System Support For Context-Aware Computing

Habilitationsschrift zur Erlangung der

Venia Legendi in Informatik

vorgelegt von

Dr. phil. nat. Christian Robert Becker

aus Hanau

Institut für Parallele und Verteilte Systeme (IPVS)

Abteilung Verteilte Systeme

Fakultät Informatik, Elektrotechnik und Informationstechnik

Universität Stuttgart

Juni 2004

Table of Content

1. INTRODUCTION 6

2. CONTEXT 10

2.1. Context and Context-Awareness 11

2.2. Classification of System Support for Context-Aware Computing 18

2.3. Dimensions 19

2.4. Classes 24

2.5. Requirements 27
2.5.1. Adaptation by application 27
2.5.2. Adaptation by System 33

2.6. Related Work 37
2.6.1. Adaptation by application – context management platforms 37
2.6.2. Smart Environments 41
2.6.3. Adaptation by System in Ad Hoc Networks 42

2.7. Contributions contained in this thesis 44

3. ON LOCATION MODELS FOR UBIQUITOUS COMPUTING 56

3.1. Introduction 56

3.2. System Model 57
3.2.1. Basic Properties of Coordinates 59
3.2.2. Geometric Coordinates 60
3.2.3. Symbolic Coordinates 60

3.3. Requirements for Location Models 61
3.3.1. Position Queries 61
3.3.2. Nearest Neighbor Queries 62
3.3.3. Navigation 63
3.3.4. Range Queries 64
3.3.5. Visualization 65
3.3.6. Requirements 65

3.4. Geometric Location Models 67

3.5. Symbolic Location Models 68
3.5.1. Set-based Model 68
3.5.2. Hierarchical Models 70
3.5.3. Graph-based Model 71
3.5.4. Combination of Graph-based and Set-based Symbolic Models 73
3.5.5. Summary 74

3.6. Hybrid Location Models 76
3.6.1. Subspaces 76
3.6.2. Partial Subspaces 77
3.6.3. Discussion 78

3.7. Summary and Classification of Existing Approaches 79

3.7.1. Set-based Location Models 81
3.7.2. Graph-based Location Models 81
3.7.3. Hierarchical Location Models 82
3.7.4. Combined Symbolic Location Models 82
3.7.5. Hybrid Location Models 83

3.8. Conclusion 84

4. USENET-ON-THE-FLY - SUPPORTING LOCALITY OF INFORMATION IN SPONTANEOUS NETWORKING ENVIRONMENTS86

4.1. Introduction 86

4.2. System Model 89

4.3. Application Scenario 89

4.4. The Usenet-on-the-fly Prototype 90
4.4.1. Functionality 92
4.4.2. Architecture 93

4.5. Information Dissemination Protocol 95

4.6. Simulations 97

4.7. Simulation Model 98

4.8. Simulation Results 102

4.9. Discussion 105

4.10. Related Work 106

4.11. Conclusion and Outlook 107

5. A PROTOCOL FOR DATA DISSEMINATION IN FREQUENTLY PARTITIONED MOBILE AD HOC NETWORKS 110

5.1. Introduction 110

5.2. System Model 112

5.3. Forwarding Strategies 113

5.4. Negotiation-based Ad hoc Data Dissemination Protocol: NADD 114
5.4.1. Data Structures 115
5.4.2. Protocol 116

5.5. Simulation 121
5.5.2. Replication Latency 122
5.5.3. Message Overhead 124

5.6. Related Work 125

5.7. Conclusion 127

6. FROM HOME TO WORLD: SUPPORTING CONTEXT-AWARE APPLICATIONS THROUGH WORLD MODELS 128

6.1. Introduction 128

6.2. Requirements 129
6.2.1. Scenario 130
6.2.2. Derived requirements 131
6.2.3. Context modeling 132

6.3. Related work 134

6.4. AHSS - A local architecture 136
6.4.1. Spatial model 136
6.4.2. System architecture 138
6.4.3. Experiences 140

6.5. Nexus - A global architecture 141
6.5.1. Platform architecture 144
6.5.2. Spatial model servers 144
6.5.3. Nexus nodes 145
6.5.4. Value added services 146

6.6. The NexusScout application 146

6.7. AHSS in Nexus 147
6.7.1. Conceptual integration 148
6.7.2. Technical integration 148
6.7.3. Experiences 149

6.8. Conclusion and future work 150

7. MIDDLEWARE AND APPLICATION ADAPTATION REQUIREMENTS AND THEIR SUPPORT IN PERVASIVE COMPUTING 152

7.1. Introduction 152

7.2. System model 153
7.2.1. Scenario 153
7.2.2. System Model 154

7.3. Requirements 156
7.3.1. Application adaptation requirements 156
7.3.2. System software adaptation requirements 157

7.4. BASE a Microbroker based Middleware 159

7.5. PCOM 161

7.6. Related Work 164
7.6.1. Middleware Systems 164
7.6.2. Component Systems and Pervasive Computing 165

7.7. Conclusion and Outlook 166

8. BASE - A MICRO-BROKER-BASED MIDDLEWARE FOR PERVASIVE COMPUTING 168

8.1. Introduction 168

8.2. Requirements 170

8.3. Related Work 172
8.3.1. Conventional Middleware Systems 172
8.3.2. Dynamically Reconfigurable Middleware 173
8.3.3. Middleware for Resource-Poor Devices 173
8.3.4. Middleware for Pervasive Computing 174

8.4. BASE 175
8.4.1. Design Rationale 175
8.4.2. BASE Architecture 178

8.5. Implementation Status and Evaluation 184
8.5.1. Implementation Status 185
8.5.2. Memory Size 185
8.5.3. Execution Performance Overhead 186

8.6. Conclusion and Future Work 188

9. PCOM – A COMPONENT SYSTEM FOR PERVASIVE COMPUTING 190

9.1. Introduction 190

9.2. System Model 192
9.2.1. BASE. 193

9.3. Adaptation Models 193

9.4. Requirements 195

9.5. PCOM 196
9.5.1. Application Architecture 197
9.5.2. Components 197
9.5.3. Adaptation 202

9.6. Evaluation 205
9.6.1. Service Selection 206
9.6.2. Communication 210
9.6.3. Resource Overhead 212

9.7. Related Work 212

9.8. Conclusion 215

10. EXPERIENCES: MINIMALISM AND EXTENSIBILITY IN BASE 216

10.1. Introduction 216

10.2. BASE – A Micro-broker Based Middleware 218
10.2.1. Design Rationales 218
10.2.2. Architectural Overview 219

10.3. First Experiences 220

10.4. Porting and using BASE 221
10.4.1. Porting BASE 221
10.4.2. BASE as a Platform for Components 222

10.5. Experiences 222
10.5.1. Technical Modifications 223
10.5.2. Conceptual Modifications 223
10.5.3. Lessons Learned 224

10.6. Conclusions 227

11. SUMMARY AND OUTLOOK 228

11.1. Contributions 230

11.2. Outlook 231

12. REFERENCES 234

13. PUBLICATIONS CONTAINED IN THIS THESIS 245

1. Introduction

Pervasive Computing has been proposed over a decade ago. Over the last years a new community has been formed which strives to accomplish Weiser's vision of disappearing computers, which seamlessly interact providing users with information and services at any time, any place. Pervasive Computing belongs to the larger class of context-aware computing. This thesis provides a classification of system support for context-aware computing. Some fundamental aspects of context-aware computing, such as location models as an underlying structure of context models and data dissemination algorithms for ad hoc based systems are presented. The focus of this thesis covers Peer-to-Peer Pervasive Computing. After a discussion of requirements a two tier approach that is based on a lightweight middleware for establishing spontaneous groups in an ad hoc network and a component system allowing automatic adaptation of applications is presented.

"The most profound technologies are those that disappear" is perhaps the most cited statement from Marc Weiser's seminal paper "The Computer of the 21st Century" [Wei91]. In his vision of Ubiquitous Computing Weiser provides us with scenarios that have left their science fiction character due to the rapid progress in technology. The miniaturization of computing platforms, the proliferation of sensor systems, and the availability of short range wireless communication technology already provides the basic building blocks of Pervasive Computing. However, there are only few commercial applications available. Modern cars integrate mobile phones seamlessly into their user control, e.g., use the board computer display and control knobs on the steering wheel.

This effect can be partially explained by the initial lack of business models and research prototypes that explored Pervasive Computing technology by applying it to fancy scenarios which rarely showed evidence of use for society or businesses. A prominent example of such scenarios is Weiser's coffee machine which automatically brews coffee to be ready when a person awakes. However, there are more convincing Pervasive Computing examples that clearly identify its relevance in research and in its applications. Support of elderly people in their homes by monitoring their body functions, dispensing medicine, notifying qualified personal in case of unusual behavior or body functions can help senior citizens to stay in their familiar environment leading to more comfort in their lives and presumably a reduction in the health-care costs. Resource management in a smart factory can help to integrate the business processes into the production flow at a fine granularity. Procurement systems can ensure that the necessary resources are available, i.e., ordered as they are needed without over-stocking, as well as the production process itself can be optimized on the base of current information captured by sensors in the production plant along with the order data, the deadlines of cooperation partners, the logistics, etc.

Although the benefits of Pervasive Computing technology exist and are widely recognized, there are still open research questions to be solved. First, Pervasive Computing systems react based on their context. Context relates Pervasive Computing systems with the physical world. Users as well as information captured by sensor platforms are used by these systems to change their behavior. Thus, Pervasive Computing can be seen as an important class of context-aware systems. Second, interoperability issues are made harder by the vast heterogeneity of devices with respect to their resources and their specialization. In addition to that, the integration of sensor and computing platforms in everyday items will lead to a number of devices in different possibly overlapping administrative domains. New concepts with respect to the organization of systems, their administration, and their deployment have to be developed.

This thesis contributes to several research questions in the domain of context-aware computing. In Chapter 2 we will classify support for context-aware computing and discuss requirements and related work. A more elaborated presentation of the contribution of this thesis ends Chapter 2. In Chapter 3 location models for Pervasive Computing are discussed. A location model represents a common spatial structure of context models and thus is of great importance in order to allow spatial reasoning and interoperability between context models. Requirements on location models are discussed based on a brief use case analysis of context-aware applications. Possible approaches to represent location models are presented and classified according to their suitability to fulfill the requirements and the involved modeling effort. Information dissemination in an ad hoc network is discussed in Chapter 4 based on the "Usenet-On-The-Fly" application which provides users an information service offering locally relevant information. Besides this novel application a suitable protocol is presented and evaluated. An improvement of the underlying protocol for ad hoc networks with frequent partitioning is discussed in Chapter 5. An important question concerning the integration of different context models is the focus of Chapter 6. The integration of a context server for Georgia Tech's Aware Home into the Nexus platform and the consequences in concept and implementation are discussed.

Chapter 7 to 10 focus on support for Peer-to-Peer Pervasive Computing. This class of context-aware computing is challenged by the frequent changes in an application's execution environment. Support for adaptation has to be provided along with flexible middleware infrastructures. Chapter 7 presents a more detailed requirement analysis. Chapter 8 and 9 present a flexible middleware system (BASE) and a lightweight component model (PCOM) allowing for automatic application adaptation. Chapter 10 finally presents experiences from the port of BASE to an embedded system and from the implementation of PCOM on top of BASE. The thesis closes with an outlook to further research questions in the domain of context-aware computing.

2. Context

This chapter introduces context as a concept to enable applications to reason about and react to changes in the physical world. Context definitions and the classification of context into primary and secondary context are introduced before system support for context-aware computing is classified and related work is presented. The contributions of the following chapters are summarized.

Context-aware systems have attracted researchers in the past years starting from location-aware computing. Early work considered context to be [SAW94] related to the location of users, the people nearby, and resources which can be accessed based on the spatial proximity. Depending on the focus of research projects, further definitions of context have been proposed. Projects related to Human Computer Interaction focused on user's activity or social environment, e.g., in order to adapt the behavior of a cell-phone [SBG99]. The user's location was only of interest as long as it could be used to derive information about his activity. The progress in technology with respect to the miniaturization of computing and sensing devices will lead to billions of information sources placed in our physical world which will constantly report changes in the physical world captured via sensors. This information is related to locations in the physical world as well as to users. This is an integral part of context concerning the locations as well as the users. Some existing information spaces, e.g., the WWW, also provide information about physical entities. Common to this information independent of its origin, i.e., sensed by sensor platforms or provided by applications or information spaces, is the relation between physical entities such as users or locations and virtual entities, such as applications. The following definition is based on the discussion in [RBB03] and reflects this more general view on context information.

2.1. Context and Context-Awareness

Definition Context: Context is the information which can be used to characterize the situation of an entity. Entities are persons, locations, or objects which are considered to be relevant for the behavior of an application. The entity itself is regarded as part of its context**.**

It is interesting to see that an entity can be part of its context itself as well as an entity can be interpreted different depending on the context. For an example consider two applications dealing with trucks. A fleet management system would keep track about the position, the freight, the route of trucks along with other information, such as the trucks' maintenance rate, the assigned driver etc. A navigation system installed in the individual trucks would also consider the delivery routes but only for the individual truck. Other information, such as traffic jam information, is used to optimize the navigation between destinations.

From a context management perspective both applications may operate on the same context information. The fleet management system may access the context model in order to retrieve all trucks in a given area in order to decide which truck a given tour should be assigned to. Individual trucks or drivers may be queried for administrative issues. The car navigation system as well accesses the context model in order to update information as well as to query for individual data relating the truck and its current position. The point in time is also crucial information for both applications. The navigation system will take the current situation into account. Prognosis of the traffic situation can be used to improve the route planning. Thus, the navigation system will access context data based on time (present and future). The fleet management system will also access information regarding the current time and the future for planning of tours. In addition to that, history plays an important role in order to account for transportation costs and for issuing invoices.

Based on the scenario above we can derive that context information is accessed based on three major criteria:

- The *identity* of the entities
- The *location* of entities
- The *time* where the information is relevant

Because of the important role of identity, location, and time we refer to these as *primary context*. The role of primary context in context management obviously is the indexing of context information. Further information of entities can be accessed once they are found using the primary index. The additional context information, e.g., load of a truck, a person's email address, the vacancy of a taxi, are denoted as *secondary context*. Possible combinations of accessing context information are (location, time), (identity, time), or (identity, location, time). Note, that time may be used implicitly, e.g., an index may only refer to a distinct location. The time can then be interpreted as the current status of the context information related with the index.

The notion of primary and secondary context does not imply the relevance of context information from an application perspective. Some context – aware applications exist where secondary context, such as a user's activity, may be of relevance. However, in order to access the context information the primary context has to be used.

Context-aware applications can make use of context in many ways. The following definition captures common understanding of context-aware applications [DA99][RBB03][CK00].

Definition Context-Aware Application: an application is context-aware if it adapts its behavior depending on the context.

Based on this definition, four classes of context-aware applications can be isolated, which either select information or services, change their presentation, or issue some action based on context, or tag information to context:

- Context-based selection: information and services which are used by an application are selected based on context information, such as a user's preference, their physical proximity (the next printer) or relevance to the user (public transport schedules from the next bus stop).
- Context-based presentation: the way which and how information is presented to the user also depends on the context. A navigation system may change the way information is displayed based on the speed of traveling from a map to a direction based output using arrows or to audio output only in order not to distract the user.
- Context-based action: in contrast to context-aware presentation where a user is explicitly involved in the interaction with an application context-based action allows to automatically react to changes in the context without prompting the user. Examples are applications which automatically forward messages to the devices in a user's proximity, facility management systems which adjust light and heating conditions to user preferences.
- Context-based tagging: in contrast to selection, presentation, and action, which lead to an immediate change in the behavior of an application, tagging of information to context allows a later action based on this information. This allows applications, such as Stick-Enotes [Pas97], GeoNotes [EPS+01], or Virtual Information Towers

[LKR99]. These applications allow to associate information with context, typically location and the id of user in case of personalized information, and display this information to users when they are in proximity of this location.

Examples of context-aware applications can be found in a variety of domains. We will present some examples of context-aware applications in visitor information systems, navigation, annotations, support in workspaces, and smart environments.

Visitor Information Systems

Supporting mobile users with information about their spatial proximity is explored in many projects. The CyberGuide [LKA+96] at GeorgiaTech has explored a tour guide for visitor tours through their laboratories. An extension to CyberGuide provided support for tours through downtown Atlanta [AAH+97]. Users were provided with information about sights or projects and could create a diary of their visit. The Guide project [CDM+00a] developed a tourist guide for the city of Lancaster. Users are provided with information about historic sights and could use additional services, such as communication and restaurant reservation.

Navigation

The REAL project [BKW02] investigates adaptive navigation systems where the information to pedestrians changes (e.g., from a map to an arrow indicating the direction) depending on their speed of travel and the display they use (head mounted display or a display integrated into a bum bag).

Annotations

Annotations combine two classes of context-aware applications. First, an annotation is made combining information with context, such as a location where the information is of relevance or a user the information is addressed to, or a combination. Context-aware actions or presentations are executed when the context the information is tagged to is observed, e.g., a user enters a room where information for him is placed. The stick-e Notes System [Pas97] is a typical candidate in this class of applications. Virtual Post-Its can be tagged to context information and are displayed accordingly. The VIT System [LKR99] is based on the metaphor of virtual information towers which maintain information with relevance for a distinct geographic area. ComMotion [MS00] incorporates time and location as context in order to remind users of distinct tasks.

Support in Workspaces

Assisting the user in his daily workspace environment was investigated in the Active Badge System [WHG92] and Active Bat System [HHS+99] projects. The forwarding of incoming calls to the closest telephone to a user or teleporting of user interfaces, where graphical user interfaces "follow" the user to the next appropriate display. In the Netman project [KSS+99] wearable computing technology was used to provide context dependent information to service personal. The TEA project [STM00] investigated context dependent configurations of mobile phones depending on the context, such as a meeting taking place, the phone located on a table or in a briefcase.

Smart Environments

Smart Environments augment a spatial restricted area, so-called active or smart spaces, with various facilities for interaction with users, such as wallscreens, ticker-display, digitally enhanced whiteboards, tables with integrated displays, etc. Additionally, devices carried by users are integrated into such environments. In the Gaia project [RC00] applications are mapped onto the devices available in such environments via predefined mapping-scripts and thus make use of the offered functionality. Adaptation at runtime allows coping with changes in the execution environments based on an application model that separates model, view, and presenters. A different approach is taken by the interactive workspaces projects (IROS [JFW02]) which aims at the coupling of applications via an event heap. The integration and leaving of devices and services is supported by the event heap which decouples the different parts of an application. Context in such smart environments is typically reflected by the location of devices and services by implicit means. Devices in the same smart environment are considered to be of relevance and made available. The spatial scope of the smart environment defines its spatial context. Typical applications are tailored towards the purpose of the physical space the smart environment is based upon, i.e., support for meetings and teaching.

The Aura project [GSS+02] is an exception here, since it aims at the support of applications across different smart environments. The Aura Context Information Services [JS03] provides context information about users and locations as well as on devices and their network connections.

