONS

To evaluate the concept, some demo applications have been developed. One demo application periodically reads the information acquired by the location tracking system of the Aware Home, and inserts it into an AHSS model of the Aware Home.

To do this, it connects to the Context Toolkit widget of the Signpost location system, one of the location systems of the Aware Home. Signpost is based on the RFID technology. The tracked persons wear RFID tags on their shoes. These tags are available in many different form factors and can for instance be glued to a shoe or even be tied to the shoelaces. As shown in Figure 6-1 on page 50, there are RFID mats placed in important places around the house, like for example in doorways.

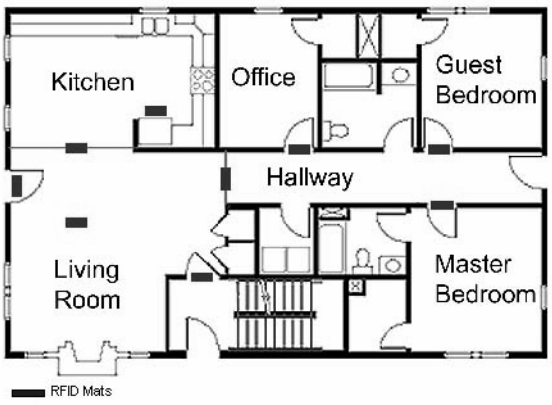


FIGURE 6-1: **Locations of the RFID mats throughout the first floor of the Aware Home**

Another application was an accuracy converter for the location system. This was necessary because some of the location system APIs in the Aware Home provide their location information as x/y/z coordinates without supporting the notion of location system accuracy. Now if the source of the location information was for example the RFID location system, then the coordinates of the tracked person would still be a single coordinate set, although the user in fact could be anywhere in the whole room. In order to address this issue, an application was created that used the spatial model of the house to translate the point-location reported by the location system API into the area the user has to be in, by using the LOD information in the model, and knowledge about the accuracy of the various location systems. Another demo application used the spatial model to locate (fictional, because not yet installed) displays in the room a person is in.

6.2 PERFORMANCE

In order to be useful for the intended usage scenarios, the Aware Home Spatial Service has to satisfy certain performance requirements. In the experiments, the AHSS was always fast enough to serve the demo applications that were built in the context of the Aware Home. But this alone is not meaningful enough. What is needed is an approximation of the performance requirements of a typical usage scenario. Is such a scenario, a variety of applications will be running in the house, and the house will be modeled at a much finer granularity then for the demo applications. Section 6.2.1 develops these requirements, and in Section 6.2.2 the results of the performance tests are presented. Section 6.2.3 summarizes the results and draws a conclusion.

6.2.1 PERFORMANCE REQUIREMENTS FOR THE USAGE SCENARIO

In order to define typical usage requirements, it is useful to base the assumptions on the usage scenario in the Aware Home. A house like the Aware Home has approximately 30 rooms. If it is assumed that in each room approximately 30 Locations are modeled, then the infrastructure has to handle approximately 1000 Locations. These are less then 0.5 square meters living area per modeled Location (based on a living area of 468 square meters for the Aware Home).

The most relevant performance characteristics are the number of possible updates/second in the model, and the number of queries/second. Updates are important, because the model is dynamic. Applications constantly update information in the model. E.g. a temperature service could continuously update the temperature of the room he serves in the model. A location service could update the position of the people it tracks. The location systems in the Aware Home have produced less than one update per second. Also, assuming that there are 5 other sensors per room, and each sensor produces one update every 10 seconds, this produces about 15 updates per second. So in summary, 20 updates per second can be assumed sufficient for the Aware Home as it is today.

Queries are used by applications to get data out of the infrastructure. This operation is the most frequent operation in the infrastructure, because it allows the applications to use the data stored in the model. The AHSS Client Service holds partial local copies of the Spatial Model and only contacts the Spatial Server when consistency with the server is required. This reduces the number of interactions with the server considerably.

When 30 applications work in the house, and each application needs to query the server 60 times per minute because the partial local copy of the model is not recent enough, this leads to 30 requests per second.

6.2.2 PERFORMANCE MEASUREMENTS

For the following performance measurements, the Spatial Server and the clients were running on a 1.13 GHZ Pentium III Laptop with 512 MB of RAM and Windows XP as an operating system. The JDK in use was the original Sun J2SE JDK 1.4.1_02.

The spatial database was deployed on an AMD Athlon XP 1700 UNIX server running Linux 2.4.21 with 768 MB of RAM. The database was accessed using a dedicated 100 Mbit Ethernet connection.

UPDATE PERFORMANCE

Figure 6-2 on page 53 shows the number of single updates per second that can be served by the AHSS spatial server, depending on the number of clients that are accessing the server. Depending on the number of clients, the Spatial Server is able to perform approximately 60 updates. This is enough to satisfy the requirements of the Aware Home as defined in 6.2.1.

It has to be considered that the hardware these tests have been performed on is not state of the art. Current server PCs provide roughly three times the performance of the test system. Also, this test used single updates, i.e. it performed one server call per update. Thus, the invocation and networking overhead is incurred once per updated location. This is a worst-case scenario. It is possible to aggregate multiple updates into one server call. In a usual usage scenario, the client updates its local copy of the spatial model, and then updates all the changed Locations on the server with only one server invocation. This has the potential to reduce the server load considerably.

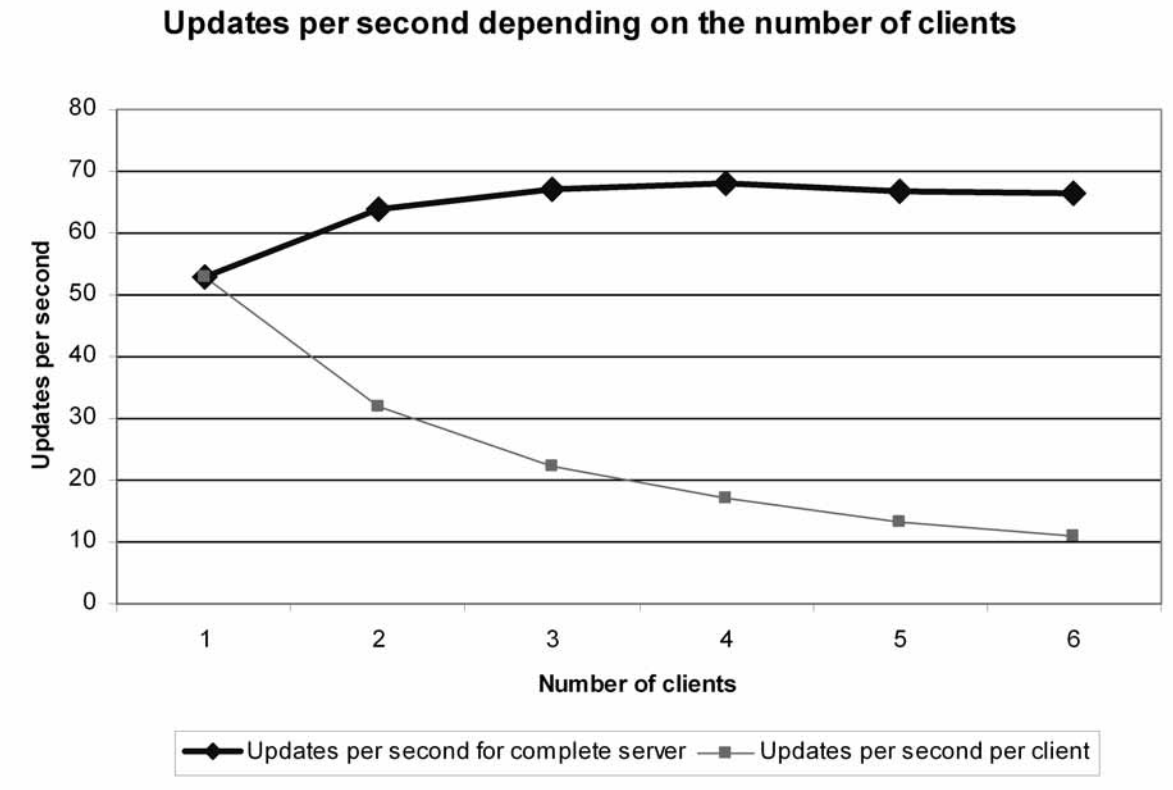


FIGURE 6-2: **Updates per second depending on the number of clients**

QUERY PERFORMANCE

To measure the query performance, 1000 Locations were stored in the Spatial Server. Then the throughput of single spatial queries was measured. The queries were designed in a way that only 1 of 1000 Locations was returned as a result. Spatial Queries were selected because they are the most complicated queries and require the most extensive server-side processing of all the queries.

With one client, the AHSS was able to process 58.2 queries/second. According to the requirements defined in Section 6.2.1, this is sufficient for the requirements of the Aware Home.

6.2.3 CONCLUSION

It can be concluded that the AHSS performs well enough for typical residential homes. Although AHSS is not optimized very much, it is able to handle the

amounts of data and the frequency of updates that is generated by home environments.

The challenge is that AHSS does not scale very well beyond a certain point. Once the capacity of a spatial server is reached, it is very difficult to scale the service. AHSS does not support the federation of multiple spatial servers, and also provides no ways to integrate an area covered by AHSS into a larger installation. So for example if every apartment of an apartment complex is served by an AHSS spatial server, then it is not easily achievable to connect these servers so that applications can easily get a unified view on the data. When AHSS is expected to scale beyond a single server, and beyond a single administrative domain, then a wide range of additional problems come up.

In order to allow AHSS to overcome these challenges, it could be extended with additional functionality. But the AHSS is targeted specifically at home environments, so it would lose its focus if the features required for federated models were added.

The approach that was chosen instead for this thesis is to integrate AHSS as part of a larger infrastructure. Nexus provides the facilities that AHSS is missing to scale beyond a single server. The Nexus architecture is also flexible enough to integrate specialized systems like AHSS. The integration is described in Chapter 7.

CHAPTER 7

INTEGRATION INTO NEXUS

Nowadays, progress is that fast, that while someone is stating that something is impossible, he is interrupted by someone who has already done it.
– Albert Einstein

7.1 General Considerations

As we have seen in the preceding sections, the AHSS supports ubiquitous computing applications in closed home environments well, but poses scalability issues for larger scenarios.

If we want to extend the scope of interoperable, context-aware applications beyond a closed environment like a single smart home, we need a common context model. It should be suitable not only for small scale indoor ubicomp applications, but for a wide range of context-aware applications. Hence the information model must be more detailed than the one presented in Section 4.1, and extensibility is very important. To achieve this goal, we have to address two major challenges: modeling standards (how can A understand what B models) and infrastructure (how can such a huge model be managed efficiently). The Nexus platform is an approach to address these challenges.

Nexus already provides elaborate concepts to deal with the challenges that AHSS is facing in larger usage scenarios. This section discusses whether and how AHSS can be integrated into Nexus, so that ubicomp applications in the home environment can still use AHSS regularly, while Nexus solves the issues that arise in large-scale scenarios.

The integration of AHSS into the Nexus federation provides various advantages. First of all, AHSS becomes part of the Nexus federated world model. Nexus applications can then use context information that is stored within the AHSS. This makes it possible to use existing Nexus applications with the AHSS without having to modify the applications. Also, Nexus applications can store their context information into the AHSS. This information can further enrich the context model which AHSS applications can access. The local AHSS interfaces are still available, so local applications can access AHSS either through the local interfaces or through the Nexus-AWQL interface.

Section 7.2 describes the Nexus Augmented World Model. Section 7.3 gives an overview of the Nexus Platform. Then in Section 7.4, the reconcilability of both approaches is discussed. Section 7.5 discusses the choices that have been made for this integration, and some implementation issues. In Section 7.7, the results of the integration are discussed.

7.2 The Augmented World Model

No single individual can model the whole world, even if the model is limited to a specific application domain like context-aware applications. But we could model the relevant parts of our smart home, another person could model the streets of Cambridge UK, and somebody else could model the Logan International Airport.

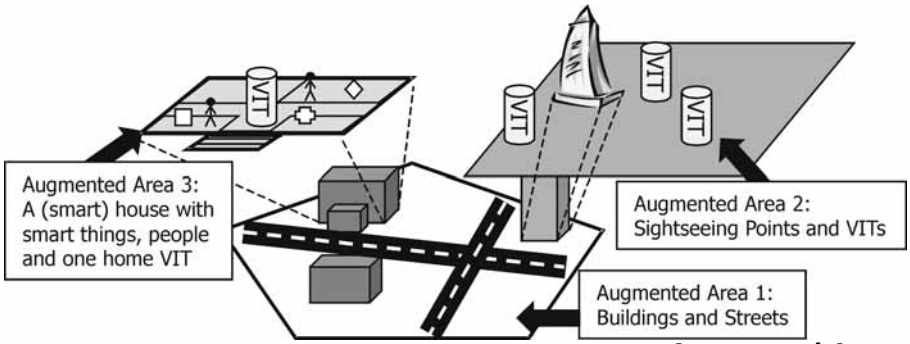


FIGURE 7-1: **Augmented Areas**

On the WWW, there is no single global information database. Instead, the web connects the numerous servers on the net. And in analogy to that, a global world model can consist of a multitude of local context models which are linked together.

To make these local models interoperable, they must rely on common standards and agreements. In the WWW, this is accomplished by the hypertext data model and the HTML standard, which is good for structuring and presenting hyper media, but says little about the content of a document (the Semantic Web [W3C 2001] research tries to overcome this). For the Nexus platform, the standard is called the Augmented World Model. As seen in Section 2.1.3, we can say that the primary context location serves as an index to secondary context. The Augmented World Model is a global object-based, common data model that defines how context information can be shared between applications and data providers. It contains real world objects like rooms, sensors, streets or persons as well as virtual objects like Virtual Information Towers (VIT) (see [LEONHARDI 1999]) or virtual Post-Its [PASCOE 1999] that are used to represent digital information in the world. The complete Augmented World Model is not stored as a single homogenous model. Instead, it is a federated, global view on all compliant local Augmented World Models. Those local models are called Augmented Areas (AA). An AA has a certain extent that describes the geographical area in which its objects reside. There is no restriction on the size or the number of objects an AA can host. AAs can overlap or model the same real world entity (see Figure 7-1 on page 55, taken from [PRESENTATION 2003]). The Augmented World Model is the federated, global view on all Augmented Areas. An application does not need to know from which AA the context information comes, and it can use data that is combined from different AAs. This federation is only possible because the AAs rely on a common data schema called the Standard Class Schema (SCS).