Project	Description	Primary Context	Class of Context Awareness
Cyberguide	Indoor guide	identity, location	selection
Guide	Tourist guide	location	selection
Stick-e Notes	Virtual Post-Its	location, identity	selection, action, tagging
VIT	Virtual Information Towers	location, time	selection
ComMotion	Location-based reminder	location, identity	selection, action
REAL	Navigation	location, identity	presentation
Active Badge: Telephone Assistant	Forwarding of phone calls	identity, location	action
Active Bat: Teleporting	Teleporting of user interfaces	identity, location	presentation, action
TEA	Adaptation of mobile phones	identity	presentation, action
Netman	Support for maintenance staff	location	selection
Gaia	Smart Environment	location	selection, presentation
iROS	Smart Environment	location	selection, presentation
Aura	Smart Environment spanning context management	location, identity	selection, presentation, action

Table 2.1: Examples of context-aware applications

Table 2.1 summarizes the properties of the presented examples of context-aware applications. The individual way, in which the context information is stored, retrieved, managed, and used by applications, is not discussed so far. The next section classifies the system support for context-aware computing and discusses related work as well as the contributions contained in this thesis.

2.2. Classification of System Support for Context-Aware Computing

A general model for the relation between the physical world and context-aware applications is depicted in Figure 2.1. Context information is formed by the current state of an application as well as from the states as they are present in the physical world, such as a user's position, services in the proximity of a user, etc. The context model as shown in Figure 2.1 separates applications from the process of sensor processing and context fusion. Moreover, this allows a number of applications to share the gathered context. Note, that this is a conceptual model and depending on the underlying system, i.e., based on an infrastructure or ad hoc network, and the way applications make use of the context information the instances of this model may differ.

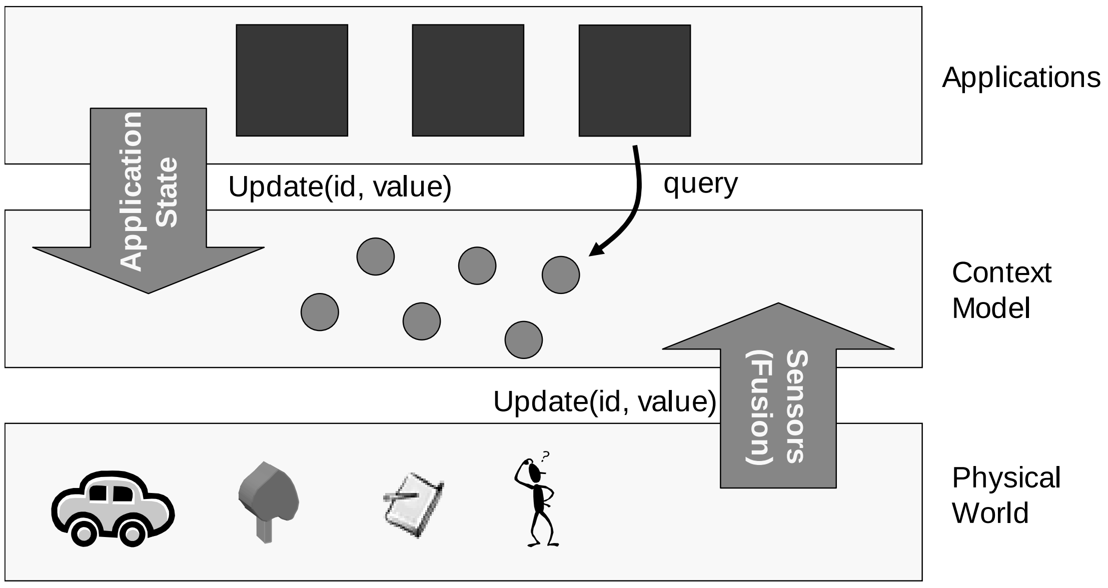


Figure 2.1: Context Model

For example, context can be managed by applications without providing means for sharing. A number of such applications will manage their local context models. Other applications may not build an explicit context model but directly access sensor information and process the obtained context information directly.

In the remaining part of this chapter we assume context information to be available via appropriate representations, such as a context model. We classify the system support along the underlying system – infrastructure or ad hoc – and along the support for application adaptation. Based on these dimensions four classes of context-aware computing are discussed (cf. Table 2.2). Requirements on system support are derived and related work is discussed. At the end of the chapter, the contributions of this thesis are summarized.

The requirements on system support for these classes are presented followed by the related work. The chapter closes with a discussion of the contributions to the classes of system support for context-aware computing contained in the following chapters of this thesis.

	Adaptation by System	Adaptation by Application
Infrastructure	*Smart Environments* iROS, Gaia, one.world	*Infrastructure-based Context Service* Aura CIS, Nexus, Context Toolkit
Ad hoc	*Peer-to-Peer Pervasive Computing* PCOM, P2PComp, MIT Pebbles	*Ad hoc based Context-Services* Usenet on the Fly RCSM, GLS, Nexus

Table 2.2: Classes of context-aware computing

2.3. Dimensions

The following dimensions are introduced to classify system support for context-aware computing:

Infrastructure-based system

Services that are required by context-aware computing form an underlying infrastructure. Examples are context services as they are provided by Aura [GSS+02] or Nexus [HKL+99] or smart environments, such as iROS [JFW02] or Gaia [RC00]. Applications typically require permanent access to the infrastructure.

Ad hoc system

In contrast to infrastructure-based systems, where the connectivity to the infrastructure is a prerequisite, ad hoc approaches do not require such services. Devices are spontaneously connected – typically by some wireless communication technology – and share their functionality or information. Examples are information propagation, such as the Usenet-on-the-Fly [BBH02], or Peer-to-Peer based approaches to Pervasive Computing, e.g., BASE [BSG+03] or PCOM [BHS+04].

Adaptation of applications has been recognized as a must for mobile application in general [Sat96]. The taxonomy provided by Satyanarayanan differentiates between no system support (laissez-faire) and application transparent adaptation. In the first case, applications have to decide which adaptation actions should be taken without any system support. In the course of this thesis we to refer to this class as *adaptation by application*, since we do not assume system support for adaptation decisions, but system support for accessing adaptation-relevant parameters, i.e., context information. The second case means, that the system adapts an application transparently according to changes in the context. Thus, we further refer to this class as *adaptation by system*.

Adaptation by Application

Context-aware applications adapt to changes in context. Based on a context model (cf. Figure 2.1) an application can retrieve information about context or gets signaled when a relevant context change has happened. If the adaptation decision is taken by the application, each relevant constellation of context parameters has to be reflected in the application code.

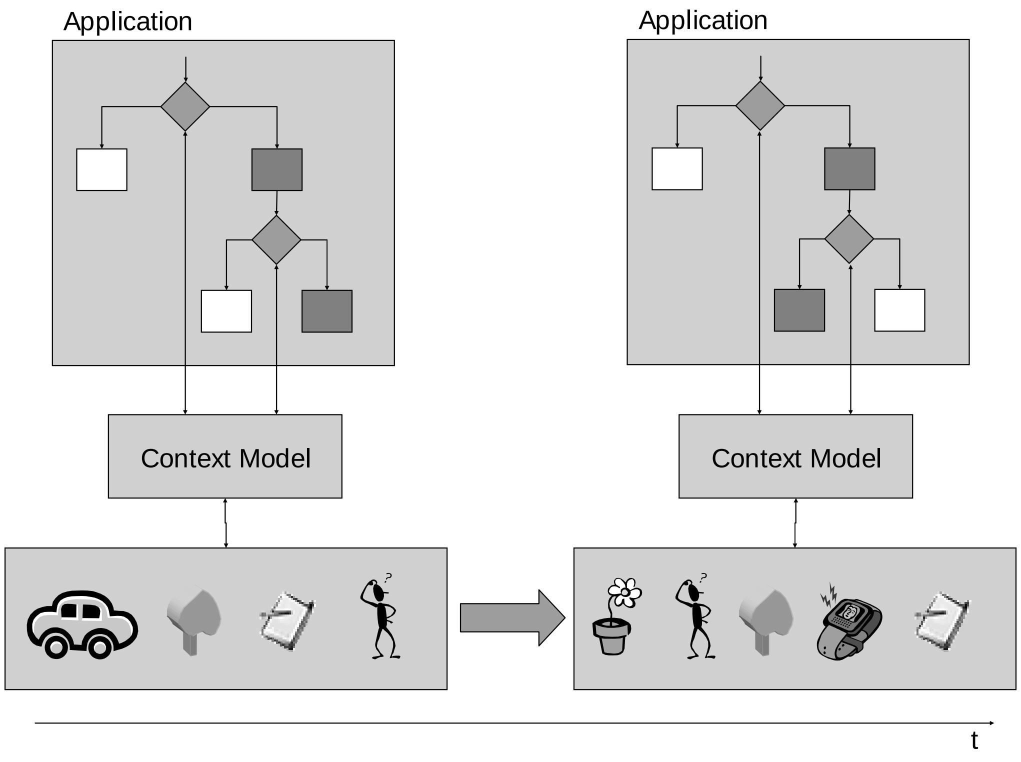


Figure 2.2: Adaptation by application

The resulting architecture is depicted in Figure 2.2 where a change in the underlying context is observed by the application based on a context model. The change of the application behavior is directly reflected in the application's internal structure. The application is depicted as a flow diagram, where the decisions in the flow are depending on the current context. The selected branches of a decision are shown by a greyed box, whereas the white boxes indicate, that a branch is not being taken. Application programmers can design their applications to react to context changes in the desired way with full control of the adaptation decision.

There is no required application architecture or framework which may restrict application programmers. On the other hand this means that every change of context leading to an adaptation has to be reflected in the application leading to an additional overhead in programming context-aware applications.

Examples for system support in this class are context services or frameworks which supply context information to applications. The Nexus platform [HKL+99] provides applications with context information without assuming an application architecture so far. Applications query context information or register spatial events in order to receive notifications on a defined predicate on the context model. The context toolkit [SDA99] does not maintain a context model but allows applications to connect to sensors or entities which aggregate context information. Based on this context information an application can choose whatever action it takes to adapt.

Adaptation by System

The effort involved in dealing with changes in context has lead to another class of system support, where the system analyzes the context and triggers reconfigurations of the applications according to the context and the application structure. The overall objective here is to relieve the application programmer from explicitly programming adaptation decisions.

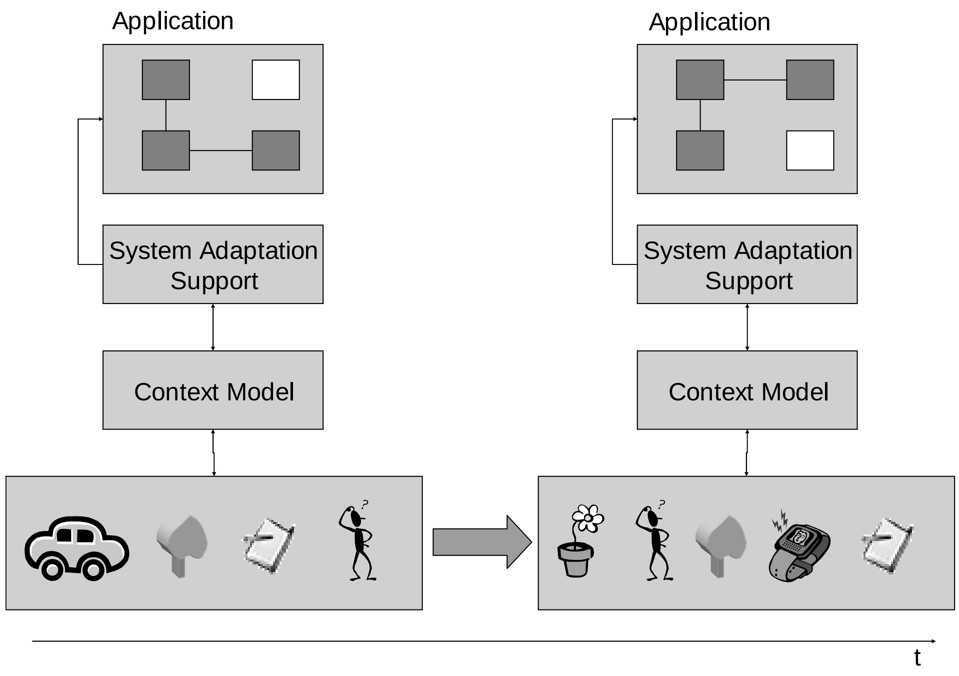


Figure 2.3: Adaptation by system

Applications are composed from building blocks, which are configured by the system according to the context and some other information that allows to determine the correct configuration of an application with respect to a given context. The building blocks of the application do not adapt to context changes. Figure 2.3 depicts the situation where the context change leads to a different configuration of the system. Again, the greyed boxes indicate that a building block was selected in a configuration and the white boxes depict building blocks not chosen.

Examples for this kind of system support can be found in smart environments. The Gaia [RC00] programming model allows mapping an application to a so-called active space by supplying scripts which assign parts of the application to devices and services available in an active space. iROS [JFW02] supports building blocks with larger granularity. Applications are composed of independent parts. The availability of an external functionality is supported by requesting this service via an event heap which dispatches the request to a suitable service. Applications are only aware that a request may not be answered when no suitable service is present in the environment. Another example from spontaneous networks based on ad hoc networks is PCOM [BHS+04], which supports fine granular application adaptation by the system based on a component-based application model.

Note, that a combination of application and system supported application adaptation is possible. A system could configure an application from building blocks, where some of these building blocks could change their behaviour in an application specific way, i.e., by reconfiguring themselves based on context information.

2.4. Classes

Based on the introduced dimensions we can now briefly classify system support for context-aware computing along these dimensions. General requirements on system support in these classes are presented along with related work. The contributions contained in the following chapters of this thesis are discussed with respect to the requirements and existing approaches.

The introduced dimensions classify context-aware computing along the underlying system, e.g., based on an infrastructure or on ad hoc networking, and the adaptation support for applications, i.e., the systems adapts an application to context changes vs. the system offers the necessary context information for an application, which can then adapt itself.

The first two classes are characterized by their support for application adaptation.

Infrastructure-based adaptation by system: applications in this class are automatically adapted by the system if relevant changes in the context occur. Since the infrastructure already provides a comprehensive set of services, temporarily present services and devices are integrated and the application execution is adapted in order to make use of this functionality. Typical candidates in this class are smart environments which are tailored towards a distinct application class in a spatial area augmented with specialized computing devices, e.g., a meeting room, and supply a set of services and devices as execution environment. Hence, applications face a rather static set of predefined services and devices which can be extended, e.g., by the integration of mobile devices user carry. Adaptation support thus means that a given application has to be mapped onto the available services and devices. Further adaptation support can provide means for the integration of dynamically changing services and devices, such as users' mobile devices. In general, adaptation support requires knowledge about the application structure and the execution environment in order to allow the system to adapt the application.

Ad hoc adaptation by system: in contrast to infrastructure-based approaches the underlying ad hoc networks reveal a higher dynamics of change in the execution environment. Due to user mobility and wireless communication the services and devices available to an application can change over time. Application adaptation support requires - similar to infrastructure-based application adaptation by system - knowledge about the application architecture. The management of the spontaneous group and available services and the resulting dynamism is the major difference between the ad hoc and infrastructure-based approaches. Examples in this class are P2PComp [FHM+04] and 3PC [3PC] which allow the composition of applications dynamically from the functionality available in a spontaneous network based on ad hoc communication.

Infrastructure-based adaptation by application: application adaptation offers a higher flexibility than automated adaptation by the system, since the application can choose its behaviour according to context. This means, that the system support has to provide access to context information by the application. In contrast to application adaptation by the system the applications are not bound to a given application architecture, since the system does not adapt the application.

Typical examples are context management platforms, e.g., context services, which maintain context information and offer query interfaces to applications. The Nexus platform [HKL+99] and the Aura Context Information Service [JS03] are examples for such context management platforms.

Ad hoc adaptation by application: providing context information to applications in ad hoc based environments in order to allow these to control their adaptation decision is similar to infrastructure-based adaptation by application. The resource restrictions on the participating mobile devices and the underlying network characteristics which lead to network partitions result in different needs for organization. As a result, the context information managed in such a setting differs from infrastructure-based context services. Typical examples are location services for mobile ad hoc networks, e.g., GLS [LJD+00] and DREAM [BCS+98], which only offer limited context information, i.e., the position of mobile objects. The absence of a central component or infrastructure requires appropriate dissemination mechanisms in order to disseminate context information from the place where it occurs to the place where applications request it, e.g., by data dissemination algorithms, such as SPIN [HKB99] for connected networks with less frequent network partitions or the one presented in [HBR03] which has been designed for frequently partitioned networks. Applications which make use of the context information are provided with an interface to the context management and are shielded from the underlying protocols and data management strategies.

2.5. Requirements

The discussion of the classes of context-aware computing support has shown that adaptation by application, e.g., via context services, in general decouple applications from the underlying system, i.e., infrastructure-based or ad hoc. However, further restrictions, such as limited memory of mobile devices, influence the context data provided in its size or level of detail. We will discuss the general requirements on context services independent of the underlying system model.

Similar to context services, the support of adaptation by system share many requirements independent of the underlying system model. The underlying system model may shift priorities between requirements, e.g., adaptation becomes more important in settings with higher device fluctuation, but in general the same requirements apply.

2.5.1. Adaptation by application

Adaptation by application is assisted by system support via context services, which provide the necessary context information applications can base their adaptation decision on.

General requirements context services have to fulfil are concerned with the representation of context information, supported queries for applications to access the context information, and the spatial layout which determines the underlying model of the part of the physical world reflected by the context model.

Context representation

Context representation obviously first depends on the context modeled. Context can be classified along its sources which already determine properties of context representation. Context information with low update rates, such as street networks, building floor plans, or 3-dimensional models of buildings can form realistic models of the physical world. More dynamic context information, such as information obtained via sensors for positioning or temperature, typically reflect a single aspect of the physical world which is captured by a sensor. So far, there is no standardization effort capturing context representation at its whole. However, for single domains standards exist or are beginning to emerge. Road networks for instance are available in different formats, such as the geographic data file GDF 77 or ATKIS [VGH+02]. The OpenGIS consortium started to define data emitted by sensors via SensorML [SensorML] an XML-based language capturing sensor properties such as the dimension, accuracy, and spatial relevance.

Even if standards and suitable representation of context information existed in each single domain, there are still open questions concerning the integration into a common context model:

- Query languages and semantics: accessing context information in a possibly application spanning way requires suitable languages serving a broad range of applications. Service models such as push and pull models should be supported. Indexing context information along the primary context has to be supported (see below).
- Multiple representations: if multiple applications and context sources feed their data into a context model multiple representations of an object may exist. The context model has to provide concepts to deal with such phenomena, e.g., choosing one representation, combining them, or prompting the user.
- Common semantic: the interpretation of context data across applications requires a common semantic. Examples are a common type schema or ontology.
- Context-specific management: the management of highly dynamic data, e.g., position information of mobile objects, requires different

handling than that of stationary objects or objects with low update rates, such as road networks. In addition to that, the organization of a context model should allow for queries incorporating the primary context, i.e., identity, location, and time.

In the following we will briefly discuss queries and service models of context services and their spatial organisation before context models are classified and existing approaches are discussed.

Queries

As noted before, queries to a context model should support the selection of entities based on primary context as depicted in Figure 2.4. Depending on the application requirements, not all queries have to be supported. If access to context information of the past of future is not an issue, the context models only store the current state reflecting the present time – or the time where the context model has been captured - based on state from the physical world.