The SCS defines the types and attributes of the Augmented World Model objects. These are ordered in a hierarchical ”is-a” relationship (inheritance). In Figure 7-2 on page 56, you can see an excerpt from the top levels of the SCS: *dataobject* is the root that defines a unique object identifier and locator for each object called the Normalized Object Locator (NOL). Objects of the type *spatialobject* all have a geographical position and optionally an extent. Because the spatial context is the primary index and therefore is crucial to performance, it is distinguished between static and mobile objects.

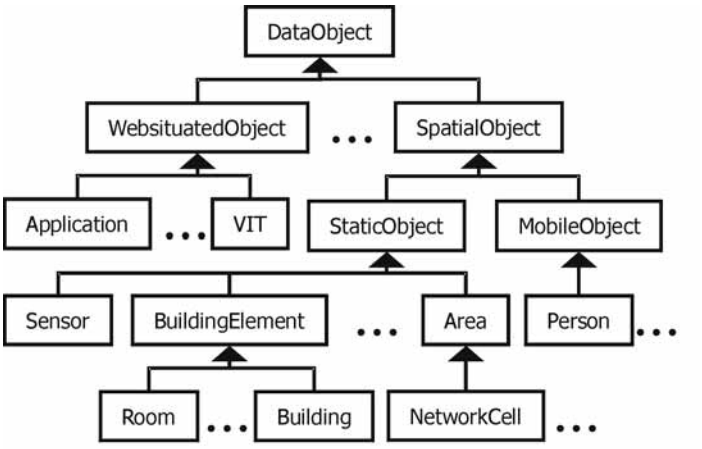


FIGURE 7-2: **Excerpt from the top part of the Standard Class**

While static objects seldom change their location and therefore can be managed in a single AA, mobile objects move around and cross the borders of AAs which leads to handovers. In total, the SCS defines about 250 classes or types for context-aware data objects; this is a form of taxonomy for context-aware applications. That model was designed by doing a use case analysis on different applications [MEßMER 2000]. Most of the attributes of the types are optional, which means that an Augmented Area does not have to provide the attribute if it has no data for it. But if an AA provides the attribute, the type and the meaning of the attribute are standardized.

It is infeasible to define all possible data objects. To provide extensibility, any data provider can define an Extended Class Schema (ECS). The classes of the ECS are derived from appropriate classes of the SCS. For example, if somebody wants to integrate LivingRoom objects that are not defined in the SCS into an AA, then she can extend the type Room with the necessary additional attributes (see Figure 7-3 on page 56). An advanced LivingRoom application can now access this information, whereas normal applications that have knowledge only of the SCS could treat the object as its parent class Room.

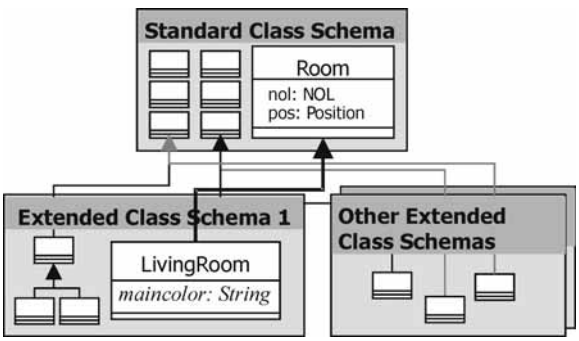


FIGURE 7-3: **Standard and Extended Class Schema**

7.3 PLATFORM ARCHITECTURE

As described above, a global context model like the Augmented World Model is feasible if independent providers can build their local models based on a flexible standard and make them available to a federation that integrates all local models into a global view. Now \an open infrastructure will be described that is able to manage this: the Nexus Platform. As you see in Figure 7-4 on page 57, it is built in three tiers: the service tier contains data and service providers for Augmented Area models. The federation tier integrates the AAs and supports value added services on the Augmented World Model. Applications are located in the application tier and use the services of the federation.

7.3.1 SPATIAL MODEL SERVERS

Spatial Model Servers (SpaSes) host Augmented Areas. They are similar to the web servers of the WWW: they provide information that is locally produced / collected for potentially global applications through a standardized interface. Instead of HTML, SpaSes transfer Augmented World Model Objects as AWML (Augmented World Modeling Language), an XML standard for serializing spatial data objects. For querying these objects, a simple spatial query language called AWQL (Augmented World Query Language) has been designed, which is used to specify the desired objects and to filter the attributes. AWQL supports spatial predicates (overlaps, inside) and nearest neighbor queries as well as data manipulation (insert, update, delete). It is represented in XML, too. Note that it is not an XML query language like XQuery – AWQL is suitable for spatial objects, not for hierarchical XML documents.

Depending on what kind of context information a SpaSe provides, the implementation of the spatial server which processes the AWQL/AWML can differ to a large degree. For static objects and large-scale spatial models, like all buildings and streets in downtown Cambridge, a full-grown database is used with a spatial extension (IBM DB2 7.1 + Spatial Extender). AWQL can be easily transformed to SQL by

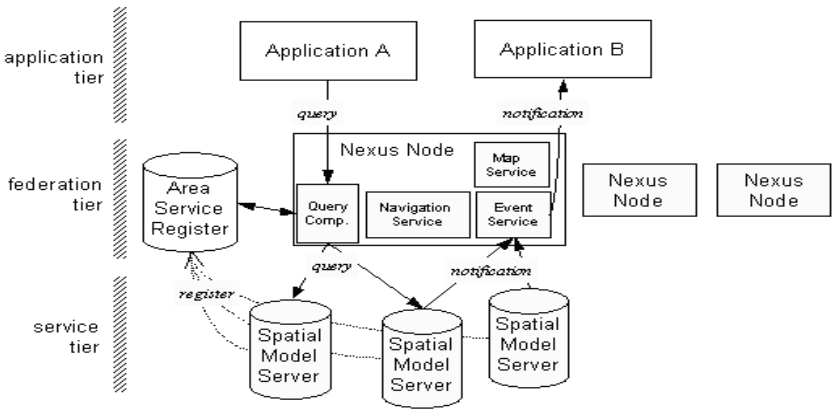


FIGURE 7-4: **Architecture of the Nexus Platform**

an XSLT style sheet. Mobile objects are managed by the Location Service. Because position data is highly dynamic and does not need to be persistent, main memory data structures are better suited for this task. The Location Service consists of hierarchically structured servers that manage objects within a certain area. When the object moves out of this area, handovers are performed.

7.3.2 NEXUS NODES

The Nexus nodes in Figure 7-4 on page 57 mediate between Nexus applications and Nexus services. They are responsible for distributing queries to different data providers and for composing the results, thus providing the Context Model for the applications. The nodes do not store persistent data, a fact that allows replicating them for load balancing.

For query distribution and service discovery, a Nexus node uses the Area Service Register (ASR). This service is a directory of the available data servers and stores the address of the server, the available object types and a polygon that encloses the positions of all spatial objects on the server. The Nexus node computes the target region and the object types of the application query and queries the ASR to find out which Spatial Model Servers cover part of the requested region and store objects of the requested types. Then the node distributes the query to the relevant data providers named by the ASR and merges their results.

7.3.3 VALUE ADDED SERVICES

In addition to the query functionality, every Nexus node supports value-added services. They use the federated context model to implement advanced services and have an interface of their own. In Figure 7-4 on page 57, you can see three different value-added services of the Nexus platform:

The Event Service monitors basic events and combines them into more complex, spatial events like ”somebody has entered the living room”, ”I am close to a shoe store” or ”two of my friends meet”. Applications can register for these events with certain predicates and receive a notification if the specified event occurs.

The Map Service gets an AWQL query and a map specification and generates an image of the resulting map. Applications can use this to show a map of the federated context to the user.

The Navigation Service extracts topological information from the federated context model. Given a start- and an end point, it returns a list of way points.

7.4 RECONCILABILITY OF THE APPROACHES

This Section examines if and how the concepts of AHSS and Nexus are reconcilable. If AHSS is supposed to work as part of a Nexus federation, it has to support the spatial model of Nexus. On the other hand, if Nexus does not support every AHSS feature, it is not that much of a problem. In that case, applications accessing the AHSS data through the Nexus can just use the features that are also supported in Nexus. These applications can then still access the data through the AHSS interfaces. Section 7.4.1 examines how an AHSS Spatial Server can fit into the Nexus architecture. Section 7.4.2 then compares the spatial models and shows how the different models can be reconciled. The choices that have been made for the implementation are described in 7.5.

7.4.1 ARCHITECTURE

When AHSS is to function as part of the Nexus platform, it has to be decided as what part of the platform it should function. In order to benefit from the federation capabilities of Nexus, it has to be placed below the federation layer of Nexus.

As illustrated in Figure 7-4 on page 57 the Nexus spatial model server is a good starting point. Like the AHSS spatial server, spatial model servers store the model information. Nexus spatial model servers have to understand queries in AWQL, and have to send replies in AWML.

7.4.2 SPATIAL MODEL

There is a major difference in the approaches of the spatial models of Nexus and AHSS. Nexus has the Nexus Augmented World Model, a standardized modeling scheme that describes the way the real world can be modeled. It describes the objects of the real world that can be described in the world model. This model provides a broad basic set of classes that can be used for most applications. For special applications, this basic model can be extended by using Extended Class Schemas.

AHSS on the other hand does not deal with the semantics of the data that much. It does not provide a class schema that defines the classes an application developer who is using AHSS can use. AHSS instead provides the basic type “Location”, which can be easily extended without going through an extended process to change an official AHSS modeling standard. The rationale behind this decision is that AHSS is targeted towards small environments like the Aware Home, where new applications are being developed in rapid succession. While there is application experience about what aspects of space are relevant to applications and should therefore be modeled, it was decided that the available experience was not comprehensive enough to define an all-encompassing modeling standard. In a research environment like the Aware Home, it is possible to reach quick ad-hoc agreements

between the programmers on how to model the house. A set modeling standard would provide little advantage, and would stifle the creativity of the developers. At the current stage, where much of the design space still needs to be explored, this would be counterproductive. So the main concern of AHSS is to provide an infrastructure to share and maintain the spatial model, to provide the primitives needed to model the world and flexible extension mechanisms for the model - without specifying how to exactly model the world.

Because of this, AHSS is in general flexible enough to accommodate the Nexus Augmented World Model. Objects in the Augmented World Model can be mapped to AHSS Locations. The attributes of the Nexus objects can be mapped to user-defined attributes in AHSS. This mapping can be done by name - Nexus attributes can be mapped to AHSS attributes of the same name. The type-mapping from Nexus to AHSS attributes can be done by utilizing the following technique: Nexus-attribute types whose semantics are supported by AHSS and have to be specially treated for querying purposes (especially the Location-types) can be mapped to the respective AHSS type. Other attributes can be stored as their string-representation as it is provided by AWML.

Because AHSS is indifferent towards the semantics of the user-defined attributes it stores, it can generally support arbitrary ECS. In fact, this flexibility creates a problem: Nexus requires that Spatial Model Servers only return attributes a client can understand. I.e. a Nexus spatial model server can only return objects as a result of a query that are of a type that is part of the Standard Class Schema or one of the Extended Class Schemata that the client explicitly specified. So for example if the result of a query is a “building”, but the client only understands “staticObject”, then the object would have to be upcast all the way to staticObject and be returned as such. Also, only the attributes that are specified in the respective Nexus class schemata can be returned. In Nexus-only operation, i.e. if only Nexus applications put objects into the AHSS spatial server, then this is no problem because these applications will only create attributes they know. But if applications access the AHSS spatial model server through the AHSS interface as well as through the Nexus interface, then there is a problem: AHSS applications could create attributes that simply do not exist in the Nexus class schemata. So for example a building could get an attribute “type-of-roof”, which simply does not exist in Nexus. Then this attribute has to be filtered out of results of Nexus queries. It is important to keep that in mind in order not to violate the Nexus specification. This will be discussed in more detail in a later Section.

So in fact it is possible to support the Nexus Augmented World Model in AHSS. One restriction that remains is that AHSS does not support multiattributes, that are multiple attributes of the same name on the same object. This is not a problem per se, because Nexus does only use multiattributes on the federation level and not on the spatial model server level. So since AHSS serves as a spatial model server, it does not have to support multiattributes. But if multiattributes in the future are to be used on spatial model server level, then the mapping has to be changed.

7.5 IMPLEMENTATION

It was decided to integrate AHSS into Nexus as a spatial model server. This way, an area which is served by AHSS appears to be an Augmented Area to the Nexus federation. The major challenge is to integrate the concepts of Nexus and AHSS, as discussed above. The integration does not have to be complete: While AHSS needs to support all the concepts of Nexus in order to function as part of a Nexus federation, the reverse is not necessary. In the approach chosen here, Nexus objects are mapped to AHSS Locations as described above. Basic attributes of Nexus attributes like the object id and the type are mapped to their AHSS counterparts. Nexus attributes that have no standardized AHSS counterpart are mapped to extended AHSS attributes.

To work as part of a federation, the AHSS spatial server has to perform several tasks. First, it needs to register with the Nexus Area Service Register to give the federation the information it needs to dispatch the queries. It has to specify which area the spatial server covers, and which Nexus object types it provides. The Nexus federation queries the spatial servers by using AWQL. Thus AWQL interface had to be added to Nexus which processes the query and update operations against the AHSS data. Also, the Nexus Federation expects the replies in AWML format. Therefore the AWQL interface has to map the AHSS Locations to Nexus objects, and then has to serialize them into AWML. AWQL queries specify the class schemata the client understands. The replies have to contain only objects that conform to these class schemata. Also, if a Nexus client queries e.g. for BuildingElements, objects of descendants of BuildingElement that match the query have to be returned. So it is necessary for AHSS to know the Nexus Class Schemata (CS) class hierarchy (Standard Class Schema plus appropriate Extended Class Schemas), which is modeled in XML. Therefore, a CS interpreter has been added which reads the appropriate CS specified in the query. The CS is specified as an XML Schema [XML SCHEMA 2000]. It then handles the CS inheritance hierarchy issues of the query and strips the resulting AHSS Locations of all attributes which are not part of the class schema. If a Nexus application is interested in getting all the spatial objects with all their attributes from the infrastructure, it can specify that the results should conform to the ”Generic” class schema. Then all AHSS Location attributes are returned.