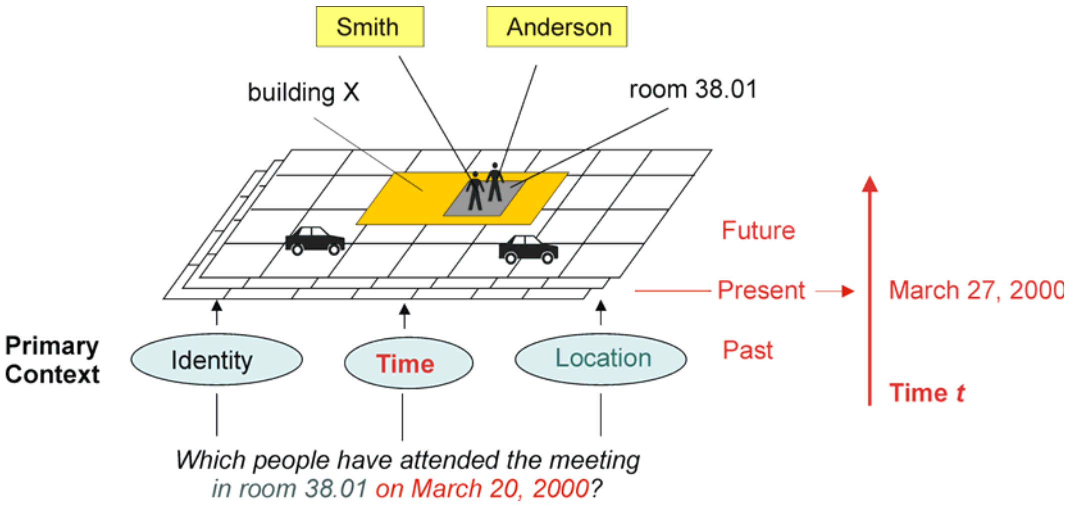


Figure 2.4: Queries to a context model

The service models supported should not only allow for queries in the pull-model but also allow for asynchronous communication. Spatial events [BR04] are an example where applications are notified about changes in the context and receive a notification. A spatial event is defined by a predicate which operates on context data. This allows to raise actions based on changes in the context model which are typically triggered by state changes in the physical world.

Spatial organisation

One important aspect of context information is related to location. This includes the position of entities as well as the spatial relation to other entities. Such relations cover the inclusion in a distinct area or range and the distance to other entities. Typical queries a context management platform should support with respect to location are according to [BD04]:

- Position: retrieve the position of an object. Examples are "where is John", "What is the position of printer PHP13".
- Range: a number of objects which are located in a spatial range are retrieved. Examples are "What objects are on Floor 2 of the Computer Science Faculty Building" which includes all objects in the rooms on the second floor as well.
- Nearest Neighbor: these queries offer a list of one or more objects which are closest to the position of an object. Queries for the next printer, restaurant, gas station thus become possible.

Although these queries at first seem simple and obviously necessary for a variety of context-aware applications, their efficient processing depends on the underlying spatial structure and the involved coordinates by the position information. Position information is obtained by positioning systems which track mobile objects and report their position to a location management system. In general, two kinds of coordinates are supported by positioning systems:

- Geometric coordinates: represent points or areas in a metric space, such as WGS 84 coordinates of GPS which represent the latitude, longitude, and elevation above sea level of mobile objects. Using geometric functions such as the Euclidian distance allows calculating distances and allows for nearest neighbor queries. Overlaps of geometric figures can be used to specify ranges by their geometric extension and determine whether ranges are included in each other which allows for range queries.

- Symbolic coordinates: in contrast to geometric coordinates there is no spatial relation offered by symbolic coordinates. Such coordinates are represented by an identifier, such as a room number or the ID of a cell or access point in wireless telephone or local area networks. In order to allow spatial reasoning about inclusion (for ranges) and distances (for nearest neighbors) explicit information about the spatial relations between pairs of symbolic coordinates has to be provided.

Location models are used to define spatial relations between locations. In general, locations can be determined by a symbolic identifier but also by a geometrically defined location. The latter allows expressing spatial relations which are not covered by the underlying metric on geometric coordinates. Consider a road network where people are bound to the spatial restrictions of the physical world. A geometric distance may be misleading, e.g., when a user has to cross a highway and another object may be closer from a user's perspective. Thus, location models are of use for geometric coordinates as well.

Choosing a suitable location model for the spatial structure of a context model is important for two reasons. First, the possible spatial queries along the primary context location depend on the location model. Second, the integration of two or more context models has to provide a mapping from one location model into another in order to allow spatial queries across the objects with possibly different location information provided by different positioning system as basic coordinates or different spatial relationships modeled in graphs or hierarchies.

Influence of system model

The general requirements on context services so far did not take the underlying system model into account. Clearly, the influence of the involved end-systems and their network connection affects the context information that a given system can maintain. Resource-restricted mobile devices will not be able to provide highly detailed three-dimensional models of larger spatial areas. Therefore infrastructure-based approaches are more appropriate to handle context for larger scopes and with higher complexity. However, an infrastructure-based context service may only serve a single house and thus scalability is not an issue there, e.g., the Aware Home Spatial Model Server [LBB+04]. Other context services, such as the Nexus platform, aim at potentially global scope context management, where the scalable integration of new context servers is a must as well as efficient query processing. If a context service allows for the integration of new context servers, a common underlying context model is required in order to integrate the context data. If such a platform should be open to new context data, extensibility of the context types stored and maintained is required.

In contrast to infrastructure-based context services there are some natural limitations in ad hoc systems which do not allow for larger scopes or higher complexity of context data. The memory limitations of the typically mobile and battery powered end systems along with the energy required for communicating larger amounts of data allows only for restricted context models, such as location information of mobile objects or some aspects of the physical world, e.g., floating car data[1]. Requirements in such settings are on suitable data organisation and data dissemination

[1] Note that energy typically is not an issue when propagating floating car data between vehicles.

algorithms. Typical trade-offs which have to be tuned to the operational environments have to deal with the freshness and availability of information versus the communication overhead of the underlying data dissemination protocols.

2.5.2. Adaptation by System

Application adaptation by the system first needs some kind of application knowledge in order to determine the possible configurations of an application. Furthermore, resources required for these configurations have to be managed and finally, adaptation decisions have to be made. Applications in such settings combine the resources available in a spatial area. Typical scenarios are smart environments, where rooms or buildings are equipped with a central control which mediates the resources in the environment, e.g., Gaia [RC00] or iROS [JFW02]. These projects reveal following the characteristics:

- Service oriented: in contrast to context services, that provide information about context, as they are addressed in adaptation by application system support, adaptation by system addresses applications composed of services. These services serve as building blocks for the adaptation support of the system.
- Dynamic composition: the available services and information in an execution environment are used by applications. Applications are not considered to be self-contained. Instead, they integrate available services and information and typically require a distinct set in order to get executed.
- Fluctuation of service availability: due to user (and thus device) mobility and other factors, such as device power down on drained batteries, the number of devices in an environment may change unpredictably. Also, device capabilities like sensors may be

temporarily unavailable, e.g., a GPS sensor stops operating when a user enters a building.

- Spatial relevance: the information and services which are used by an application typically only have relevance in the proximity of the user. This kind of context-awareness is reflected by the organization of an execution environment, e.g., smart environments mediate between applications and the services in a given spatial area. Ad hoc systems reflect the spatial relevance by the organization of services reachable via the underlying ad hoc network typically by restricting communication between involved devices to one or more hops.
- High number of devices/service: the integration of embedded systems into nearly every object of our daily life leads to situations where hundreds of services are available in a single room.
- Heterogeneity: the specialization of devices with respect to the offered services, e.g., a powerful computer that only serves as presentation service, and the miniaturization and thus restrictions on computing power as well as memory along with other resources, lead to an increased heterogeneity compared to classical computing environments.
- Administrative domains: the pervasion of our daily environment with pervasive computing nodes will lead to a multitude of administrative domains which may overlap depending on the roles the participating user fulfils. A janitor may access all relevant parts of a facility management system whereas the employees of competing companies in the same building will be restricted in their access to the systems of their respective companies. A user will seamlessly connect and disconnect to a variety of administrative domains and use the services and information

available to him while moving on through his work or recreational day.

Based on these characteristics of application adaptation by system, which are also common for the area of Pervasive Computing, we can derive the following requirements on system support:

- Spontaneous networking: devices should allow the dynamic networking with other devices. Especially, the integration of new devices as well as the leaving of other devices has to be supported.
- Adaptation by system: the constant change of devices in the execution environment leads to services entering and leaving the environment as well. Hence, applications have to adapt to the resources available in order to continue their execution when a resource is no longer available or to improve the performance when a better resource becomes available. The number of possible combinations of resources in such dynamic settings along with the complexity of programming adaptive applications leads to the requirement of adaptation by system.
- Interoperability: the heterogeneity of devices will lead to a number of interoperability protocols from sensor to complex application specific protocols. In order to allow the seamless integration of all of these devices interoperability protocols and bridging between them is necessary. Additionally, the system software has to be available on all devices or provide means for integrating devices into the environment.
- Security: attacks on such systems can have severe impact not only on the data stored but also on the physical environment. Motion detectors can be used to check whether a house is occupied, actuators can be tampered with and result in damage in the house – consider a frozen heating system due to a vandalizing attack

shutting down the system. Thus, environments build by adaptation by system, such as Pervasive Computing environments, have to provide means to secure the system in the presence of dynamic integration of devices and restricted resources.

Influence of system model

System support for adaptation by system shares the same requirements independent of the underlying system model. However, priorities of the requirements shift depending on the system model. Spontaneous networking is an issue in both cases. Infrastructure-based approaches, e.g., smart environments, have to provide means to integrate devices dynamically as well as to handle their exit. The fluctuation of services available to an application in an ad hoc system will likely be higher than in an infrastructure-based approach. Thus adaptation becomes even more important. This is made harder since the devices are typically mobile and battery powered. The state for adaptation decisions, e.g., dependencies between services or resource conflicts, is distributed among the devices in a peer group formed by the underlying ad hoc network. Infrastructure-based approaches can gather the state at a central instance and complex adaptation decisions can be placed on nodes with suitable computing performance and most likely powered by a constant power source.

The class of applications in such settings also may differ. A smart environment can be tailored to support distinct application classes, such as a smart teaching room [JFW02]. Peer-to-Peer based applications typically can only rely on more generic sets of services since their execution environment can not be determined beforehand.

2.6. Related Work

The related work is discussed for context management platforms first, followed by smart environments and Peer-to-Peer based Pervasive Computing.

2.6.1. Adaptation by application – context management platforms

Context models and their corresponding management architectures can be classified (cf. [RBB03]) along the dimensions of

- Spatial Scope: denotes the spatial area which is covered by the context model. This area can range from rooms in a smart environment over smart homes to global scope.
- Complexity of abstractions: refers to the level of detail and the details which are provided by the context model. Complex model could incorporate highly detailed 3D models of buildings whereas simple 2D models are commonly used, e.g., in navigation systems.
- Dynamism: the rate in which updates to information in the context model is supported typically depends on the provided complexity and scope. Cell-phone networks allow for high dynamism with respect to the managed position of mobile users but rely on a rather simple context model representing the position of users in terms of the cells their mobile terminal is logged in.

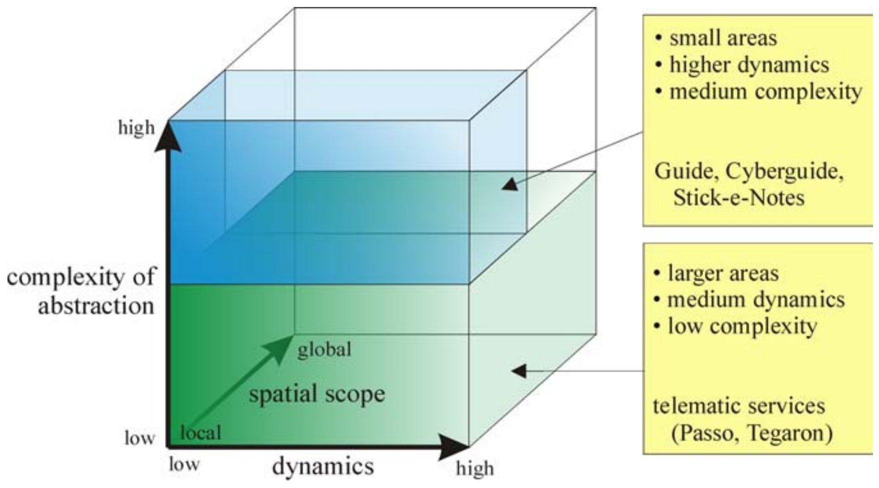


Figure 2.5: Classification of context models

Figure 2.5 depicts the three dimensions of the context model classification. Already existing context models from research and industry are either serving small areas with higher details and dynamics or larger areas but only with limited dynamism or complexity. To the best of our knowledge, only two projects so far address high detailed context information for larger scopes, namely Nexus [HKL+99] at the Universität Stuttgart and ContextWeaver at IBM Research [LSD+02].

Table 2.3 provides an overview of examples of context management platforms. The Guide project [CDM+00a] realizes a tourist guide for the city of Lancaster. The information is organized around locations which are modeled by the Wireless LAN access point deployed in the city center. The underlying context model is extensible in form of HTML extensions which assign information to a distinct location.

The context information service [JS03] of the Aura project [GSS+02] relies on the Aura context model which captures the relations (network connection, physical space) between entities (devices, people) from the computational and physical context, e.g., users and road networks in contrast to printers and network connections. Based on a simple meta model instances can be created and stored in a database. A SQL-like syntax for queries is provided.

The location stack [HBB02] is an example for a specialized context model which is created by combing different positioning systems and determines the position of mobile objects by sensor fusion.

A rather simple programming and data model is offered by the context toolkit [SDA99] which aims at capturing sensors and allowing the fusion of sensor data by combining them in a fusion architecture. Applications access context data by connecting to a so called context widget or interpreter which represent access points to a context source or fusion point. There is no explicit model supported and each application has to model its context model based on the obtained sensor information.

The REAL project [BKW02] mainly aims at the investigation of multi-modal user interfaces. Since the focus is not on context management, the underlying context model is not designed for extensibility. However, complex context representations and large scope is supported although the current architecture limits the scope by storing context information in a local database.

The Nexus project [HKL+99] aims at large scale context management supporting highly dynamic and complex context information. A federation is used to combine context models into one global spatial world model providing applications with a uniform view on the spatial world model. A standard class schema provides a common semantics across the federated context models. Besides context queries, spatial events are supported as well as context-aware communication, such as hoarding [BMR04] and geocast [DR03]. A deeper discussion of Nexus is presented in Chapter 6.

Recent research at IBM T.J. Watson addresses similar goals as Nexus. However, there are only aspects published, such as requirements [LSD+02], an event specification language [CLC+02], and an overall architecture [CPW+02]. The context representation and the semantics are not predefined by the system allowing application specific context to be managed.

Project	Specialization	Dynamic	Complexity	Scope
Guide	Generic HTML-like object structure	Extensible models	Cell-based granularity	City center
Aura CIS	Generic in the Aura context model	Dynamic (e.g., position information)	Hybrid location model (2D)	Not restricted
Location Stack	Tailored towards location management	Highly dynamic	Uses (some) complex models for fusion	Not restricted
Context Toolkit	Generic as long as widgets are provided	Dynamic, tailored towards sensor integration	No underlying model; mainly sensor abstraction	Not restricted
REAL	Fixed models for context representation	Rather static	Indoor: 3D; Outdoor 2D Models	Only restricted by local database
Nexus	Generic; an extensible class schema models semantics	Highly dynamic (sensor integration) plus static objects	Hybrid 2D-2,5D Models	Indoor, Outdoor; targeted at global scope

Table 2.3: Examples of context models

So far, we have discussed related work in infrastructure-based systems. There are so far only few examples for context services in ad hoc systems. The Nexus project starts to explore context management in ad hoc systems. Besides of that, there are mainly location services used for routing protocols representing adaptation by application in ad hoc systems, e.g., the Grid Location Service [LJD+00] or the location service of

DREAM [BCS+98]. Another example is the Usenet-on-the-fly [BBH02] presented in Chapter 4, which manages information propagation with spatial scope in an ad hoc network.

We will now present the related work in the dimension of adaptation by system split into approaches relying on an infrastructure, i.e., smart environments, and approaches in ad hoc based networks.

2.6.2. Smart Environments

Prominent examples for adaptation by system relying on an infrastructure can be found in Pervasive Computing, where smart environments manage the functionality of a distinct spatial area. The smart environment offers basic services for devices to register their services with the SE and to look-up resources required for their execution. This is typically allowed dynamically in order to support spontaneous networking. Interaction between the devices without the SE is not supported. Although this allows keeping the memory footprint on the participating devices small, communication and interaction is always mediated via the SE. This may lead to higher energy consumption than necessary if the device/service of interest is nearby and to performance bottlenecks. The larger the spatial area a SE controls gets the more devices interact. Scalability is an issue in this domain. However, most existing approaches either focus on a spatially restricted area – typically of room size – or support only a more or less fixed set of services and thus restrict the possible traffic.

Examples for this domain are Aura [GSS+02], Gaia [RC00], iROS [JFW02] to name a few representatives. In contrast to Aura the system model of Gaia, and iROS mostly address room-size smart environments. Applications adapt initially to the services available in the SE. Further adaptation is supported on different levels. Gaia offers an application model that is a variation of the model-view-controller pattern and mediates between the state, the presentation, and the processing of an application. Adaptation mechanisms can be applied to a coordinator component which may react to resource changes. An initial mapping of an application to a specific smart environment is provided by a scripting language. iROS introduces a coarse grained application model where the distributed parts of an application in a SE are basically self-contained applications that make use of additional functionality. A so called event-heap – a tuple space with an aging mechanism – allows applications to request services. The requests are purged via the aging mechanism when no suitable service is available.

Aura addresses context-aware applications for larger spatial areas than the aforementioned projects. A context information service [JS03] provides information about the physical space and the users as well as the computational entities and their network connection. Applications are formed by tasks which capture the user's intention and provide automatic adaptation by the system based on the information of the context service. The concrete evaluation of the user's intention and the mapping onto tasks is on-going research.

2.6.3. Adaptation by System in Ad Hoc Networks

Ad hoc based system support with application adaptation by system leads to Peer-to-Peer systems where nodes in the ad hoc network interact as equal peers. Reflecting the asks of a smart environment in a Peer-to-Peer (P2P) based organization means that there is no central control mediating the discovery, composition, and execution of an application. This class of systems seems promising, since the spatial relevance of information can be easily reflected by the typically spatially restricted wireless communication. There is no need for central directories and communication is not routed via a potential bottleneck. Interaction with services being open to the public can be easily established on a Peer-to-Peer base allowing users to access public services in different administrative domains, e.g., a floor-plan or indoor navigation system. However, security is also an issue in Peer-to-Peer Pervasive Computing, because there is no central control and many of the participating devices may not be capable of executing powerful encryption algorithms because of the involved effort in calculation.

To the best of our knowledge, there are only few projects contributing to Peer-to-Peer Pervasive Computing. The MIT Pebbles project is addressing a P2P based approach but requires a central instance to execute the planning algorithms for the assignment of tasks to devices. However, the project is still at an early stage [Pebbles] and the composition of an application relies on more than local knowledge and a central evaluation in order to map the application onto the devices in the P2P network. The Reconfigurable Context-Sensitive Middleware (RCSM) [YKW+02] is also contributing to Peer-to-Peer Pervasive Computing. In contrast to the MIT Pebbles, BASE [BSG+03] and PCOM [BHS+04], RCSM relies on a specification of context information which is used for service specification and selection. The work presented in [FHM+04] addresses similar objectives as BASE and PCOM. However, the support for restricted devices is limited since an existing component model (OSGi) is extended for spontaneous networking. The support for reselecting components is comparable to the adaptation of communication in BASE. Using application knowledge represented in contracts like in PCOM is not considered.

To sum up, context-aware computing has matured in the past years and a variety of concepts and architectures are being explored in different projects. However, there are to the best of our knowledge only few to none projects addressing ad hoc based support with adaptation by system so far, although this class of support for context-aware computing seems promising for a variety of reasons, e.g., the integration of applications into smart environments as well as interaction between devices in the absence of a smart environment. This thesis contributes in particular to this research area.