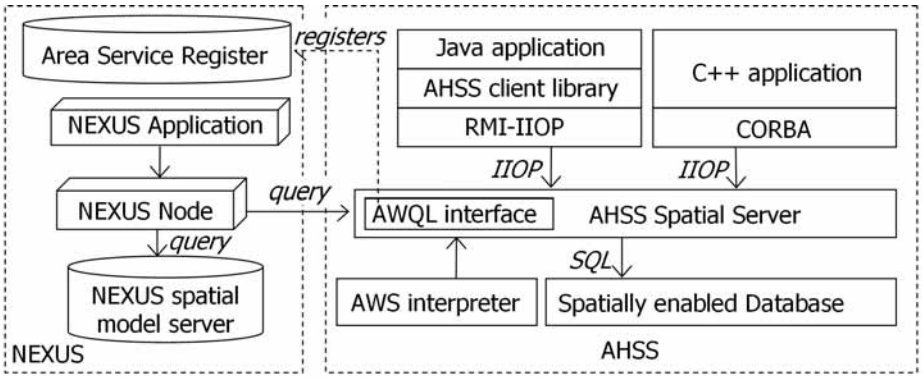


FIGURE 7-5: **The extended AHSS architecture**

Figure 7-5 on page 62 gives an overview of the necessary changes to the AHSS architecture.

AHSS uses the JDOM library to do the XML handling. An incoming AWQL request is represented as a JDOM tree. Then parts of that tree (the restriction, filter, closest, update and scope elements) are passed to handler methods that deal with them. The processing of the query follows the structure of the AWQL document. First, the restrictions are evaluated, and all Locations that conform to the restrictions are retrieved from the database. In this process, the inheritance relations of the specified SCS and ECS are considered. Then, if a “closest” clause exists, it is evaluated. Afterwards, the results are serialized into an AWML result. In the serialization process, the “filter” clauses are evaluated, and only requested attributes are serialized. In this process, the specified Augmented World Schemata are evaluated, and only Locations of the types that a client understands are returned. Then, the resulting AWML is returned to the client.

7.6 PERFORMANCE

In Section 6.2, the performance aspects of the Aware Home Spatial Service when used though the AHSS-IIOP interface were examined. But when the AHSS is accessed through the AWQL interface, then additional processing time has to be spent. Because XML is used, the amount of data that has to be transmitted is larger than in the native AHSS case. Also XML has to be parsed, and also generated for the reply. So the question arises how much performance this additional effort costs. So the same test scenarios as in 6.2 were tested again, this time through the AWQL interface. The update scenario yielded a throughput of 10 updates per second with one client. This is about 1/6th of the throughput of the native AHSS interface. The query performance also dropped to about 1/6th of the throughput of the native AHSS interface (to about 10 queries per second).

It can be assumed that the performance penalty can be reduced by optimizing the XML handling in the server. At the moment, AHSS uses JDOM to handle XML. Transition to lower-level XML processing APIs like SAX could increase the performance. Also, using up to date server hardware, the system can still provide enough performance to serve an Aware Home scenario.

Furthermore, the integration in the Nexus platform makes it possible to use the federation capabilities of Nexus, which makes it possible to use multiple servers to cover a house. Also, it can be assumed that local applications can still use the native AHSS-IIOP interface. This allows them to use the lightweight access method for most of their “everyday-access”. The AWQL can then be used only for external access which comes through the federation, and which is supposedly less frequent.

7.7 CONCLUSION

The integration of AHSS into Nexus provides the advantages that were expected. First of all, AHSS becomes part of the Nexus federated world model. Nexus applications can then use context information that is stored within the AHSS. This makes it possible to use existing Nexus applications with the AHSS without having to modify the applications. Also, Nexus applications can store their context information into the AHSS. This information can further enrich the context model which AHSS applications can access. The local AHSS interfaces are still available, so local applications can access AHSS either through the local interfaces or through the Nexus-AWQL interface.

This approach makes it possible to use simple, specialized infrastructures to support local applications, while using the Nexus federation to support greater scalability needs. The context model of each specialized infrastructure has to support topographic modeling and has to be flexible enough to store the attributes which are required by Nexus.

One major remaining issue is the semantic integration of the two systems. While Nexus applications can access data in the AHSS, they usually (without resorting to the “Generic” class schema workaround described in 7.5) can only access objects and their attributes that are also contained in a Nexus class schema. So in fact, Nexus applications will usually only access information that is modeled according to Nexus standard. So it will only see the subset of information it knows about. The Nexus world with its rigid semantic standards is relatively stable, while AHSS has a tendency to be more dynamic in its modeling standards. So in order to optimize the benefits of the Nexus/AHSS integration, it would be beneficial to identify a set of modeling “best practices” for indoor applications. Once these best practices stabilize, they could be formalized as a Nexus ECS. This class schema could then serve as a basic modeling standard. Most of the relevant information could then be modeled within this standardized schema. AHSS applications could then build on this standard and dynamically extend it as needed. So AHSS would serve as a test bed,

in which the modeling practices can evolve. Once they are stable and proven, they could be published as Nexus class schemata.

CHAPTER 8

EXTENDING THE AUGMENTED WORLD MODEL FOR INDOOR APPLICATIONS

Everything in software changes. The requirements change. The design changes. The business changes. The technology changes. The team members change. The problem isn't change, per se, because change is going to happen; the problem, rather, is the inability to cope with change when it comes.
– Kent Beck, Extreme Programming Explained

8.1 EXTENDING THE AUGMENTED WORLD MODEL

The current Augmented World Model of Nexus already offers a wide range of classes to model the real world. The Nexus Augmented World Model, as defined in the standard and extended class schemata was first developed by Jens Meßmer in [MEßMER 2000], and has since then been under continuous development by the Nexus project. Through the course of implementation of the Nexus project, the class schemata have been extended where it became necessary, and various changes have been made. The current authoritative state of the Nexus Augmented World Model can be found in the internal wiki pages of the project.

The classes that are specified by the Nexus Augmented World Model cover the requirements that were elicited from the usage scenarios that were already explored. So far, ubiquitous computing scenarios in home environments have not been considered in much detail in the Nexus project. While there are some classes suitable for indoor application scenarios, there are other classes missing that are relevant to ubiquitous computing in home scenarios. Using observations and experiences from the Aware Home, it was possible to identify objects that are not contained within the current Augmented World Model, but which are useful in home scenarios. Because these classes are mainly useful for indoor scenarios, they were encapsulated into an extended class schema, the HOME schema.

This schema cannot capture all possibilities of objects that might be important for the indoor domain. Instead, it aims at providing a core of objects in the home domain which can be considered relevant according to the experiences and the work in the Aware Home.