2.7. Contributions contained in this thesis

The contributions of this thesis address several classes of system support for context-aware computing as depicted in Table 2.4. Chapter 3 contributes to systems supporting adaptation by application by providing a thorough discussion about properties of location models, which are required for the underlying spatial structure of context models. Chapter 4 and 5 address support for adaptation by application in ad hoc systems. First, a novel application for information exchange in ad hoc networks is presented along with the underlying data dissemination algorithm. Second, an improvement of the underlying data dissemination algorithm in order to support frequently partitioned networks is presented. The integration of different context-models is the focus of Chapter 6 where the integration of Georgia Tech's Aware Home context server into the Nexus platform is described.

Chapter 7 to 10 address an important area of adaptation support by system in ad hoc systems: Peer-to-Peer Pervasive Computing. Requirements on system support, a middleware platform, a lightweight component model, and experiences are the contributions of these chapters.

In the remaining part of this chapter, the contributions of the following chapters are presented in more detail.

	Adaptation by System	Adaptation by Application
Infrastructure	*Smart Environments*	*Infrastructure-based Context Service* Chapter 3 Chapter 6
Ad hoc	*Peer-to-Peer Pervasive Computing* Chapter 7 Chapter 8 Chapter 9 Chapter 10	*Ad hoc based Context-Services* Chapter 3 Chapter 4 Chapter 5

Table 2.4: Contributions of this thesis

Chapter 3: On Location Models for Ubiquitous Computing

Location models play an important role for context services. The structure of the context model has to support relations between locations in order to allow for queries, such as position, nearest neighbor, and range queries. This chapter first provides an overview of application requirements and motivates why these kinds of queries have to be supported in order to support context-aware applications.

Properties of coordinates, symbolic as well as geometric, are discussed. The main contribution of this chapter is the discussion of different models for pure symbolic location models and the integration into hybrid location models, i.e., combined models which allow geometric and symbolic coordinates for the reference of locations. The discussed approaches to location modeling are set-based, graph-based, and hierarchy-based along with their extension to hybrid location models by attributing locations with geometric coordinates. The chapter also provides a classification of possible approaches to location models and their assessment along the criteria of supported queries and the involved modelling effort.

Chapter 4: Usenet-on-the-Fly: Supporting the Locality of Information

Support of adaptation by application requires some kind of notion about the relevance of information and services in order to allow applications to choose appropriate services and information. One natural way to reflect the local relevance of information is to restrict its dissemination to a distinct scope. The underlying assumption is that users are more interested in information and services nearby than in far-away ones.

This chapter contributes a novel application making use of the underlying ad hoc network characteristics. Information is exchanged between mobile nodes, which form a mobile ad hoc network, whenever two nodes are communication range. Users subscribe to information channels, similar to the UseNet, and the application synchronizes the local databases with nearby nodes. Such a system can be applied in many settings where information is of interest but not worth connecting to an infrastructure, such as a menu of the day of a restaurant or public transport schedules in the vicinity of a transportation platform. The information is collected by the underlying system and made available to the user by the application interface. Filters can be used to configure channels and topics to be synchronized.

The underlying three-way-handshake protocol is evaluated in order to show the feasibility of the approach. A prototypical implementation of the Usenet-on-the-Fly prototype based on the underlying data dissemination protocol is provided as well.

Chapter 5: Data dissemination in Frequently Partitioned MANETs

This chapter provides an enhancement of the data dissemination protocol used for the Usenet-on-the-Fly application in Chapter 4. The three way handshake protocol presented in Chapter 4 negotiates all information during its lifetime with other nodes. This can lead to a variety of problems in mobile ad hoc networks. First, bandwidth is consumed to advertise information to other nodes which already may have the information blocking other nodes. Second, battery power is consumed for sending advertisements. Since power is considered a precious resource in MANETs, an optimization of the dissemination protocol was pursued.

Advertisement messages are split into three classes. The classes relate to information depending on their freshness to the local node and their popularity to other nodes. The first class consists of all messages received until a distinct threshold in time. These messages are considered to be new and should be propagated to neighboring nodes. The second class is build from all messages beyond the threshold. They remain in the second class as long as their popularity is high enough. The popularity is increased whenever a neighboring node requests the data item after an advertisement. Messages in this class age into the third class if there was no request for a longer period of time. The third class of messages contains the history of messages received. Based on different strategies, e.g., randomly or round-robin, messages of this class are presented to neighboring nodes. This can help to propagate information across network partitions. If a message from this class gets requested by neighboring nodes it may be put into the second class again due to its increased popularity.

Chapter 6: Frome Home to World – Supporting Context-Aware Applications through World Models

Capturing and managing context information can be a time and cost consuming task. Obviously, sharing context information is a way to distribute the costs over multiple applications or at least to justify the effort. So far, there are little experiences in combining context models and assess where specific context models for a distinct domain can be reused in other domains or at least allow for an integration.

An initial assessment from two perspectives of context management is presented in Chapter 6. A context server from a smart environment domain – the Georgia Tech's Smart Home Spatial Server (AHSS) – was integrated into the federation of the Nexus platform.

Nexus	AHSS
Open Platform Support for arbitrary context-aware applications Common semantics via standard class schema XML-based query and modeling language	Tailored toward Aware Home applications No explicit semantic Interoperability via the Internet Inter ORB Protocol (IIOP)

Table 2.5: Objectives of Nexus and AHSS

Table 2.5 lists the different objectives of the AHSS in contrast to Nexus. While Nexus was designed a priori to support arbitrary applications with context information the AHSS was tailored to the needs of Aware Home applications. A use-case analysis showed that standard databases are sufficient for the context information necessary in the smart home. A spatial database was used to allow querying for location as primary index. No explicit semantic was modeled, since the members of the Aware Home project were assumed to keep track of the semantics of the individual applications. In contrast to that the Nexus platform provides a common class schema which provides a basic semantic across all context servers integrated in the federation. Extensibility of the so called standard class schema allows application specific extensions.

The semantic modeling of the Nexus platform is represented in its context modeling language (Augmented World Modeling Language, AWML) and its query language (Augmented World Query Language, AWQL) as well. The AHSS relies on SQL-queries which are dispatched to the AHSS via CORBA's interoperability protocol IIOP.

The integration of the AHHS into the Nexus platform logically means to register a context server that provides a spatial model for a distinct geographic area (the Aware Home's spatial extension) and provide all context types which are provided. This enables the Nexus federation to select context servers based on the geographic area and context types requested by applications.

Technically, this resulted in mainly two changes with respect to the integration of the context data and processing queries. A wrapper was provided which parsed queries from the Nexus federation in AWQL and mapped it onto the AHSS context model. Therefore, an integration of the data provided by the AHSS into the common class schema was necessary.

The experiences gathered from this experiment showed, that the integration of the AHHS into Nexus was possible without any changes in Nexus at all. The necessary wrapping of the AHSS with respect to the query processing and context model could be realized based on standard components available for the Nexus platform. The only task which – obviously – required a deeper understanding of the AHSS context model was the integration of the context data stored in AHSS into the common class schema. But if this task is done once, every extension in the AHSS' context model becomes available for Nexus applications as well.

Chapter 7: Middleware and Application Adaptation Requirements and their Support in Pervasive Computing

The classification of system support for context-aware computing has shown that the area of adaptation by system in ad hoc settings is only addressed by few projects so far. Chapter 7 identifies this research area and provides application scenarios and a system model. Based on these, requirements are derived and discussed with respect to their relation to middleware support or application support.

Namely, applications have to adapt whenever a service or a local resource, such as a GPS sensor, fluctuates in quality or availability. This leads to the requirement of uniform programming abstractions in order to provide a comprehensive framework for applications in order to adapt to these changes. The requirements on system software thus contain a uniform abstraction to applications for the access to remote services and local resources as well. Monitoring of local resources and device and service discovery has to be provided flexibly enough to support a variety of transport protocols and service discovery protocols along with different resource characteristics.

The most notable requirement stated is the decoupling of the communication model of application and interoperability protocol. Nearly all existing middleware platforms today reflect the communication pattern of the application, e.g., Remote Procedure Call, in the corresponding interoperability protocols, e.g., by using request-/response-messages over a the same communication channel. The characteristics of ad hoc networks impose problems here, since the spontaneous network connection between peers can break during interaction. However, many mobile devices are equipped with more than one communication module, e.g., infrared, Bluetooth, or 802.11. As long as there is one communication link between two devices, communication can take place. This requires switching the communication link and potentially the protocol stack of the interoperability protocol during an interaction, e.g., send a request via a SOAP stack using 802.11 and receiving the response via an event-based protocol on Bluetooth. Obviously, the communication model of the application stays a remote procedure call but the communication model of the interoperability protocol changes during interaction. This requires a flexible middleware architecture that provides advanced synchronization mechanisms. This middleware is presented in the next chapter.

Chapter 8: BASE – A Micro-Broker based Middleware for Pervasive Computing

As stated in Chapter 7, the area of adaptation by system in ad hoc systems is little explored. Chapter 8 picks up the requirements from Chapter 7 and presents a micro-broker based approach for a flexible and extensible middleware platform. The heterogeneity of devices with respect to their resources leads to the first design goal of minimalism and extensibility, in order to support resource-restricted devices as well as to make use of resources on more powerful devices. This is realized by using a micro-broker design. The micro-broker only provides mechanisms to dispatch so called invocations to corresponding plug-ins, synchronize the invocations according to the application communication model, and allow to install and remove plug-ins. This offers a small memory footprint, since only required plug-ins have to be installed. Additionally, the abstractions provided to the application are the same for dispatching invocations to device-local resources as well as to transport plug-ins which transmit an invocation to a remote device. The design of the plug-ins represents transport plug-ins as well as device resource plug-ins as invocation consuming entities. Possible results are sent back to the micro-broker as different invocations, indicating correspondence to the id of the prior invocation. This allows the micro-broker to synchronize invocations independent of the protocol used in the transport plug-ins. Moreover, the transport plug-in receiving the first invocation can be different from the one receiving the corresponding invocation, e.g., for a result. This allows the system to adapt to the availability of communication protocols between two devices without requiring the application to deal with the resulting problems, e.g., messages losses or blocking calls.

BASE meets the requirements stated in Chapter 7. The memory footprint of approximately 130 KBytes seems promising even on embedded systems. However, the abstractions provided to the application programmer require engineering adaptation support to single resources. Hence, an adaptive application has to provide adaptation support based on the resources it utilizes. Namely, a callback to handle unavailability has to be provided and in case one or more resources fail or become available the programmer has to provide logic in order to react accordingly. The different levels an application can run on based on the resources available are thus represented in the adaptation code. Clearly, this is no support for adaptation by system. In order to provide higher abstractions for application programmers, PCOM – a component system based on BASE – was developed, which is presented in the next chapter.

Chapter 9: PCOM – A Component System for Pervasive Computing

As mentioned above, BASE offers flexible support for interoperability in ad hoc networks. Context is represented by the spatial proximity of devices, which is reflected by their wireless communication. The communication between peers in such a network is automatically adapted in order to allow communication over arbitrary communication technologies and protocols. The programming abstractions of BASE require application programmers to reflect the configurations of an application explicitly in the code depending on the resources available in the current execution environment.

PCOM was designed to ease the implementation of applications that adapt to different configurations depending on the available resources. The key concept in PCOM is to represent the dependencies between components in an application explicitly. Based on this information the system is enabled to provide means for adaptation, e.g., when a component becomes unavailable another suitable component is searched for and - if available - integrated into the application.

PCOM relies on the concept of a component container which manages the lifetime of an application. An application is composed of components, which may interact across device boundaries, but are atomic with respect to their distribution, i.e., a component itself is executed in one container. Components specify their dependencies to other components and to the underlying computing platform via contracts. Contracts thus capture required components and resources of the device a component is executed on. An application is modeled starting from a root component along the dependencies to components required for the execution. An application is specified by the resulting tree of components along their dependencies.

PCOM containers are based on BASE and thus are able to discover themselves and communicate. The containers exchange information about the contracts they can offer to other containers based on the components installed.

Adaptation in PCOM is supported by three means. When a contract changes, i.e., a component contract breaks, a callback is called. In this callback three different options can be taken. First, the component which depended on the contract can stop its operation, breaking its contract, and thus escalating the handling up the tree. Second, the component can

choose to reselect the dependency to another component which is automatically handled by the PCOM container, and third, a component may choose to provide individual handling.
Together with BASE, PCOM provides system support for adaptation in highly dynamic environments, as they are present in ad hoc based environments, such as Peer-to-Peer Pervasive Computing.

Chapter 10 : Experiences – Extensibility and Flexibility in BASE

This chapter discusses experiences gathered when BASE was ported to a small embedded Java microprocessor and during the design and implementation of PCOM.
Initially, BASE was designed to work on resource-restricted devices. Based on the Java 2 Microedition the Connected Device Configuration (CDC) was chosen. Although this configuration already restricts many features of the Java programming environment, there are two features available that are not supported on the even more restrictive Connected Limited Device Configuration (CLDC), namely serialization and dynamic class loading.
The required changes during the port did not affect any conceptual design decision. Dynamic class loading could be omitted by providing a graphical editor to programmers to customize a BASE configuration. Automatic loading of required plug-ins is a feature which helps programmers in dynamic and resource-rich environments, but typical devices running the CLDC are fixed with respect to their resources. Hence, a static configuration can be provided. Object serialization is a powerful concept that eases the exchange of objects across devices realizing presentation layer issues. We are using a mechanism quite similar to the serialization concept provided by Java. Each object has to provide a serialization method which provides means to serialize the attributes of an object.

Even more convincing experiences were collected when designing PCOM on top of BASE. There were no changes necessary in the implementation or design of BASE. PCOM was realized as a service using the underlying abstractions for service and resource usage as well as the signaling mechanisms to indicate availability or unavailability of services and resources. There are only two modifications made in BASE in order to improve performance. The proxies for PCOM components directly inherit from the corresponding BASE proxies in order to avoid a call through an indirection layer. PCOM uses the BASE registries for remote devices in the execution environment directly and not through the proxy. This also saves one indirection through a proxy.

Most notable, the abstractions provided by PCOM could be realized in about 30KBytes leading to an overall size of 160 KBytes for BASE and PCOM.

3. On Location Models for Ubiquitous Computing

Common queries regarding information processing in ubiquitous computing are based on the location of physical objects. No matter if the next printer, next restaurant, or friend is searched for, a notion of distances between objects is required. A search for all objects in a certain geographic area requires the possibility to define spatial ranges and spatial inclusion of locations. In this chapter we discuss general properties of symbolic and geometric coordinates. Based on that, we present an overview of existing location models allowing for position, range, and nearest neighbor queries. The location models are classified according to their suitability with respect to the query processing and the involved modeling effort along with other requirements. Besides an overview of existing location models and approaches the classification of location models with respect to application requirements can assist developers in their design decisions.

3.1. Introduction

Location plays an important role in the domain of location-aware and context-aware systems. Especially in the ubiquitous computing domain location is commonly considered to be an important source of context [Sch95] but not the only one [SBG99]. However, whenever applications or users are interested in objects depending on their location or spatial relationship location models are required in order to provide notions about distances or ranges. This chapter presents an overview of possible approaches, discusses existing work, and classifies the approaches and existing work according to their suitability to allow for range and nearest neighbor queries.

Information about locations is presented in different formats. Geometric coordinates as they are used by GPS refer to a point or geometric figure in a multi-dimensional space, typically a plane or a three-dimensional space. The topological properties of such a space allow the calculation of distances between locations and their inclusion in other locations.

Symbolic coordinates on the other hand do not provide any reasoning about their spatial properties (distance and inclusion) without any additional information. Such coordinates are available via cell-ids in cellular networks, such as GSM or wireless LAN, as well as via other positioning technologies, such as radio frequency tags (RF ids) or infrared beacons.

Examples for the use of location information in applications are navigation services or location-based information systems, which select services based on their spatial proximity, e.g., the nearest printer, or notify when some events occur in the vicinity, e.g., a friend appears or an accident happens.

In order to allow such applications based on symbolic coordinates, a notion of spatial relations such as distance and inclusion is required. This information has to be modeled explicitly in a location model.

In this chapter we will discuss general requirements on location management and derive three types of queries – position, nearest neighbor, and range - which should be supported by location models. The properties of symbolic coordinates are discussed in general. Based on these properties different kinds of location models are discussed and classified along their suitability to support the queries.

3.2. System Model

Our system model consists of three kinds of components (cf. Figure 3.1):

The location model is the central part of our system model. It stores representations of static and mobile real world objects like representations of buildings and people, respectively. It is not the focus of this paper to describe how these objects are managed by an infrastructure, but we concentrate on the typical properties of the different kinds of location models. Examples of such location models are the Nexus platform [HKL+99, NGS+01], the context information server [JS03], or the guide project [CDM+00b].

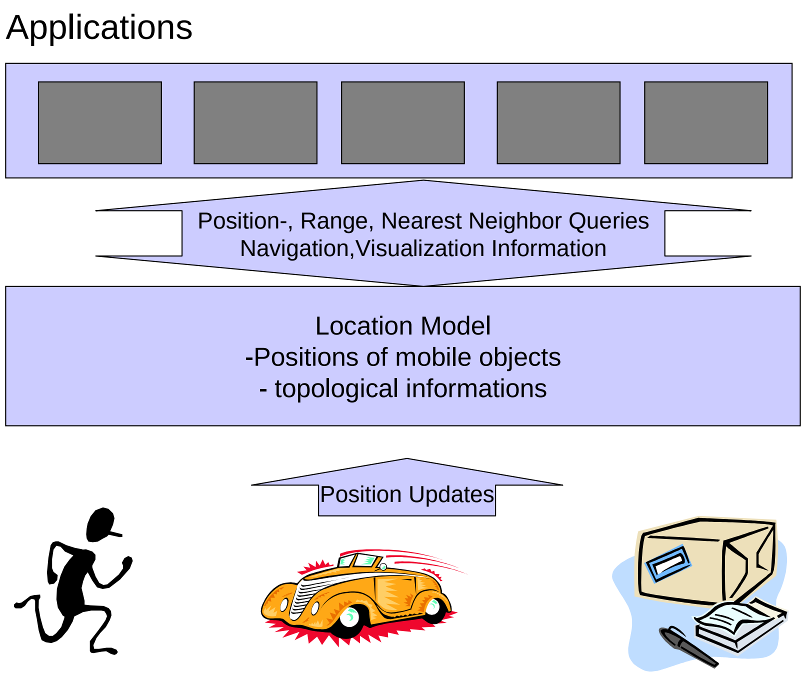

Applications
Position-, Range, Nearest Neighbor Queries
Navigation,Visualization Information
Location Model
-Positions of mobile objects
- topological informations
Position Updates

Figure 3.1: System model

Applications query the location model in order to carry out different tasks like navigation (see next section). They also update the location model, e.g., by inserting new objects into the model, deleting old objects, or by altering existing objects whose state has changed. For the context of this paper, we are interested in the different kinds of queries and tasks that are carried out by these applications because they determine the internal structure and organization of a location model. As will be shown later in this paper, the suitability of a location model for distinct queries depends on its internal organization. This is especially of interest, when a location model is not tailored towards a single application or domain but should manage information for a variety of applications and their potentially diverging requirements.

Positioning systems update position information of mobile objects like persons or cars. The output of these systems also influences the location model as we will see in the next section. However, the multitude and variety of positioning systems and its discussion is beyond the scope of this paper. For the remaining part of this paper we will assume that a positioning system allows a mobile object or tracking system to issue a position update with a coordinate identifying a location to the location model. This is sufficient for the discussion of the properties of location models. However, the interested reader can find an overview of different positioning systems in [HB01]. Fusion aspects of different positioning systems into a common location framework are presented in [HBB02]. In the following, a brief overview of the properties of coordinates as they are provided by current positioning systems is presented.

3.2.1. Basic Properties of Coordinates

A *coordinate* x is an identifier which specifies the position of an object with respect to a given *coordinate system*. A coordinate system is a set X of coordinates. Some examples for different kinds of coordinates and coordinate systems are:

- Geographic coordinates in the WGS84, used by the GPS, are expressed as triples containing the geographic longitude, latitude, and the elevation above main sea level.
- The Active Bat System [WJH97] is a high-resolution indoor positioning system providing three-dimensional coordinates – i.e., x, y, z value - with respect to a local Cartesian reference system.
- The Active Badge System [WHG92] provides symbolic identifiers for locations via infrared. Coordinates are the symbolic identifiers of the fixed IR sensors registering the users' active badges that transmit a unique identifier.

Two basic classes of coordinates can be identified from these examples: *geometric* and *symbolic coordinates*.

3.2.2. Geometric Coordinates

Geometric coordinates define positions in the form of coordinate tuples relative to a reference coordinate system. We further distinguish global and local geometric coordinate systems. The World Geodetic System 1984 (WGS84) is a global reference system and thus can be used to define coordinates anywhere on this planet, whereas the Cartesian coordinates of the Active Bat System are typically only valid locally, e.g., in one room equipped with such a system.

Geometric coordinates can be used to calculate the distance between two geometrically defined positions. Through geometric operations it can also be determined if two areas overlap, touch each other, or one area contains the other, i.e., topological relations like spatial containment can be derived from the geometry of objects. Hence, geometric coordinates already allow simple spatial reasoning.

3.2.3. Symbolic Coordinates

Symbolic coordinates define positions in form of abstract symbols, e.g., the sensor identifiers of the Active Badge system, or room and street names, etc. In contrast to geometric coordinates, the distance between two symbolic coordinates is not implicitly defined. Also topological relations like spatial containment cannot be determined without further information about the relationship between symbolic coordinates. Symbolic location models provide this additional information on symbolic coordinates.

3.3. Requirements for Location Models

In order to derive requirements on location models and discuss their properties with respect to the organization, we will motivate queries to location models from the perspective of users and applications. Besides position queries, which are obviously needed in location-based applications, the necessity of nearest neighbor and range queries is motivated. This will serve as foundation of the later classification of location models. The choice of a distinct location model will dependent on the queries required by applications. Therefore, we have to consider these queries and tasks in order to assess the functional requirements for location models.

3.3.1. Position Queries

The determination of the positions of mobile and static objects like users, buildings, bus stops, etc. is a common building block of location-based and context-aware systems. The tasks described below cannot be carried out without the known positions of objects. Therefore, all location models contain this information, but they differ in the way it is represented.

The definition of a position requires some form of *coordinates*. Based on an object's position actions can be carried out, such as teleporting the user's interface [WJH97], controlling the input and output of applications to arbitrary spaces in the physical environment via projection techniques [PPL+03], or in industrial settings, such as a smart factory [BJR+03], the positions of resources and tools can be monitored in a production planning system. Such systems require a common interpretation of the coordinates in a specific global coordinate system. Within moving objects, such as trains, local reference systems can help to address objects, such as travelers with respect to their compartment in the train and not their absolute position to the ground.

This shows that a general location model has to support *different coordinate reference systems, global and local* ones.
Beside well-known geometric coordinates, some positioning systems provide symbolic coordinates, e.g., the cell id in a cell-phone network or identifiers of infrared beacons, and often these symbolic coordinates can be interpreted more intuitively by users than geometric coordinates. Later we will show, how simple symbolic location models can be set up allowing for spatial reasoning with low modeling effort. Therefore, this kind of coordinates has to be supported as well.

3.3.2. Nearest Neighbor Queries

A nearest neighbor query is the search for the *n* objects closest to a certain position. For instance, a user can search for the nearest restaurant with respect to his current position, or the next printer. Beside known object positions, the definition of a *distance function* on the coordinates is required for this type of queries. For geometric coordinates, the direct physical distance between two positions can be calculated using well-known formulas like Pythagoras in Cartesian systems. If only symbolic coordinates are modeled then the model must contain explicit definitions of distances between these coordinates, e.g., to define the distance between room number X and the printers in the rooms number Y and Z, since symbolic coordinates do not contain a natural embedment into a metric space.
There are other notions of distance that are often more relevant than the direct physical distance. For instance, for a pedestrian it might be impossible to cross a highway. Therefore, a restaurant across the highway with a direct physical distance of 100 m might be farther away than a restaurant with 200 m direct physical distance not located across this highway. In these cases additional model information like the road

network a user uses to get from location A to B has to be taken into account. For such more complex nearest neighbor queries, this leads to similar requirements as for navigational tasks described in the next subsection, because "paths" between locations have to be found and their "lengths" have to be compared.

To sum up, a notion of "distance" is required in many context-aware or location-based systems. An explicit location model is required for symbolic coordinates as they do not provide implicit distance functions. Systems based on geometric coordinates can benefit from such a model as well, as spatial restrictions can be modeled, e.g., road networks.

3.3.3. Navigation

Navigation systems become standard equipment in nowadays cars. Such systems require a location model to find paths between locations. Possible paths are defined by the transportation network (roads, train or bus routes, etc.) and consist of several interconnected locations. This means, it does not suffice to know the geometry e.g., of roads, but it is also important to know how to get from one location to neighboring locations, e.g., from one road segment to another road segment at a junction, and finally to the destination. Therefore, the *topological relation "connected to"* has to be modeled that describes these interconnections between neighboring locations (cf. Figure 3.2).

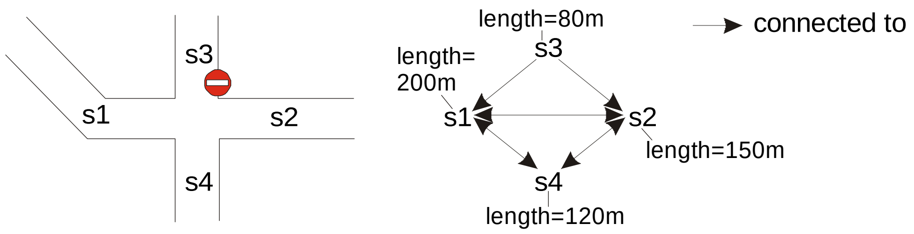


Figure 3.1: Road geometry (left) and road topology (right)

There are different kinds of navigational tasks, e.g., finding the shortest path or the fastest path. Finding for instance a suitable path for a person in a wheelchair requires additional information about locations, e.g., staircases or elevators. Therefore different attributes need to be modeled to implement these variants, e.g., the distance that has to be traveled to get from one location to another location, the maximum allowed speed on a road segment, the presence of stairs, which cannot be used with a wheelchair, etc. Even highly dynamic information like the current traffic situation on a road can be part of the model. In general, this means modeling some kind of weight on path segments. The "length" of a path is then calculated by summing up the weights of each path segment.

3.3.4. Range Queries

A range query returns all objects within a certain geographic area. It can be used for instance to query the occupancy of a room as well as for a check whether an evacuation plan is processed correctly, i.e., if a room is empty before the fire doors are closed and sealed. Also, simple algorithms for new types of communication can be implemented on the basis of range queries, e.g., geocast [DR03], i.e., the sending of messages to receivers in a certain geographic area. First, a range query can be used to determine all receivers in the target area of the message. Secondly, the message is sent to these receivers, e.g., using multiple unicast messages.

First of all, object positions have to be known to answer a range query. Additionally, the *topological relation "contains"* has to be modeled, i.e., it has to be defined whether a coordinate lies within a spatial area. For geometric coordinates, this information can be derived from the known geometry. But for symbolic coordinates, this relation has to be defined explicitly. For instance, a model can define that the room 2.062 is on ("within") the second floor that in turn is part of ("within") a certain building, etc. Thus, querying for a larger area automatically includes all objects from locations that lie within that area.

3.3.5. Visualization

Drawing maps is one of the most obvious application of location models. Maps can be used for many different tasks like positioning, navigation, etc., which we have already described in the subsections above. A map helps the user to execute these tasks manually or it is used to display the results of these tasks if they are carried out automatically. All model information introduced above can be visualized, but usually a map is drawn, which requires a more or less detailed *geometric representation* of these objects, depending on the desired level of detail (see below).

3.3.6. Requirements

From the use cases presented above, the following requirements for location models can be derived. Note, that not all of these requirements have to be fulfilled at the same time. However, being aware of the application requirements is crucial in order to choose the appropriate location model organization.

Based on position, nearest neighbor, and range queries it can be concluded that a location model should provide:

- *Object positions:* Positions of objects have to be modeled in form of coordinates. Supported coordinates and reference systems are
    - *Geometric and symbolic coordinates*
    - *Multiple, local and global coordinate reference systems*
- *Distance function:* Distances between spatial objects have to be modeled. This can also be the "size" of a location, e.g., the length of a road segment, which represents the distance one has to travel when crossing this location in order to reach another location.
- *Topological relations:* The following topological relations between spatial objects have to be modeled:
    - *spatial containment* in order to allow range queries, and

    - *spatially connected to* for navigation services.

Furthermore, the position of objects alone is not sufficient for some applications which also require the direction of a moving object or the orientation of a user, e.g., in order to provide information about the building a tourist looks at.

- *Orientation:* In addition to positions of mobile objects, the orientation in the horizontal and/or vertical dimensions can be supported.

These requirements have to be regarded in conjunction with the requirement of *minimal modeling effort*. There are different factors that influence the modeling effort:

- *Accuracy:* The model should describe the real world as accurately as possible, i.e., the stored information should be consistent with the real world. Accuracy is not a question of the model type but of how the model is created and updated and of the dynamics of the modeled objects: Highly dynamic objects require high update rates, e.g., highly mobile objects will have to update their position frequently to get accurate position information. These issues are not the focus of this paper, and therefore accuracy will not be considered any further.
- *Level of detail:* The level of detail describes the precision or granularity of the model. Fine-grained models describe locations down to room level or below; coarse-grained models stop at buildings or larger. A flexible model allows both ends of the scale.
- *Scope:* The scope is the area covered by the model. Local models may only describe one single room, whereas global models at the other end of the scale describe locations all over the world.

The two last items are intimately connected. Highly detailed models usually only describe small parts of the world, because they require high modeling effort; coarse-grained models may have a larger scope [RDD+03]. Also the architecture used to manage the model plays an important role for the level of detail and scope. A federation of highly-detailed partial models with limited scope can be used to extend the scope of the (federated) model and make highly detailed global models feasible [NGS+01]. In this paper we do not consider how location models are management, but we concentrate on the general properties of the different kinds of location models. The following discussion first addresses location models for geometric and symbolic coordinates. Then the integration of geometric coordinates into symbolic location models leading to hybrid location models is discussed. Based on this discussion a classification of the general approaches is presented and existing work is classified.

3.4. Geometric Location Models

Geometric models describe locations by geometric figures. If not only global coordinate systems are to be used but also local ones, the position and orientation of local systems with respect to other local systems or the global system has to be defined in order to translate coordinates of one system to other systems.

On the basis of geometric coordinates the topological relation "contained in" can be derived. In contrast to the containment relation, the "connected to" relation modeling e.g., doors connecting rooms cannot be derived from location geometries. This relation has to be modeled explicitly. If this information is modeled, it can be used to improve the notion of distances, e.g., by incorporating the distance a user has to travel in contrast to the direct distance reflected by the underlying geometry. However, it is also reasonable for a geometric location model to store the spatial containment relation explicitly since geometric operations are costly.

3.5. Symbolic Location Models

In this section we describe different types of symbolic location models and discuss their suitability for the different types of queries described in the requirements section of this paper. Set-based, hierarchical, and graph-based models are presented.

3.5.1. Set-based Model

A set L of symbolic coordinates forms the basis for the set-based approach. Locations comprising several symbolic coordinates are defined by sub-sets of the set L. As a simple example consider a building with several floors. The set L consists of all room numbers of this building. The second floor as shown in Figure 3.3 can be modeled by the set L_{floor2} = {2.002, 2.003, …, 2.067}. Further arbitrary locations may be defined, e.g., the locations A = {2.002, 2.003} and B = {2.003, 2.005} in Figure 3.3.

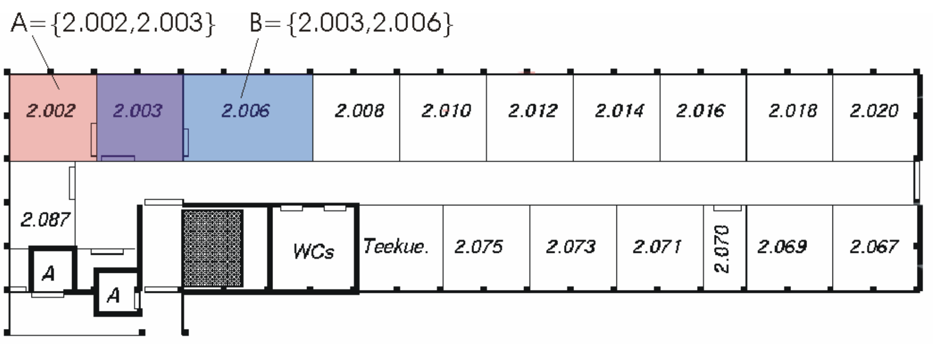


Figure 3.3: Set-based location model

This model can be used to determine overlapping locations and as a special case of overlapping locations the containment relation by calculating the intersection of two sets L_1 and L_2. If $L_1 \cap L_2 \neq \emptyset$, then L_1 and L_2 overlap. If $L_1 \cap L_2 = L_1$, then L_2 contains L_1. Thus, this model can be used for range queries where the range is defined by one set R of symbolic coordinates, and all sub-sets of R define locations within R.

This model can also be used to express a simple qualitative notion of distance between symbolic coordinates by modeling sets of "neighboring" symbolic coordinates, which we call neighborhoods (by L_{con} we denote the set of neighborhoods). For instance the sets *A* and *B* in Figure 3.3 as well as the set L_{floor2} defined above are such neighborhoods in L_{con}. Distances between the symbolic coordinates *x*, *y* and *x*, *z* are compared as follows:

$$\mathrm{d}(x,y) < \mathrm{d}(x,z) \Leftrightarrow \exists L_1 \forall L_2 \in L_{\text{con}} (x \in L_1 \wedge y \in L_1 \wedge x \in L_2 \wedge z \in L_2 \rightarrow L_1 \subset L_2)$$

That means, the two smallest neighborhoods containing *x*,*y* and *x*,*z*, respectively, define the distance from *x* to *y* and *x* to *z*. Consider for instance the three symbolic coordinates 2.002, 2.003, and 2.006. d(2.002, 2.003) < d(2.002, 2.006) because *A* (the smallest neighborhood that contains 2.002 and 2.003) is a proper subset of L_{floor} (the smallest neighborhood that contains 2.002 and 2.006 in our example). To achieve a fine distance granularity, neighborhoods can be defined for each pair of directly connected locations, e.g., rooms which are connected by a door. For instance, the locations *A* and *B* introduced above are such locations. Larger neighborhoods are defined recursively by joining smaller neighborhoods which have non-empty intersections, e.g., the neighborhood C = $A \cup B$. By modeling pairs of connected locations, also possible paths can be derived. A negative effect of this approach is the huge number of resulting sets and the involved modeling effort.

Beside this qualitative notion of distance, this approach does not permit to define a quantitative notion of distance, e.g., to make statements like "the distance between *a* and *b* is as long as the distance between *c* and *d*". Therefore, the support for queries related to spatial distances (e.g., nearest neighbor queries and navigation) is limited.

In contrast to set-based location models which do not contain explicit relations between locations, the following two models, i.e., hierarchical and graph-based, model relations between locations.

3.5.2. Hierarchical Models

Hierarchical models consist of a set of locations *L*. The locations are ordered according to the spatial containment relation, i.e., a location l_1 is an ancestor of a location l_2 ($l_1 > l_2$), if l_2 is spatially contained in l_1. If locations do not overlap each other this leads to a tree-based model [JS02]. If overlapping locations are to be modeled the more general lattice-based model is applicable where intersections of locations are modeled by separate locations with more than one parent location [KEG93, DR03]. Figure 3.4 shows an example of such a lattice-based model. The set of locations *L* consists of the building *B*, the floors $F_1,\ldots, F_m$, two wings W_1 and W_2, and several rooms $R_1,\ldots, R_n$. The locations F_iW_j denote intersections of the floor F_i and the wing W_j. Figure 3.4b also shows the relationship of the hierarchical models to the set-based approach. Locations in the hierarchy can also be interpreted as sets of symbolic coordinates. Overlapping locations are defined by the intersection of sets. Therefore, hierarchical models can be seen as a special case of set-based models.

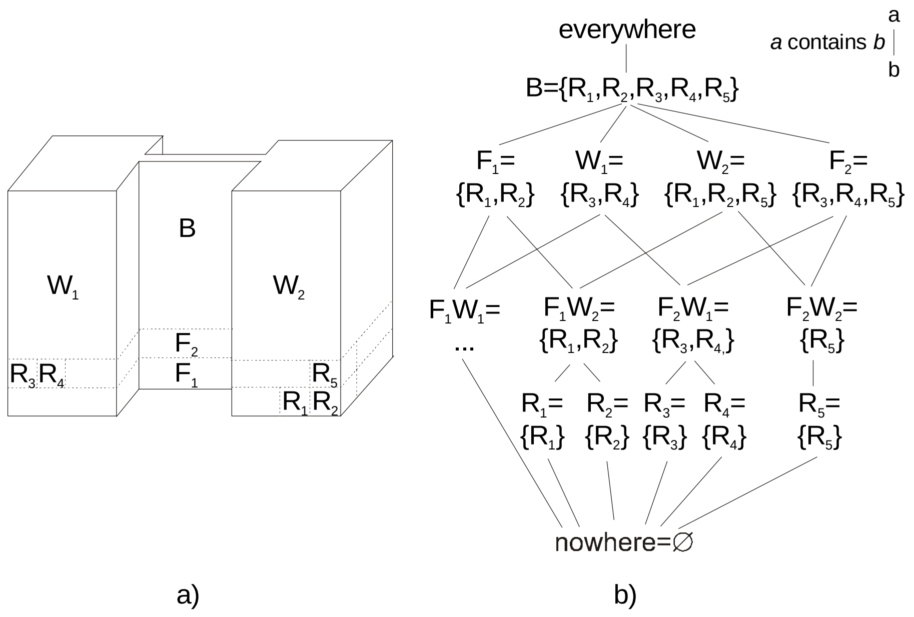


Figure 3.4: Hierarchical lattice-based location model

Because the hierarchical models are based on the containment relation they support range queries naturally. A range is defined by a location in the hierarchy and the descendants of this location denote locations within this range.

A simple notion of distance comparable to the one discussed in the previous sub-section can also be applied to hierarchical models:

Given three locations $l_1, l_2, l_3 \in L$. Then $d(l_1, l_2) < d(l_1, l_3)$, if $\sup(\{l_1, l_2\}) < \sup(\{l_1, l_3\})$.