As it has already been described in the last Section, one of the design principles of AHSS was *not* to create heavyweight modeling standards, because the design space for ubicomp applications in the home domain was not explored to the extend to allow us to elicit the requirements in a comprehensive and stable fashion. The requirements for spatial models in the indoor domain are still a very fast moving target. Every day new applications in the indoor domain are explored, and the modeling has to evolve with them. So the Augmented World Model extensions can by no means be final or comprehensive for the domain. They should be considered as a first shot on how to create models for indoor applications.

There are two kinds of active objects in the model. *Reference objects* and *Integrated Objects. Reference Objects* describe an object or resource in the real world, and provide information on how to use them. Their main use is to provide means for discovery of these objects or resources. This means that the spatial model itself currently does not provide the means of accessing these resources directly. For example, right now the model does not implement a standard protocol to handle multimedia streams. Although this is targeted for later in the Nexus project, it is not available yet. So right now, other existing protocols and infrastructures have to be used to access multimedia devices like cameras. For other resources like printers, there are a variety of standardized and proven protocols available (e.g. the Internet Printing Protocol) that it does not seem to be necessary to invent new protocols. Ref-

erence objects in general only have a small set of attributes that can be used for resource discovery. This is mainly because most of the attributes of these resources are very specific, and the configuration of these resources is managed by the protocol that is used to access them.

Integrated objects are active objects whose functionalities are handled directly by the model. Applications do not have to use another protocol to access the functionality of these objects. The real world resources they represent directly communicate through the spatial model, i.e. they write new data into the spatial model, and they read data directly from it.

8.2 THE HOME EXTENDED CLASS SCHEMA

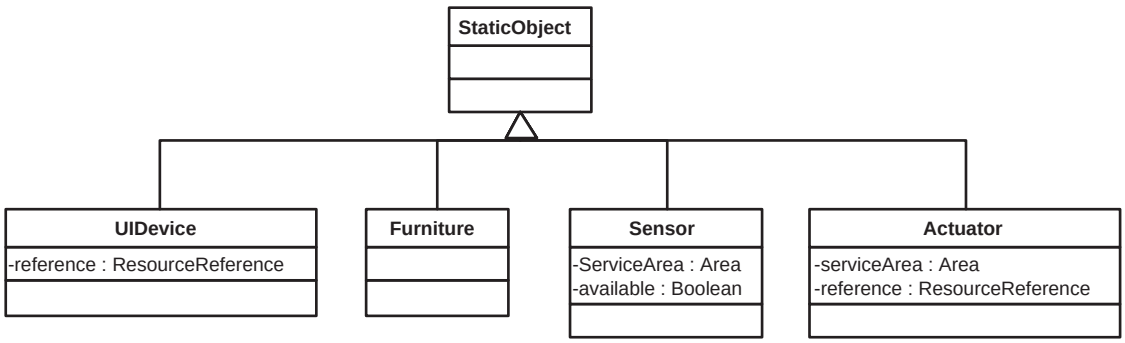


FIGURE 8-1: **The top level of the class hierarchy**

All classes in the HOME ECS (extended class schema) are subclasses of the Static Object class, because their position is fairly static. The subclasses will be described in the following sections.

8.2.1 FURNITURE

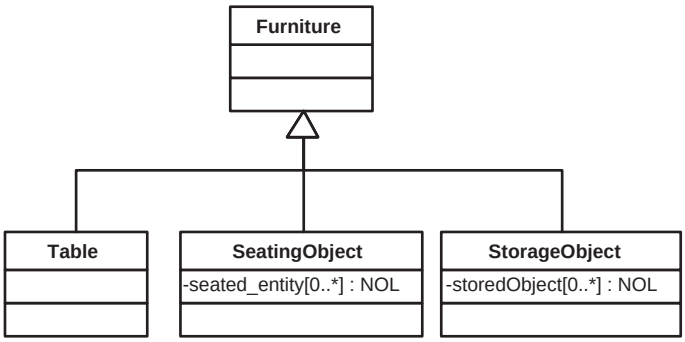


FIGURE 8-2: The Furniture type hierarchy

The standard class schema can structure environments down to room level. In home scenarios, as we have seen in the scenarios in Section 3.1, rooms generally are not empty. Instead they are populated by various static objects which impede movement. Special kinds of furniture can be used to store objects or to contain (seat) people. Furniture in general is passive and does not provide any active functionality in the model.

8.2.2 USER INTERFACE DEVICES

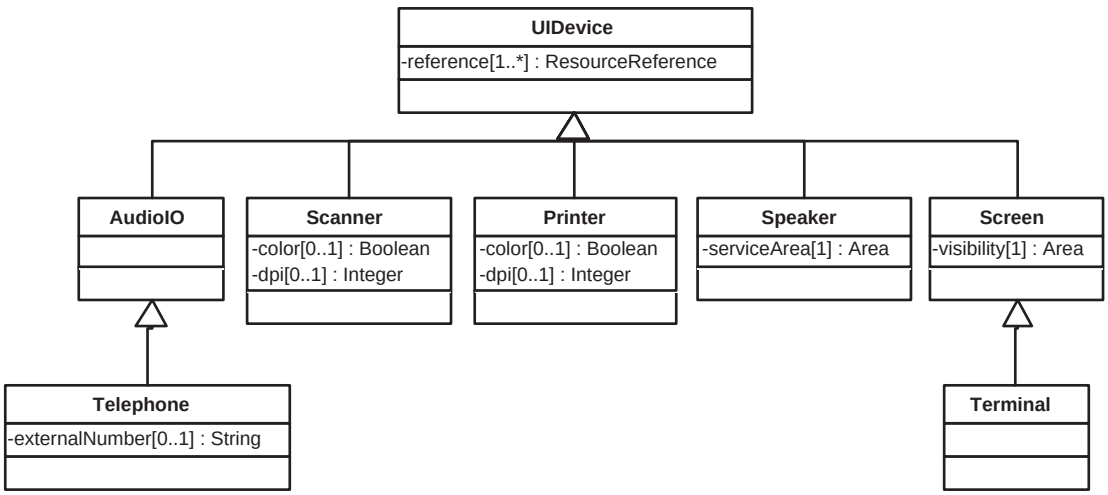


FIGURE 8-3: The UIDevice type hierarchy

User Interfacing is a central task. It is important to interface with the inhabitants of the house in order to provide information and services, and also to get their input. This is evident from all scenarios in Section 3.1 – they all have to interact with the user. UIDevices represent devices that are designed specifically for user interaction. While a room microphone can be used to get input from the user, it is not modeled

as a UI device, because it can also be used as a normal sensor to monitor the environment without being part of a direct user interaction (see Section 8.2.4). All UIDevices are reference objects and therefore have a reference that describes the protocol and the address of how the resource can be utilized.

AudioIO objects are objects that can send and receive audio and target a single person. This could for example be a headset. Telephones are AudioIO objects that are reachable from outside the modeled environment and have an external address (the external phone number). These devices are important for communication applications. Printer objects and scanner objects combined can deliver the functionality of a fax machine and are an important tool for collaboration and the exchange of documents. Speakers can be used to relay communication, play music or to deliver pagings. A screen can be used to display information or entertainment. A Terminal is a screen combined with input devices like a keyboard, a mouse or a graphic tablet or something similar.