$\sup(\{l_1, \ldots, l_n\})$ denotes the supremum (least upper bound) of a set of locations. For instance, the two rooms R_1 and R_2 located on the same floor and in the same wing in Figure 3.4 are considered to be closer to each other than the rooms R_2 and R_5, which are only in the same wing but on different floors ($F_1W_2 = \sup(\{R_1,R_2\}) < \sup(\{R_2,R_5\}) = W_2$). In some situations this interpretation of distance may be counter-intuitive. If for instance a short connection exists between R_2 and R_5, e.g., stairs, the R_2 could be closer to R_5 than to some room located on the same floor and wing as R_2. Hierarchical models provide no means to model interconnections between locations, and therefore this situation can not be handled adequately. As for the set-based approach, this notion of distance is also only qualitative.

3.5.3. Graph-based Model

In the graph-based approach, symbolic coordinates define the vertices V of a graph $G = (V,E)$. An edge is added between two vertices if a direct connection between corresponding locations exists. Edges or vertices can be weighted to model distances between locations. Figure 3.5 shows an example of a graph-based model for the already presented second floor of a building. In this example the distance between two coordinates is just the number of hops but with additional information a higher accuracy could be achieved. [Dro03] gives a deeper discussion of this aspect of graph-based models.

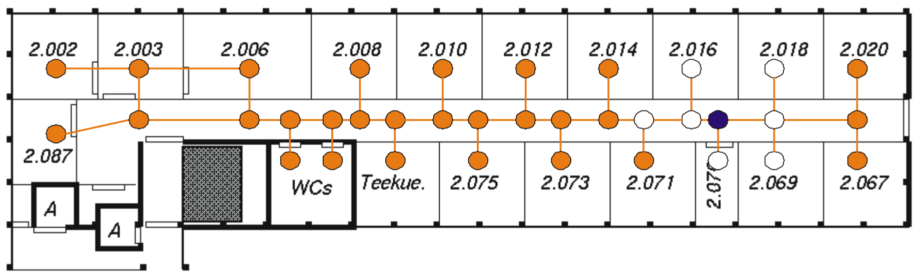


Figure 3.5: Graph-based model

From the construction of the graph it is already clear that a graph-based model supports the definition of the topological relation connected to as well as the explicit definition of distances between symbolic coordinates. It is therefore well-suited for nearest neighbor queries as well as navigation. For the latter the edges or nodes can be further attributed to model e.g., speed limits, vehicle restrictions, etc. [VHG+02].

For range queries first the range itself has to be defined, i.e., an area has to be described within which we want to search for included objects. The only locations which are explicitly defined in the graph-based model are the nodes of the graph, e.g., the rooms shown in the example above. This is surely a very limited set of ranges. Because the graph-based model allows to define a distance between symbolic coordinates this distance can be used to define ranges. That means, an object is in the area, if the distance between its position and a reference location is at most the radius of the area. In Figure 3.5 for instance the white locations are within the range defined by a reference location marked black and the radius 2, thus all objects at these locations are within this range. What we are missing is the possibility to *explicitly* define bigger locations comprising several smaller locations, e.g., a whole floor, building, or even parts of a city. In the next section we will show how this limitation can be overcome by combining the different types of symbolic location models.

3.5.4. Combination of Graph-based and Set-based Symbolic Models

Our discussion of the different location models has shown that for symbolic coordinates the graph-based approach supports queries based on distance and the definition of connected locations well, whereas the set-based approach can be used for range queries with explicitly defined locations like floors, building, etc. representing ranges. Therefore, a combination of graph-based and set-based symbolic locations models can be used to combine the benefits of both types of models.

The set-based part of the combined symbolic location model consists of a set of symbolic coordinates. Locations are sub-sets of this set of locations, e.g., representing rooms, floors, buildings, etc. This part of the model is used for range queries as described in the section about set-based models.

In the graph-based part of the combined model, locations are connected by edges if a connection between these locations exists in the real world. For instance, two rooms will be connected in the graph, if there is a door between these two rooms; two floors will be connected if stairs lead from one floor to the other, etc. As mentioned in the previous section, edges can also be weighted to model different distances. Figure 3.6 shows an example of the resulting combined model.

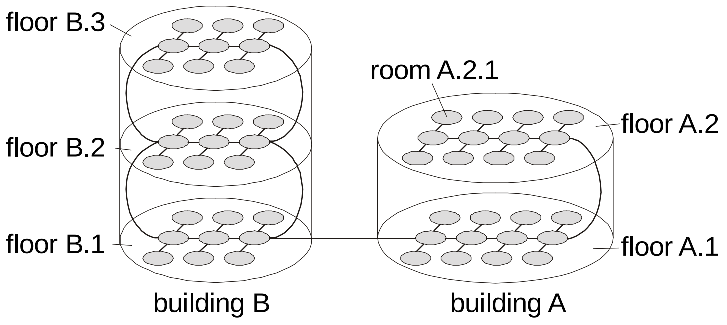


Figure 3.6: Combined symbolic location model

Besides the already mentioned support for different topological relations and distances and the range and nearest neighbor queries based on this information, this model shows another interesting feature. It allows to generate views with different levels of detail. Figure 3.7 shows three examples. The first example shows the rooms on one particular floor and their connections. This view will be used if a very fine granularity is required, e.g., if we are searching for the next printer. Figure 3.7b shows only the floors of building *A*. Floor A.1 and A.2 are connected because elements of Floor A.1 and A.2 have a connection – e.g., two hallways connected by an elevator. Finally, Figure 3.7c depicts only buildings and the paths between them. The latter could be used in a scenario where only coarse-grained location information suffices, and so it allows to generate small models that cover large areas, e.g., a whole city district.

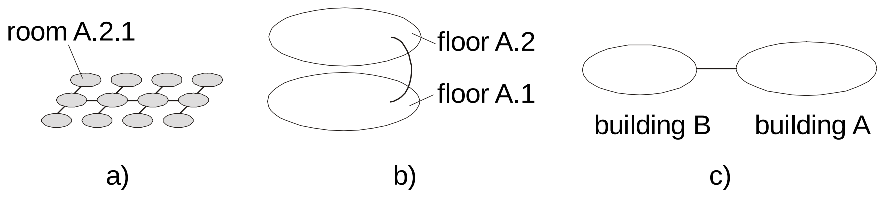


Figure 3.7: Levels of detail

3.5.5. Summary

We now summarize the properties of the different types of symbolic location models presented in this section.

symbolic model type	**supported coordinate types**	**modeling effort**[2]	**distance support**	**“connected to” relation support**	**containment relation support**
set-based	symbolic	high	limited	yes	good
hierarchical	symbolic	low to medium	very limited	no	good
graph-based	symbolic	low to medium	good to very good	yes	limited
combined (set-based & graph-based)	symbolic	medium	good to very good	yes	good

Table 3.1: Properties of symbolic location models

We see that the graph-based approach as well as the hierarchical models support the containment relation well, making them suitable for range queries. The graph-based approach is well-suited for all kinds of queries where distance plays an important role, e.g., nearest neighbor queries and navigation. The combined symbolic location model combines the benefits of all other symbolic model types at the cost of higher modeling effort.
Still the accuracy of the combined model can be further improved by adding geometric information. The next section presents different hybrid models, which integrate symbolic and geometric information.

[2] Modeling effort is always dependent on the granularity and scope of location information as stated in the requirements section. Therefore, we give a range here.

3.6. Hybrid Location Models

The combined symbolic location model presented in the previous section shows how the benefits of set-based and graph-based models can be integrated into a common symbolic model. There are two major arguments for additionally adding geometric information to such a symbolic model. First, geometric information can be used to achieve higher accuracy and precision for all kinds of distance related queries. Secondly, arbitrary geometric figures can be used for instance to define ranges for nearest neighbor queries, whereas symbolically defined locations are always restricted to a given structure.

We distinguish between two types of hybrid location models. The first approach, which we call the sub-space approach, stores geometric information for every modeled location. The second approach only stores geometric information for some locations, leading to partial subspaces.

3.6.1. Subspaces

The basis for this hybrid location model is a symbolic model like the combined symbolic model presented in the previous section. Additionally, the geometric extent of locations is stored in the location model. The geometric extent can be either defined using a global reference system like the WGS84 or local reference systems where coordinates are only valid within a certain scope, e.g., in one building or room. Subspaces are formed by embedding coordinate systems into other coordinate systems by defining the position and orientation of embedded systems (a detailed description of this embedding of subspaces can be found in [JS02]). With this information, coordinates can be translated from one system to other systems, and thus coordinates of different systems can be compared.

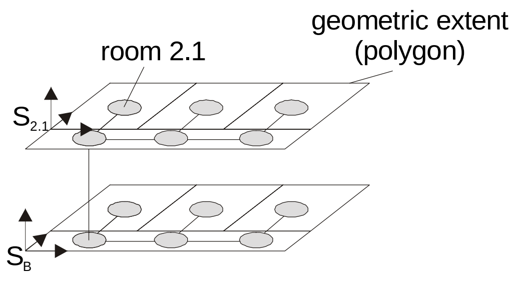


Figure 3.8: Hybrid location model with subspaces

Figure 3.8 shows a simple example of a hybrid location model using subspaces. The symbolic part of this model is based in a graph defining the interconnections between the rooms on a certain floor. The extent of every room is also modeled geometrically using the coordinate system S_B of the building B. Within room 2.1 a local coordinate system $S_{2.1}$ is defined that is embedded into the system of building B. The system of building B in turn may be embedded into a global coordinate system. The known geometry can be used to define precise distances between rooms.

3.6.2. Partial Subspaces

In contrast to the subspaces approach, the partial subspaces approach does not assume that the geometric extent for every location is modeled, but only for some locations. Figure 3.9 shows an example, where a geometric location model exists for the outdoor domain, but within buildings symbolic models are used. By linking geometric information to symbolic locations, the symbolic building models can be embedded into the global geometric model. The benefit of this integration becomes clear when we consider a range query with a geometrically defined range, e.g., a polygon drawn on a city plan. Users within a building may only know a symbolic position like room 2.1 in building B. Through the known geometric extent of the building, the user's position can be approximated geometrically with the geometry of the whole building. This approximated geometric position can be compared to the geometrically defined range of the query, and thus the query can be answered. Of course approximation has its limitations. For instance, using geometric areas within a building that is only modeled symbolically first seems to makes no sense. But it remains an interesting alternative that can be used to reduce modeling effort.

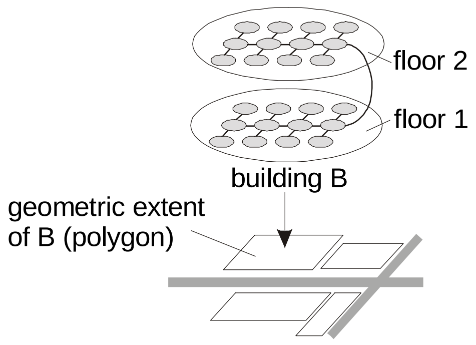


Figure 3.9: Hybrid location model using partial subspaces

3.6.3. Discussion

A summary of the properties of the presented location models is shown in Table 3.2. In contrast to the purely symbolic models presented in the previous section, all hybrid models support geometric coordinates as well as symbolic coordinates. By using geometric information, distances can be modeled more accurately and precisely.

The spatial containment relationship does not need to be modeled manually if the geometry of locations is known. This information can be derived by using geometric operations. Still it makes sense to have a model that stores the containment relation explicitly to allow for efficient queries.

Geometric information can also be used to find out whether two locations lie next to each other, but connections like doors or junctions can no be derived from geometric information and therefore have to be modeled explicitly as for the symbolic approaches.

Compared to the subspaces approach the modeling effort can be reduced by using a partial subspace model where not every location is modeled geometrically. Still a geometry can be associated with location by using approximation.

model type	**supported coordinate types**	**modeling effort**	**distance support**	**"connected to" relation support**	**containment relation support**
subspaces	symbolic, geometric	high to very high	very good	yes (if modeled explicitly)	yes
partial subspaces	symbolic, geometric	high	good to very good	yes (if modeled explicitly)	yes

Table 3.2: Properties of hybrid location models

3.7. Summary and Classification of Existing Approaches

This section briefly summarizes the properties of the different location models presented so far. Existing work is classified along the classes of location models.

Table 3.3 summarizes the classes of location models, their properties and the existing work.

Since the discussion so far has shown that there is no location model serving all requirements at a time with similar modeling effort, designers of location management systems have to choose an appropriate structure of the underlying location model. Especially, the trade-off between supported queries and the involved complexity of the location models has to be taken into consideration.

Table 3.3 classifies the location models with respect to the supported coordinate types (sym=symbolic, geom=geometric), the supported queries (P=position, R=range, N=nearest neighbor), and the modeling effort. Examples for projects using the location model class are listed as well and are discussed in the following sub-sections.

	supported coordinates		supported queries			modeling effort	projects
	sym	geom	P	R	N		
set-based	☑	☐	●	●	⦿	➚	• Guide [CDM+00b] • comMotion [MS01] • QoSDREAM [NC01] • ActiveBadge [WHG92] • Open Distributed Office [RLU94]
graph-based	☑	☐	●	⦿[3]	●	➔	• Aware Home [ODA01] • MavHome [RBB+03]
hierarchical	☑	☐	●	●	○	➔	• MOOsburg location model [GSF+01] • Semantic Spaces [BS01]
combined symbolic	☑	☐	●	●	●	➚	• Active Map [Sch95]
subspaces (hybrid model)	☑	☑	●	●	●[4]	🡅	• Jiang [JS02] • Leonhardt [Leo98]
partial subspaces (hybrid model)	☑	☑	●	●	●	➚	• Nexus [BBR01, DR03] • Semantic Location Model [HL04]

Table 3.3: Properties of location models and overview of existing implementations

[3] "Range" defined by distance to reference location.

[4] If the "connected to" relation is modeled.

3.7.1. Set-based Location Models

Modeling symbolic locations as identifiers and mapping object ids to location ids in location services has been widely adopted. The Guide project identifies the locations of interest to tourists by the WaveLAN access point id [CDM+00b]. The Active Badge system stores the identifier of a user's badge with the symbolic location where the badge has been observed. Without defining further locations as ranges only position queries can be processed with minimum modeling effort. However, an extension of such systems allowing for overlapping sets of locations and thus range queries has been used in the Open Distributed Office projects [RLU94]. The modeling effort increases with the number of locations introduced to the system. QoSDREAM [NC01] relies on a mapping of location identifiers and object ids. By applying observers to sets of locations, applications can be notified when a mobile object has been observed in a set of locations. This provides means for range queries but causes considerable effort, since the overlay of observers modeling spatial inclusion has to be set up based on the basic sets.

3.7.2. Graph-based Location Models

This class of location models naturally provides means to model distance making them suitable for all navigation oriented tasks. Applications can be found in the domain of smart environments [ODA01, RBB+03]. Spatially scoped areas are modeled by the location users populate, e.g., floors and rooms, and a connection model defines connectivity and distance. Navigation services incorporating the positions of individual objects can be implemented that way. There is no direct notion of ranges. Either a combined approach is taken modeling ranges as an overlay structure – in the simplest case ranges are specified as sets of locations themselves – or ranges can be defined based on their extension, i.e., by a reference location and the distance to this location.

3.7.3. Hierarchical Location Models

In contrast to graph-based models, which reflect distance well but require additional overhead to express ranges, hierarchical models are designed to reflect the inclusion of locations. This allows to structure locations into a hierarchy. It is noteworthy that although approaches such as EasyLiving [BS01] or MOOsburg [GSF+01] only model the spatial inclusion between locations other kinds of hierarchical relations can be modeled such as an organizational structure. A company may structure its location into development, marketing, research, and production. A distributed systems development team – and its offices – may be organized to be nearer to the distributed systems research team in a hierarchy than the theoretical computer science research group in the offices nearby.

Ranges and their relations – spatially or with respect to other criteria such as organizational relations – are well reflected in a hierarchy. Distances do not come with a direct concept in such location models. One way to use a hierarchy to compare distances between positions is to consider the smallest locations in the hierarchy that contain these positions. That means, positions grouped by smaller locations are considered to be closer to each other than positions grouped by larger locations, e.g., two rooms on the same floor can be said to be closer than two rooms where the smallest common range is the building.

3.7.4. Combined Symbolic Location Models

An obvious approach combining the benefits of graph-based and hierarchical location models are combined symbolic location models, such as those used in the Active Map [Sch95]. Either a common data structure is applied that allows to reflect the inclusion relation as well as the connected-to relation between locations such as in [BBR01], or two different location models are maintained where one reflects the distances and the other the ranges. Clearly, the expressiveness of such models combines the benefits of both models but with a trade-off with respect to the modeling effort, which basically consists of the effort of creating two location models. This effort is only justified when applications require range *and* nearest neighbor queries. This will likely be the case when a location model is set up to serve a number of applications, e.g., by providing an application spanning context model.

3.7.5. Hybrid Location Models

Hybrid location models provide information about locations based on symbolic *and* geometric coordinates, which are used to define the spatial extent of locations.

Basically, all of the symbolic models above can be extended to a hybrid model by annotating location with their spatial extent. A graph-based model may use this information to calculate the weight of connections or rooms in order to provide more accurate distances. Since the effort of obtaining spatial extensions of locations is rather high, some projects consider a combined model as basis, e.g., [BBR01] and [HL04]. The effort of annotating all locations in a location model with geometric information can be used to map the symbolic coordinates into a global, geometric reference systems realizing a subspaces approach [JS02]. If this is not necessary, a partial subspace approach can be taken. Such approaches can be realized either top-down or bottom-up. In [DR03] a top-down approach is taken that allows approximating the spatial extents of children in a location hierarchy by the extents of their father nodes. A bottom-up approach would annotate the leafs in a location model and approximate the spatial extents of a father node by the extents of its child nodes. The top-down approach allows the integration of an area that is modeled by a hierarchy of symbolic locations into a geometric model. The root of the integrated hierarchy is exact with respect to the annotated spatial extent whereas the approximation leads to some errors in the spatial extents along the hierarchy. In contrast to that, the bottom-up approach provides the highest accuracy at the leafs. The modeling effort is great for this approach if the hierarchy has many leaves. Clearly, it is application dependent which approach should be taken under given requirements.

3.8. Conclusion

Modeling locations is crucial for most location-based or context-aware applications. Location models provide means for spatial reasoning based on coordinates, e.g., the determination whether a coordinate is within a given range or which coordinates are nearby. Although geometric coordinates already provide an implicit notion of distance and ranges, location models allow to model the constraints of the physical world, e.g., road networks or floor plans. For symbolic coordinates like room or floor numbers, a location model with explicitly modeled relations between locations is essential to support queries beyond simple position queries.

The requirements of applications can be manifold. Since the structure of a location model determines which kinds of spatial reasoning can be processed, a number of location models may be appropriate. Beside the relevant queries a location model has to support, especially the modeling effort has to be taken into consideration when choosing a location model for an application or a platform serving a number of applications. A hybrid model managing geometric and symbolic coordinates supports all kinds of location-based queries very well but is at the same time the most complex type of location model. Location models managing only symbolic locations can be set up more easily. If, beside object positions, distance is the only relevant information, a graph-based symbolic model can be used, whereas range queries are supported very well by hierarchical symbolic models. If higher accuracy is required only partially within limited areas, a partial subspaces model, which augments a symbolic model partially with geometric information, might be the right choice.

The discussion of location models in this chapter shows that there is no location model which satisfies all identified requirements at a time with a low modeling effort. Designers of context-aware applications and systems thus have to choose location models carefully with respect to the required spatial reasoning and the involved modeling effort.

4. Usenet-on-the-fly - supporting locality of information in spontaneous networking environments

People on the move are typically interested in information with respect to their proximity. Location-based services in general supply users with information about their proximity typically relying on an infrastructure storing the information and tracking the mobile objects, i.e., users. In this chapter we present an approach for spontaneous, i.e., ad hoc, networks inspired by the Usenet. Information is exchanged using a peer-to-peer synchronization mechanism. The information is made available through channels grouping related information. The information propagation is solely based on spontaneously connected devices not requiring any infrastructure. Our prototype implementation shows the technical feasibility of our approach, whereas simulation results show the applicability of information diffusion in outdoor scenarios with a realistic number of nodes, covering a city center.

4.1. Introduction

Location-based services (LBS) gain popularity. While many commercial approaches are tied to the cellular phone infrastructure, e.g., [GSM][LBS][Swiss], researchers address solutions for the indoor domain [HHS+99][KOA+99][WJH97] and outdoor domain [CDM+00a][Pas97] or both [HKL+99] based on their own infrastructure. Common to these approaches is the necessity of an infrastructure storing location-dependent data and management of user positions.

The availability of small computing devices, e.g., Personal Digital Assistants (PDAs) or cellular phones, equipped with short range radio transmission technologies such as Bluetooth or IEEE 802.11 allows information exchange on a peer-to-peer basis whenever two devices are within each other's radio range. Additionally, information of the environment can be captured from sensors equipped with similar radio technology.

The overall focus of our research is to investigate what mechanisms are needed to support applications for mobile users in a ubiquitous computing environment using ad hoc communication. The goal is to provide a foundation that allows users to successfully interact with other users and their environment. This includes collecting and providing information about the spatial context of the user.

A major problem in mobile ad hoc networks is the management and dissemination of information. Since the mobile devices are restricted in their resources, a complete replication of information will not be possible. Information exchange should be restricted with respect to the spatial scope of the information and the interests of the user. Another issue is the multitude of available information: how can a user determine or specify which information is interesting to him or her?

In this chapter, we present an approach for information dissemination based on epidemic algorithms, i.e., diffusion. As one possible scenario, imagine that it is Saturday night and a large number of young people are walking around in the city center looking for some fun. What they really want to know is what is currently going on: where are the cool parties, the hip discos or the most popular bars. So the information needed depends very much on the current context of the user, especially the location. The distribution of the information can be asynchronous and possibly anonymous, which fits well with our proposed diffusion-based approach.

If cellular phones are equipped with short range radio technologies, such a Bluetooth, they are the ideal devices for our scenario. Almost all young people have cellular phones and use them frequently for writing SMS messages, especially in Europe. (The SMS Service allows the transmission of short text messages between cellular phones using the cellular phone infrastructure). So the general technology is well-introduced, the only difference being that the messages are exchanged using ad hoc connections between devices in the proximity.

In order to structure the information exchanged between devices, the information is grouped into channels according to subjects, similar to newsgroups in the Usenet. Users can subscribe and unsubscribe to channels. Information is only propagated in a distinct area with respect to its locality. As a result, an easy-to-use application can provide users with information about their proximity.

To show the technical feasibility of our concept, we have built a prototype application. However, since the usefulness of our application can only be determined based on a large user population and since the technology is not yet widespread enough for a large-scale usability study, we have conducted a number of simulations to provide some evidence that the diffusion-based approach makes sense in the given context. An important aspect of the simulation is the mobility of the users. Therefore, we need a mobility model reflecting the characteristics of user mobility that may have an influence on the diffusion. In this chapter we take two mobility models and compare the results of the respective simulations: the random waypoint model that is widely used for the evaluation of algorithms in ad hoc networks and a graph-based mobility model [THB+02] that takes the possible user paths, i.e., streets, into account and is therefore more realistic.

The structure of this chapter is as follows: In the next section we present our general system model. Then we describe our application scenario, focussing on the "Usenet-on-the-fly" prototype, followed by a detailed description of the underlying information dissemination protocol. After that we present simulations of the information dissemination protocol and discuss their results. Following a discussion of related work, the chapter concludes with an outlook on future work and a summary.

4.2. System Model

The system consists of mobile nodes users carry. Examples of such nodes are devices like cellular phones or PDAs capable of short range radio transmission. The communication between nodes occurs spontaneously, i.e., whenever two devices are within radio range of each other, they discover each other and can exchange information. Additionally, sensors or info stations may provide local information of the environment to the thereby formed mobile ad hoc network (MANET).

The information exchanged in such a MANET can differ widely and is obviously application-dependent. For the remainder of this paper we consider information to be of local interest. Dishes of the day, temperature of rooms, bus schedules are mostly relevant in the proximity of their real-life source. Since we do not assume any access to an infrastructure we want to investigate, how peer-to-peer computing in such spontaneously formed networks can be used for information dissemination.

We assume the nature of information to be "nice to have". If some information was critical to a user, the user would pay for an uplink to an infrastructure, e.g., via wireless cell-based communication.

4.3. Application Scenario

The application scenario we want to look at is concerned with the propagation of information with a local scope in a MANET. We do not consider multi-hop messages, e.g., routing, here, but only dissemination of information with multiple, previously unknown receivers. The information, as mentioned before, is assumed to be locally relevant. Hence, an information dissemination protocol has to discard the information when the scope of the information is left. Recipients of information must be provided with a classification of the information in order to decide, if they want to accept it and store it locally.

Due to the multitude of information and corresponding information possible in such scenarios, we reduce the complexity by focussing on a simple scenario. The aim is to provide an evaluation of diffusion-based information dissemination and demonstrate how information with local relevance can be handled in such ad hoc scenarios.

The information in the context of this paper is represented as a message. A message contains a source, which created the message, a topic, which classifies the content of a message, and a body carrying the information of the message.

Messages could represent sensor data, with the sensor ID as source, the kind of sensor information, e.g., temperature or humidity as topic, and the currently sensed value as content of the message. Another example could be the provision of bus schedules, where the distinct bus station is the information source and the topic would determine a transport schedule with the message body containing the next bus departure. Moreover, users could also provide information, e.g., rankings of restaurants or shop offers and feed them into the system by creating messages. Actually, this inspired our prototype application - Usenet-on-the-fly - which is presented in the next section. Following that, we explain the information dissemination protocol in more detail and present some simulation results.

4.4. The Usenet-on-the-fly Prototype

The Usenet provides users with the ability to subscribe to so-called newsgroups where they can read, post and reply to articles. The newsgroups group articles with a distinct topic. It is considered rude in the Usenet community to place articles in inappropriate groups - being "off-topic". The Usenet does not rely on a centralized infrastructure. Instead, servers providing "news" to users allow them to read, post and reply to articles. This local news is propagated over news feeds to other news servers which present these articles to their users, receive the replies and postings and offer these as news feeds to other news servers.

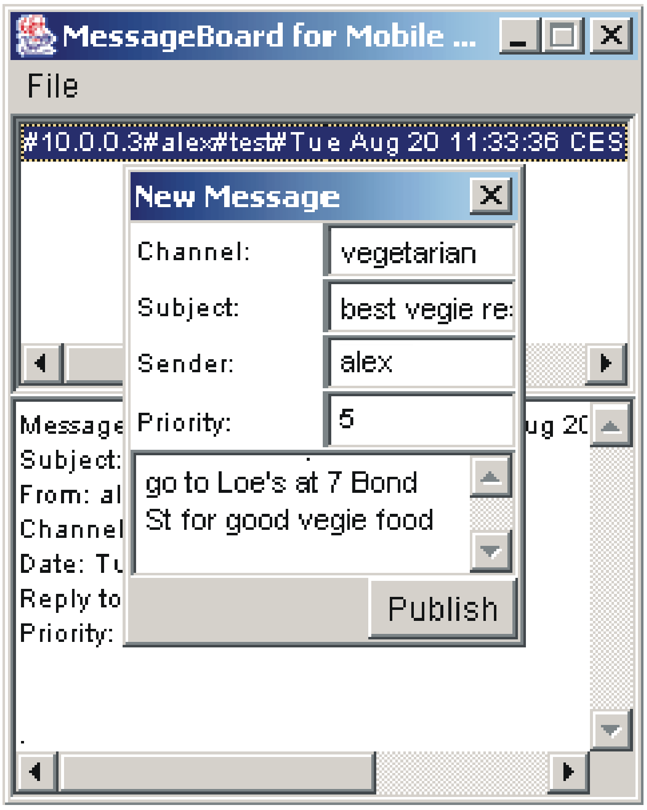
The architecture of the Usenet originates from former times when many computers were not permanently linked to each other as nowadays via the Internet. However, this situation reflects the characteristics of an ad hoc network where nodes are not permanently available but only when they are in the vicinity of other nodes. The concept of categorizing information by grouping them into newsgroups according to topics - or in our terminology: channels - and peer-to-peer reconciliation of content matches the needs of information propagation according to our requirements.

The data model of our Usenet-on-the-fly is directly corresponding to the messages as they were informally defined in the previous section:

- Message headers, i.e., the channel name, the subject and the sender
- Message content, i.e., the actual information

Scoping of the information is simply done by adding a hop count. Thereby, the scope within which a message is presented to other nodes is restricted, which, in most cases, automatically leads to a geographical scoping.

4.4.1. Functionality



#10.0.0.3#alex#test#Tue Aug 20 11:33:36 CES

Channel: vegetarian
Subject: best vegie re:
Sender: alex
Priority: 5

go to Loe's at 7 Bond St for good vegie food

Message
Subject:
From: al
Channel
Date: Tu
Reply to
Priority:

Figure 4.1: Usenet on the fly user interface

Figure 4.1 shows the user interface of the Usenet-on-the-fly prototype. The prototype is realized as a Java application. We used notebooks and Compaq iPaqs equipped with WaveLan cards as an evaluation platform.

A user can create channels and messages and open an existing channel to retrieve messages. Figure 4.1 shows the dialog for creating a message. This dialog combines the creation of a new message with the possible creation of a new channel. Additional attributes, e.g., the priority, allow filtering of messages in order to save bandwidth or space on the devices.

Users can subscribe to a topic, i.e., a channel, and receive all messages on that channel. The local database containing the messages is updated whenever another node is met. Both nodes negotiate about their channels and contents and exchange the difference. New channels are presented to the user who can subscribe to them or simply ignore them.

When users on the move are visiting different places, only the information concerning these places is offered in the channel. Information is scoped in its lifetime by a time-to-live (TTL) as well as in its propagation scope by a hop count. The message exchange is based on a single hop communication, i.e., devices only communicate with other devices in their transmission range. Hence, restricting the number of times a message can be passed on between nodes leads to a geographical scoping.

As an example consider a user subscribing to "restaurant menu", "bus schedule", and "restaurant recommendations". The channel "restaurant menu" will contain the dish of the day of the restaurants within a distinct vicinity, depending on the hop count. Also, only the bus schedules of nearby bus stops are presented in the "bus schedule" due to the scoping. Not only stationary entities like restaurants or bus stops can create messages. Other users can use the "restaurant recommendation" channel to express their satisfaction about a particular restaurant. This information is scoped with respect to its local lifetime and geographical scope as well.

4.4.2. Architecture

The Usenet-on-the-fly prototype was built in a straightforward way. Nodes maintain a small database where the channels and all messages are stored. The database is regularly scanned and messages whose TTL has expired are deleted. The user interface operates on the database and allows the display of channels and their messages as well as the creation of new messages and replies.

The content of local databases are synchronized with other nodes whenever they are within their radio transmission range. First, the channels are compared and new channels and messages are announced to other nodes. Before offering a message, the hop count is considered. If the scope of a message has been reached, it is no longer propagated to other nodes.

The resulting architecture is depicted in Figure 4.2. Central to the system is the database where messages are stored in the corresponding channels. The user interface accesses the database in order to display available channels and on selection of channels the messages of the channel. The user can create new messages and channels leading to new data in the database.

The content of the database is propagated by a simple diffusion protocol. We will describe the protocol in detail in the next section. For short: the protocol announces locally available data to other nodes. These nodes can request the information and store it in their databases. After a node has propagated its database content, it switches the role and updates its database by the advertised channels and messages of the other node. This data reconciliation occurs whenever two nodes "meet". To allow nodes to continue exchanging data when they stay in communication range, they can end their communication and then "rediscover" each other.

The communication subsystem is built on top of a minimized servlet container which offers the Simple Object Access Protocol (SOAP) for message exchange. The SOAP standard is well-suited for interoperability between different platforms. However, it currently restricts us to unicast communication, so we cannot take advantage of broadcast protocols that are suitable for propagating data to a larger set of recipients.

The prototype has been built in Java. The platforms for evaluation were notebooks and Compaq iPaqs, both equipped with WaveLan. It is available for download from http://www.informatik.uni-stuttgart.de/ipvr/vs/de/people/haehnejg/#misc

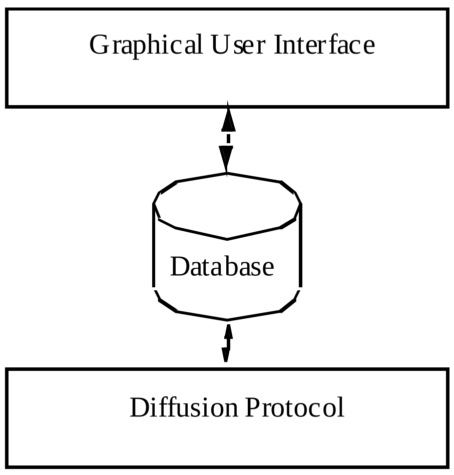


Figure 4.2: Usenet-On-The-Fly Architecture

4.5. Information Dissemination Protocol

The messages in the system are disseminated using a diffusion-based protocol that we call Channel and Message Diffusion Protocol with Negotiation (CMDPN). The pseudo-code of an algorithm implementing that protocol on a given node is shown in Figure 4.3.

When a node A discovers another node B in its transmission range, it sends an advertisement message listing all the channels (consisting of a unique channel ID and a description of the channel topic) and the IDs of the messages it currently has in its database. Node B then goes through the advertised channels and checks, if it has seen them before. If not, the user is given the channel description and is asked, if he or she wants to subscribe to the new channel. Having updated the subscription information, Node B goes through the advertised message IDs pertaining to those channels it has subscribed to. It creates a request message containing the message IDs of the messages it does not have in its database yet. On receiving the request from Node B, Node A collects the requested messages and sends them to Node B, which updates its database accordingly.

```
TYPES
        message_id: unique id
        channel_id: unique id
        topic: string
        message_body: string

        message: struct
                channel_id
                message_id
                message_body

        VARIABLES
        channel_topic = array[channel_id] of topic
        message_ids = array[channel_id] of list of message_id
        messages = array[message_id] of message
        seen_channels = list of channel_id
        subscribed_channels = list of channel_id
EVENT HANDLERS
        ON_NODE_DISCOVER()
           Channel_ADV ca = empty list
           Message_ADV ma = empty list
           for each channel_id in subscribed_channels do
                append(ca, (channel_id, channel_topic[channel_id]))
                for each message_id in message_ids[channel_id] do
                      append(ma,(channel_id , message_id))
                od
           od
           if not empty(ma) then
                send_message((ca, ma))

        ON_RECEIVE_ADV((ca: Channel_ADV, ma: Message_ADV)):
           for each (channel_id, channel_topic) in ca do
                if channel_id not in seen_channels then
                      append(seen_channels, channel.channel_id)
                      if ask_user(channel_topic) then
                            append(subscribed_channels, channel)
                            channel_topic[channel_id]:=
                            channel_topic
                      fi
                fi
           od
           Message_REQ mr = empty
           for each (channel_id, message_id) in ma do
                if channel_id in subscribed_channels then
                      if needed(message_id) then
                            append(mr, message_id)
           od
           if not empty(mr) then
                send_message(mr)

        ON_RECEIVE_REQ(mr: Message_REQ):
           Message_DATA md = empty
           for each message_id in mr do
                append(md, messages[message_id]))
           od
           if not empty(md) then send_message(md)

        ON_RECEIVE_DATA(md: Message_DATA):
           for each message in md do
                if needed (message.message_id) then
                      append(message_ids[message.channel_id],
                            message_id)
                      messages[message_id]:= message
                fi
           od
```

Figure 4.3: Channel and Message Diffusion Algorithm with Negotiation (CMDPN)

Of course, the same protocol is applied in the other direction between node B and node A

The CMDPN is a simple protocol for replicating Usenet-style messages. Its purpose is to minimize the exchange of unwanted messages, saving bandwidth and energy, which are scarce resources for mobile devices using wireless connections.

Further improvements could be:

- To further reduce the data that needs to be exchanged, the protocol could be split up into two phases. In the first phase only the channel information is exchanged. Then, in the second phase, only the information about the messages pertaining to those channels a node is subscribed to need to be exchanged.
- The actual messages could be exchanged according to user-defined priorities, which is especially helpful, if the devices are not within communication range long enough to exchange all messages of interest.
- A history of mobile nodes and the data which has recently been exchanged with them could be kept to keep the message advertisement messages small.

In the following section, we will present some simulations to evaluate the effectiveness of the dissemination of messages to a population of nodes, i.e., how many nodes have received a certain message in what period of time.

4.6. Simulations

This section will describe the simulation context used for the evaluation of the CMDPN protocol, as well as the results, including a discussion of the results.

Since, as a first step, we were mainly interested in the maximum effectiveness of message dissemination, given a certain population of nodes, we used simplified assumptions:

We assume that all nodes are interested in all the messages and that these messages pertain to a single channel. We did neither restrict the hop count, nor set a restrictive TTL, so the messages are distributed within the whole area over the time of the simulation. To control the introduction of messages into our system, so that we could more easily calculate the spreading of information, we assume that the messages are introduced into the system by special stationary „sensor nodes", e.g., providing the local temperature or introducing the meal of the day. Each of the sensor nodes continuously provides the same single message to the mobile nodes in its proximity.

The CMDPN protocol was simulated using our Java-based CanuSim simulator, which implements a simple MAC layer that prevents multiple nodes from accessing the same wireless channel simultaneously. The main advantage of our simulator is that the two mobility models we present in the following can easily be integrated.

4.7. Simulation Model

The simulations were performed for an outdoor context using two different mobility models. The first model is the so-called random waypoint model (RWP). This model is often used for the evaluation of algorithms in the area of MANETs [BMJ+98] and originated in the application area of rescue and disaster operations. In the RWP model a mobile node chooses a random destination and a speed and then moves directly to the destination using the given speed. The size of the area covered was 2462 m x 1733 m, equivalent to the area of the city center used later on.

The second mobility model - the graph-based mobility model GBM [THB+02][Ste02] - assumes that mobile nodes do not move randomly, but according to an infrastructure, e.g., road map or building layout. It models the spatial environment as a graph. The example graph for our simulations models a typical city center, as it can be found in Central Europe. The model contains 115 locations on an area of 2462 m x 1733 m interconnected with 150 edges. Figure 4 shows a sketch of the city graph used.

In several scenarios different numbers of mobile nodes as well as sensor nodes, each providing one piece of information, were placed randomly on the graph. Destinations were chosen randomly out of the 115 locations in the graph scenario or randomly in the RWP scenario. The mobile nodes moved around at normal pedestrian speed, i.e., between 3 and 5 km/h.