8.2.3 ACTUATORS

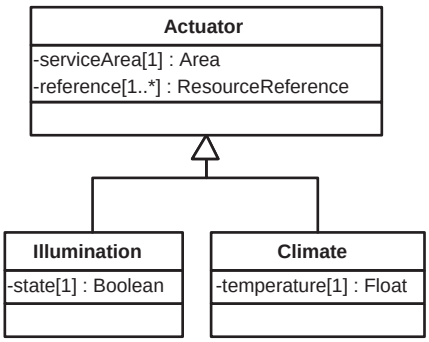


FIGURE 8-1: The Actuators type hierarchy

Actuators are very important in the indoor domain, but their semantics are application specific. So only two special actuator types have been specified in this schema. Illumination control controls the illumination of the area it is associated with. Climate control objects represent the air conditioning control that controls the climate of the respective area. An example for using actuators in the scenarios in Section 3.1 is that the Smart Doorbell eventually has to open the door.

8.2.4 SENSORS

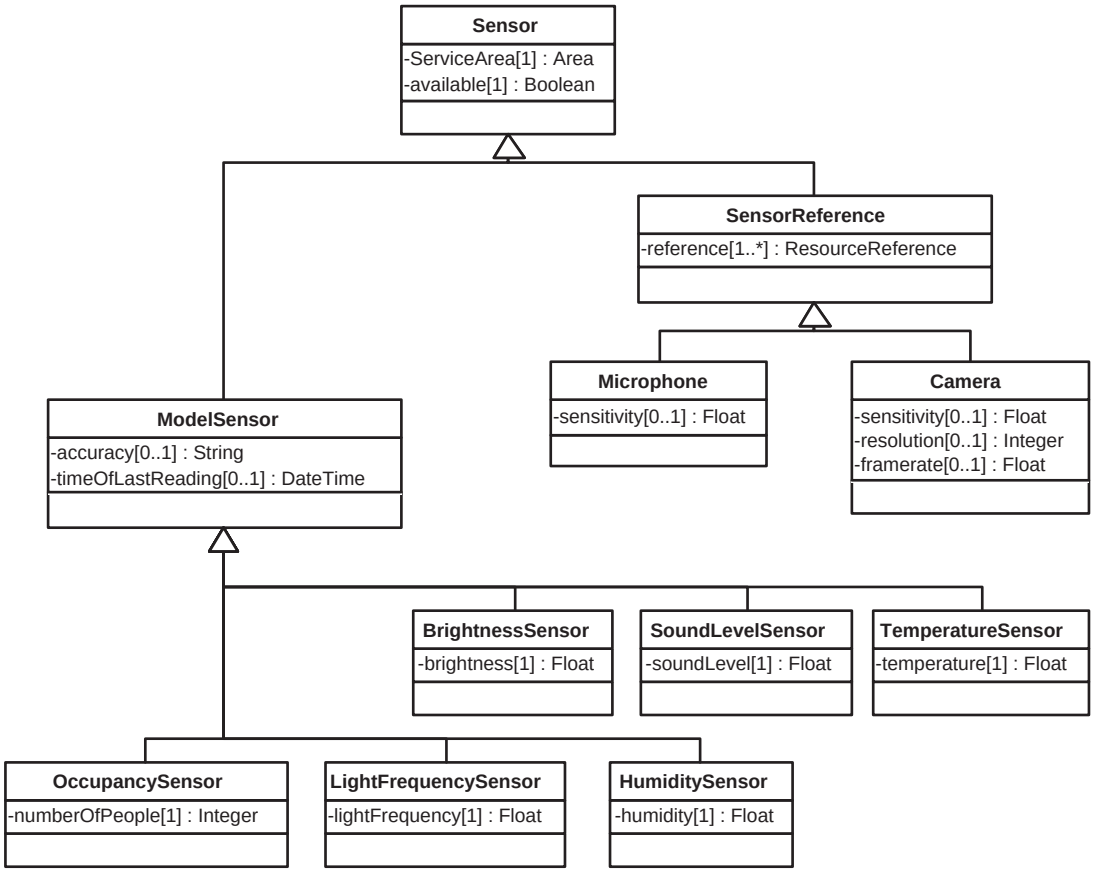


FIGURE 8-4: The Sensors type hierachy

Sensors are a crucial part of augmented home environments. They are used to gain awareness of the actions of the inhabitants of the augmented environment, and also about the state of the environment. In the scenarios in Section 3.1, microphones are used to detect interaction attempts by the user. Cameras provide information about activity in the house. Brightness, sound level, temperature and the other sensors can provide information that can be useful for the authorities that are notified by the smart alarm system in case of an emergency.

CHAPTER 9

CONCLUSION

If you wait for a complete and perfect concept to germinate in your mind, you are likely to wait forever.
– Tom DeMarco

9.1 SUMMARY

In the first part of this thesis, a specialized spatial modeling infrastructure for smart home environments was created. The thesis started out by first defining the relevant terms, introducing the projects that are the environment this thesis was created in, and then discussing relevant work.

Then the requirements for the specialized infrastructure were elicited by using the scenario analysis method. Consequently, the architecture for the specialized infrastructure, the Aware Home Spatial Service (AHSS) was developed and then implemented.

The experience with AHSS system showed that while AHSS was satisfying the requirements it was created for, it was not suited very well to scale beyond a single server.

This started the second part of this thesis – the integration of AHSS into Nexus. After a discussion of the Nexus architecture, the possibilities to integrate AHSS into Nexus were discussed, and its challenges were outlined and the challenges were met. Then, possible additions to the Nexus Augmented World Model were discussed in order to better support ubiquitous computing applications in home environments.

9.2 CONCLUSION

The specialized spatial modeling infrastructure that has been developed in this thesis, the AHSS, is a lightweight and pragmatic approach for an infrastructure that supports spatial modeling for home environments. It is no silver bullet for all the infrastructure challenges in ubicomp environments, nor does it aim to be. It provides a basic set of vocabulary (the AHSS spatial model) to create spatial models. It also provides a simple but powerful infrastructure to share this spatial model among multiple applications. Because of its great flexibility, the infrastructure is especially well suited for dynamic research environments like the Aware Home, where new applications are developed almost on a daily bases. For an environment like this, it is very important that extensions and even changes to the spatial model can be done quickly and easily.

While this works well and offers enough performance for residential homes, it is clear that this model does not scale well beyond a single building (i.e. the area that can be covered by a single AHSS server). AHSS does not support multiple administrative domains and cannot federate multiple AHSS spatial servers. In short: It is the pragmatic solution for “your home”.

On the other hand, there is Nexus, which uses a comparatively more heavyweight approach and addresses the challenges that AHSS does not address. It uses the approach of defining and standardizing the spatial model in great detail, while still keeping it extensible.

The integration of AHSS into Nexus gives you the best of both worlds. AHSS applications can still use the simple AHSS interfaces with all its flexibility, while the Nexus platform makes the standardized parts of the AHSS spatial model available as part of a global context model. Application developers can use AHSS to quickly develop their applications and to evolve the spatial model needed for those applications. Once the requirements for new parts of the spatial model have reached a certain stability, they can be migrated into the Nexus world. For example, they can be standardized as a Nexus Extended Class Schema.

The extensions for the Augmented World Model that have been proposed in this thesis can only be a first approach. It is very difficult to conceive the requirements for world models in great detail in a top-down, from-scratch approach by just thinking about it. The requirements for a world model come from applications. So in order to create a well-founded and sound world model, I believe that application experience is very important. But not only on the conceptual level, but right down to the implementation of these applications – because many problems with good-looking concepts become evident only after trying to implement them. For this reason, people working on software infrastructures should be in close contact with application developers. And once application developers start using these infrastructures, their feedback can evolve these infrastructures. Therefore it seems to be important to provide working infrastructures quickly, even if they are not the silver bullet. That way application developers can get accustomed to and evaluate the concept of spatial modeling for their domain, and provide valuable feedback. This thesis is a step into that direction.

9.3 Future Work

In the long run, it will be necessary to create modeling standards for a variety of application domains. These standards have to be based on application experience. They also need to be part of a widespread consensus in order to be accepted. There needs to be a standardization process for these spatial modeling standards.

CHAPTER 10

REFERENCES

Educated is the one who knows where to find what he does not know
- Georg Simmel

[BAUER ET AL 2002]

Bauer, M., Becker, C., Rothermel, K.

Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks

Personal and Ubiquitous Computing. Vol. 6(5-6). S. 322-328, London: Springer-Verlag (2002)

[BECKER ET. AL 2002]

C. Becker, G. Schiele, H. Gubbels

BASE - A Micro-kernel-based Middleware For Pervasive Computing

To appear in Proceedings of the IEEE International Conference on Pervasive Computing and Communication (PerCom), Fort Worth, USA 2002.

[BECKER ET AL 2001]

C. Becker and G. Schiele

New mechanisms for routing in ad hoc networks

4th Plenary Cabernet Workshop, 2001, Pisa, Italy

[BRUMIT ET AL 2000]

Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S.

EasyLiving: Technologies for Intelligent Environments

Handheld and Ubiquitous Computing, September 2000.

[CANU 2003]

<http://canu.informatik.uni-stuttgart.de/>

[CHEN 2000]

Chen, G., Kotz, D.

A Survey of Context-Aware Mobile Computing Research

Dartmouth Computer Science Technical Report TR2000-381, Dartmouth College (2000)

[COSCHURBA ET AL. 2000]

Coschurba, Peter; Kubach, Uwe and Leonhardi, Alexander

Research Issues in Developing a Platform for Spatial-Aware Applications

Proceedings of SIGOPS European Workshop, Kolding, Denmark, 2000

[COULORIS ET AL]

<http://www-lce.eng.cam.ac.uk/qosdream/>

[DEY 1999]

Dey, A., Abowd, G.

Towards a better understanding of context and context-awareness

Georgia Tech GVU Technical Report, GIT-GVU-99-22 (1999)

[DIGITAL POWERLINE]

<http://www.powerlinecommunications.net/AscomPowerlineCommunication.htm>

[ESPINOZA 2001]

Espinoza, F., Person, P., Sandin, A., Nyström, H., Cacciatore, E., Bylund, M.

GeoNotes: Social and Navigational Aspects

Proc. of UBICOMP 2001, Atlanta, USA (2001)

[GML 1.0]

OpenGIS Cons.

Geography Markup Language Specification 1.0

<https://feature.opengis.org/rfc11/GMLRFCV1_0.html>

[GRZAN 2002]

Grzan, Stjepan

Enabling technology for an indoor location aware information system

Diplomarbeit 1958. Computer Science Faculty, University of Stuttgart, Germany, 2002

[HARTER 1999]

Harter, A., Hopper, A., Steggles, P., Ward, A., Webster, P.

The anatomy of a context-aware application

Proc. of fifth annual International Conference on Mobile Computing and Networking, Seattle, WA, ACM Press (1999) 59-68

[HOHL ET AL. 1999]

Hohl, Fritz; Kubach, Uwe; Leonhardi, Alexander; Rothermel, Kurt and Schwehm, Markus

Next Century Challenges: Nexus - An Open Global Infrastructure for Spatial-Aware Applications

Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99), Seattle, Washington, USA

[HOMEPNA 2003]

<http://www.homepna.org/>

[KHELIL ET. AL 2002]

A. Khelil, C. Becker, J. Tian, and K. Rothermel

An Epidemic Model for Information Diffusion in MANETs

Proceedings of the International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems at MobiCom 2002, Atlanta, USA.

[KIDD 1999]

Kidd,C., Orr, R., Abowd, G., Atkeson, C., Essa, I., MacIntyre,B., Mynatt, E.,Starner, T., Newstetter, W.

The Aware Home: A Living Laboratory for Ubiquitous Computing Research

Proc. of 2nd International Workshop on Cooperative Buildings - CoBuild (1999)

[LEONHARDI 1999]

Leonhardi, A., Kubach, U., Rothermel, K.

Virtual Information Towers - A metaphor for intuitive, location-aware information access in a mobile environment

Proc. of third International Symposium on Wearable Computers, San Francisco, CA (1999) 15-20

[LEONHARDI & ROTHERMEL 2001]

Leonhardi, A.; Rothermel, K.

Architecture of a Large-scale Location Service

Technical Report TR-2001-01, University of Stuttgart, 2001

[MEßMER 2000]

Jens Meßmer

Modellierung der Augmented World in Nexus

Diplomarbeit Nr. 1870, Computer Science Faculty, University of Stuttgart, Germany, 2000

[OGC 99-049]

Open GIS Consortium Inc.

OpenGIS Project Document 99-049: Simple Features Specification For SQL;

<http://www.opengis.org/techno/specs/99-049.pdf>

[PASCOE 1999]

Pascoe, J.

The Stick-e note architecture: Extending the interface beyond the user

Proceedings of the 1997 International Conference on Intelligent User Interfaces (1997) 261-264

[POSTGIS 2003]

Refractions Research

PostGIS

<http://postgis.refractions.net>

[PRESENTATION 2003]

Daniela Nicklas

Nexus Internal Presentation

[USE CASE 2000]

Nexus Research Group

Use-Cases für Nexus und mögliche Realisierungen

Nexus Research Group - Internal Paper, June, 14th, 2000

[W3C 2001]

World Wide Web Consortium

The Semantic Web activity

<http://www.w3.org/2001/sw>

[WEISER 1991]

Mark Weiser,

The Computer for the Twenty-First Century

Scientific American, September 1991

[X10 2003]

<http://www.smarthome.com/about_x10.html>

[XML 1998]

World Wide Web Consortium

Extensible Markup Language (XML) 1.0

<http://www.w3.org/TR/1998/REC-xml-19980210>

[XML SCHEMA 2000]

World Wide Web Consortium

XML Schema Part 1: Structures

<http://www.w3.org/TR/xmlschema-1>

LIST OF FIGURES

Erklärung

Ich versichere, dass ich diese Arbeit selbständig verfasst und nur die angegeben Hilfsmittel verwendet habe.

Othmar Lehmann