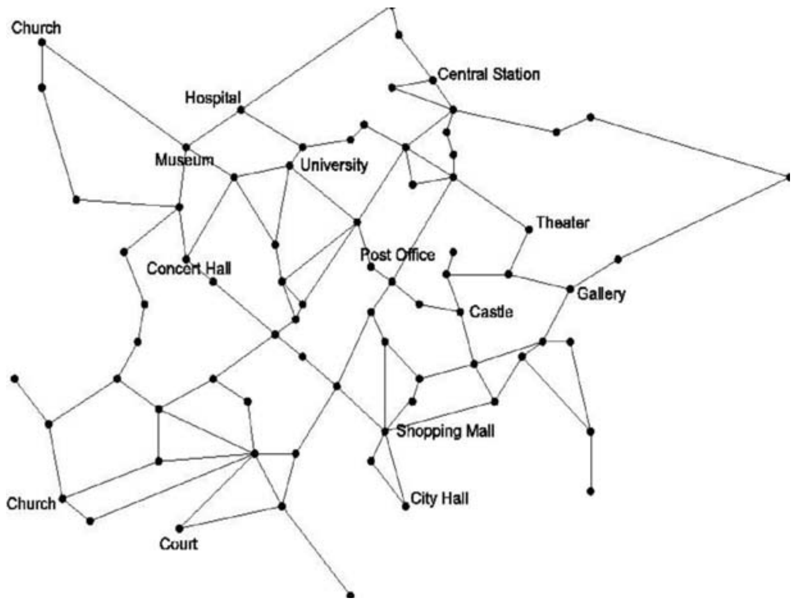


Figure 4.4: City Center Graph

On reaching a destination, mobile nodes stayed there for 12 to 20 minutes, representing pedestrians stopping at a shop or station, before choosing a new random destination. The sensor nodes remained stationary, broadcasting their sensor information to mobile nodes within transmission range. The mobile nodes all used CMDPN as the protocol for message exchange. The time needed to discover a node in transmission range was assumed to be between 2 and 3 seconds, which corresponds, for example, to average Bluetooth discovery times [KL01]. All scenarios used a transmission range of 75m.

The simulation runs were terminated when a certain level of *information spreading* was reached. The information spreading is calculated by

$$is = \frac{\sum_{m \in M} \text{databasesize}(m)}{|M| \times |S|}$$

where M denotes the set of mobile nodes and S denotes the set of sensor nodes that each introduced a single message (500 bytes) into the system. The function *databasesize(m)* sums up all the messages stored on the mobile node *m*.

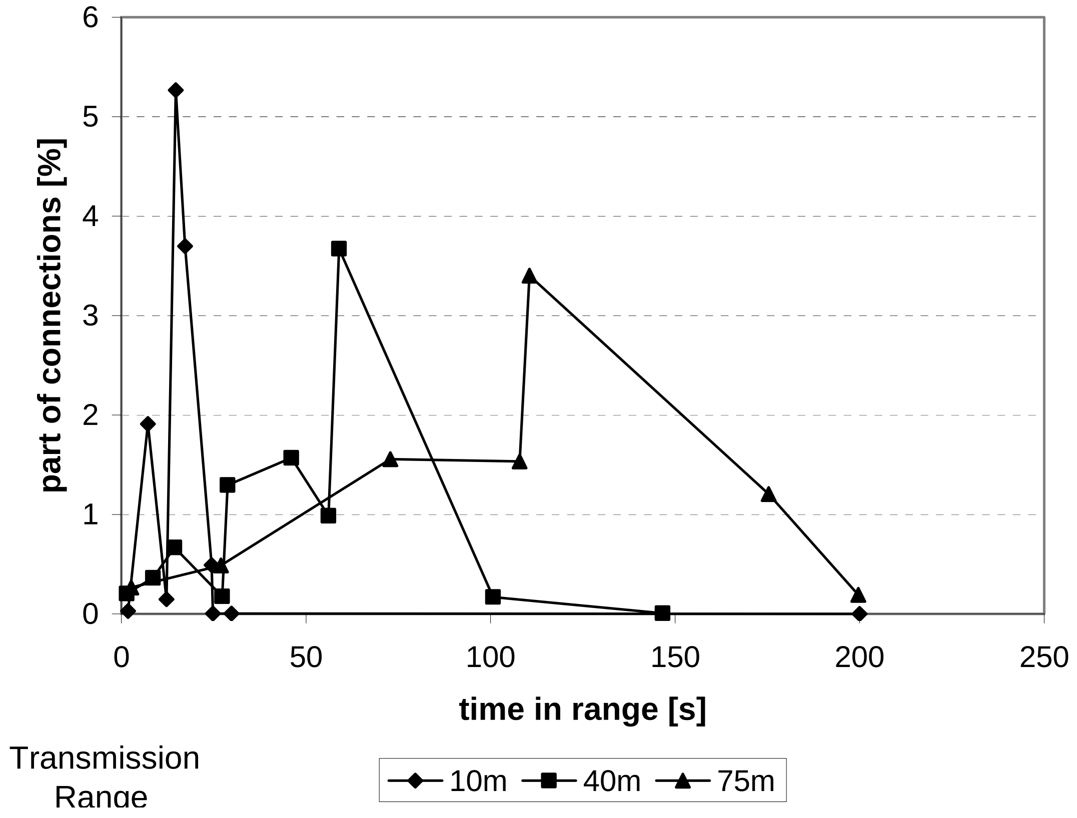


Figure 4.5: Time in Communication Range Depending on the Transmission Range between Mobile Nodes and Stationary Sensor Node

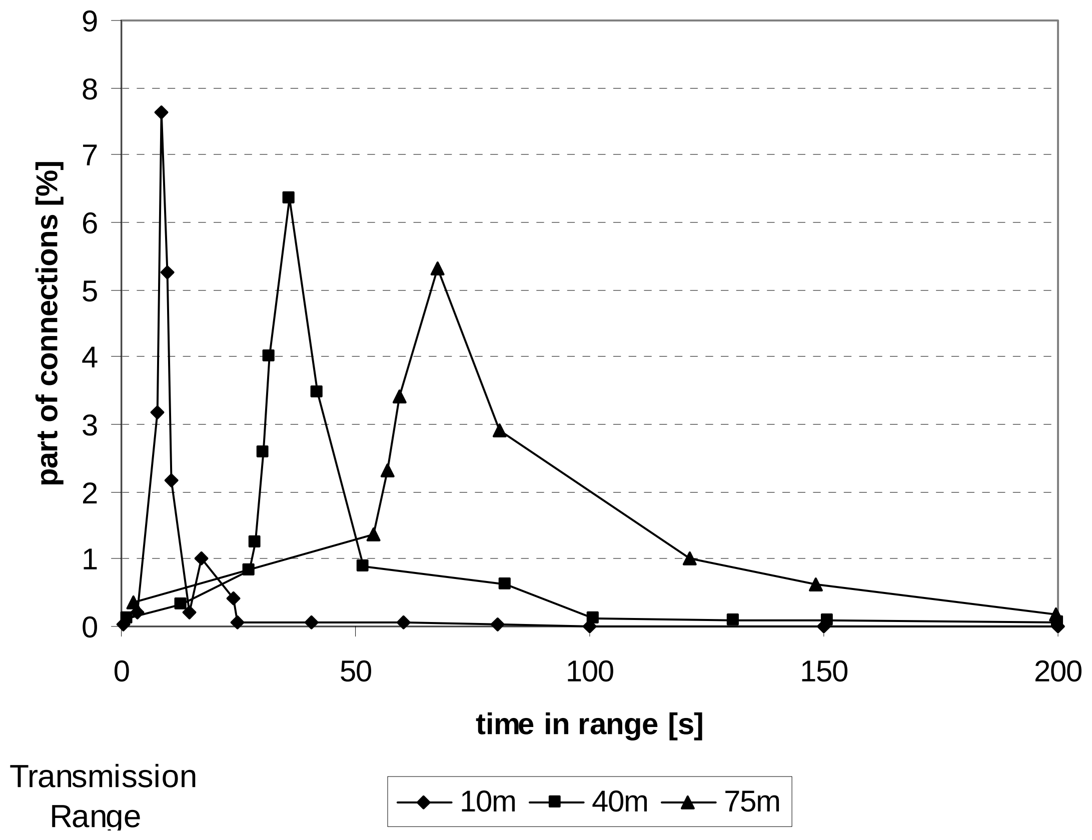


Figure 4.6: Time in Communication Range Depending on the Transmission Range, between Mobile Nodes

We have integrated the graph shown in Figure 4.4 into our simulator for MANETs in order to simulate realistic mobility patterns of users. A first interesting result concerning the relation between communication time and transmission range is shown in Figure 4.5 and Figure 4.6. We placed 100 sensor nodes in the city center scenario and measured the average time of communication between 1000 mobile nodes and the 100 sensor nodes. The figures are based on a one hour simulation of the scenario.

Figure 4.5 shows the distribution of time mobile nodes are in communication range with any sensor node. Assuming a communication range of 75 meters most nodes have more than 50s per connection to communicate, whereas 10m transmission range allows only less than 15s for most connections.

Figure 4.6 shows the distribution of communication time between the mobile nodes. Here the results are slightly worse for the 10m transmission range, since the mobility of nodes shortens the transmission time to 7 seconds, whereas for 75m most nodes still have more than 50s per meeting. Information dissemination in such ad hoc networks has to be aware of these small slots for communication, i.e., not relying on stable routes and long-term communication relations

4.8. Simulation Results

This subsection presents the simulation results of the aforementioned diffusion algorithm based on random waypoint versus the graph-based mobility model.

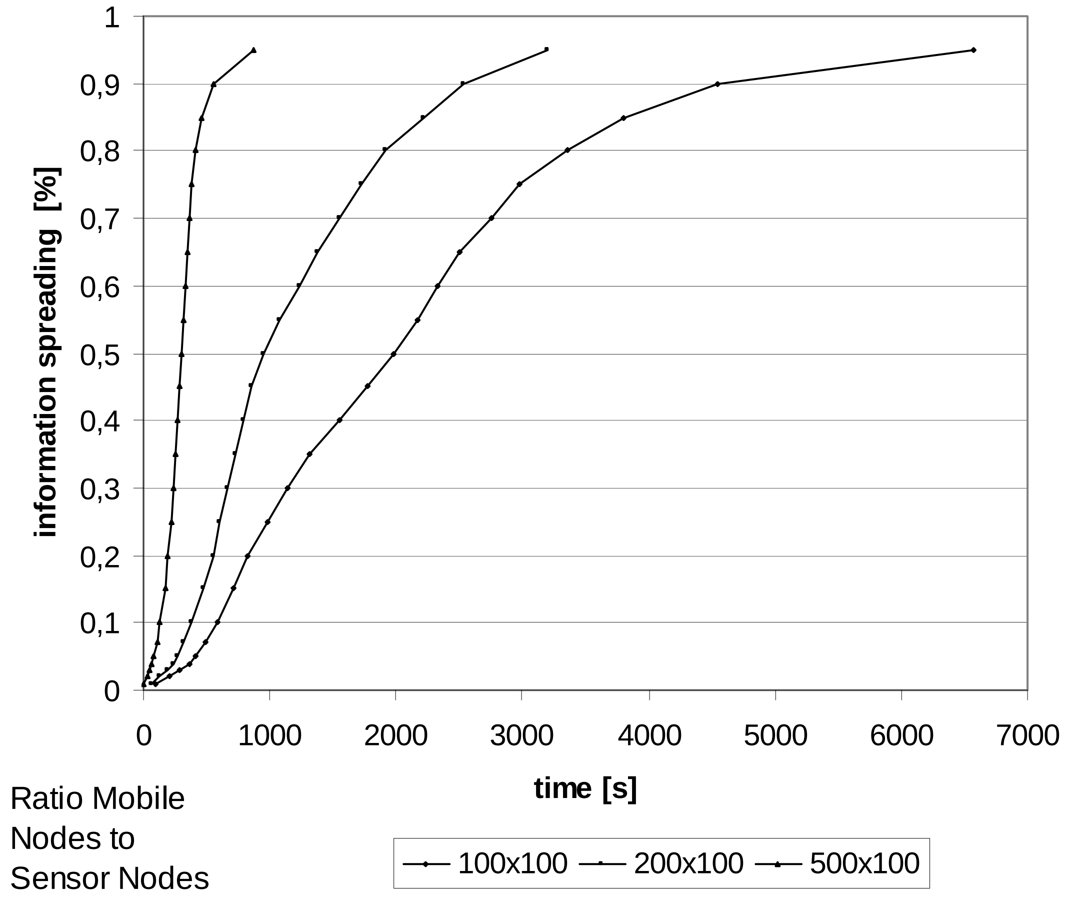


Figure 4.7: Information Spreading over Time for Random Waypoint Movement with 100 Sensor Nodes

Figure 4.7 and Figure 4.8 present the simulation results for the RWP scenario based on a transmission range of 75m. The simulations were run until 95% information spreading was reached, meaning that every mobile node carried almost all the information disseminated by the sensor nodes. The results show that a higher number of mobile nodes supports the information spreading. Nevertheless even a small number of nodes leads to a reasonably fast message replication considering the large area: 100 mobile nodes discovered 100 sensor nodes on an area of approximately 4 square kilometers reaching an information spreading of 95% in only little more than two hours moving at pedestrian speed. A larger number of mobile nodes (500) reaches the same amount of information spreading in less than half an hour. Remember that 500 people in a city center is still a fairly small number.

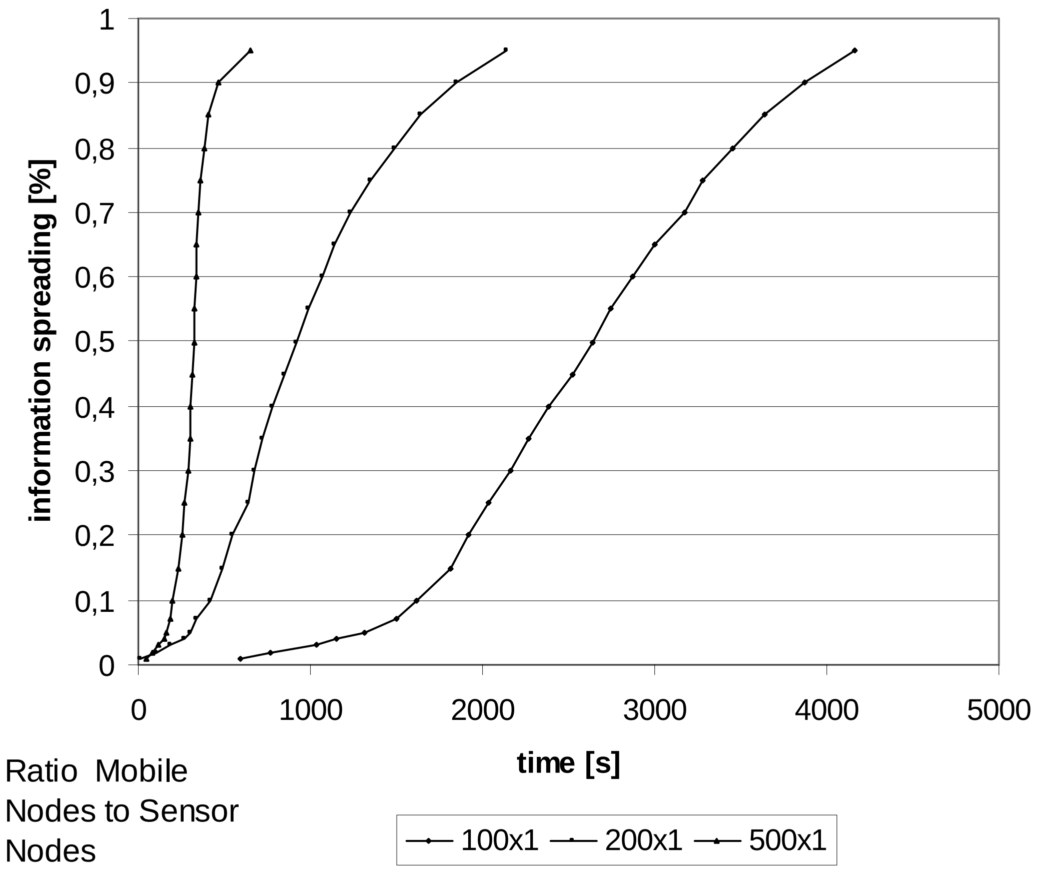


Figure 4.8: Information Spreading over Time for Random Waypoint Movement with One Sensor Node

Figure 4.9 shows the results of the simulations performed with the GBM pattern. The obtained results show a significant improvement over the results of the RWP pattern. The graph-based simulations showed to be approximately twice as fast until the information spreading of 95% was reached. The major reason for this improvement is the fact that the mobile nodes only move along the edges of the graph and do not occupy the whole area as they do in the RWP model. Since the GBM pattern represents our initial outdoor scenario better, we expect CMDPN to behave towards those results in a "real world deployment".

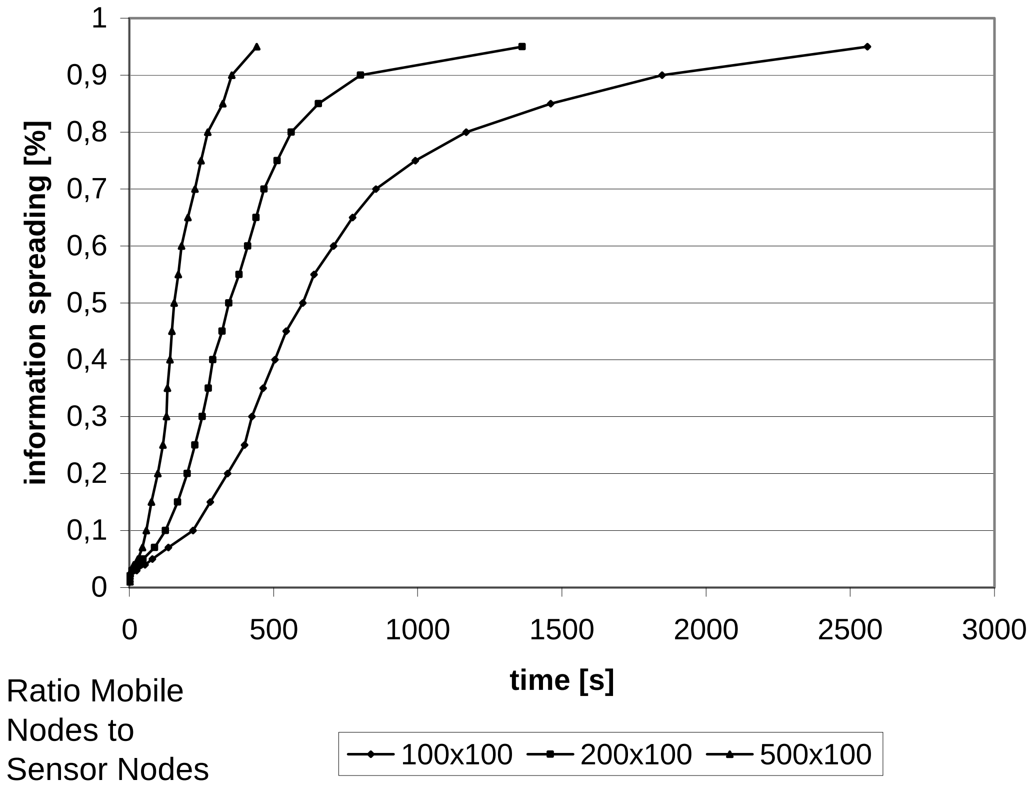


Figure 4.9: Information Spreading over Time for Graph Walk Movement with 100 Sensor Nodes

A second set of simulations with only one sensor node was conducted to investigate the effect of the spreading of a single information item. Figure 4.8 and Figure 4.10 show the results obtained with RWP and GBM pattern respectively. The results show that, once the information has been picked up and passed on a few times, the steepest rise of the curve is reached. This shows that the information is spread very quickly around its source supporting the locality aspect of many information items in ubiquitous computing.

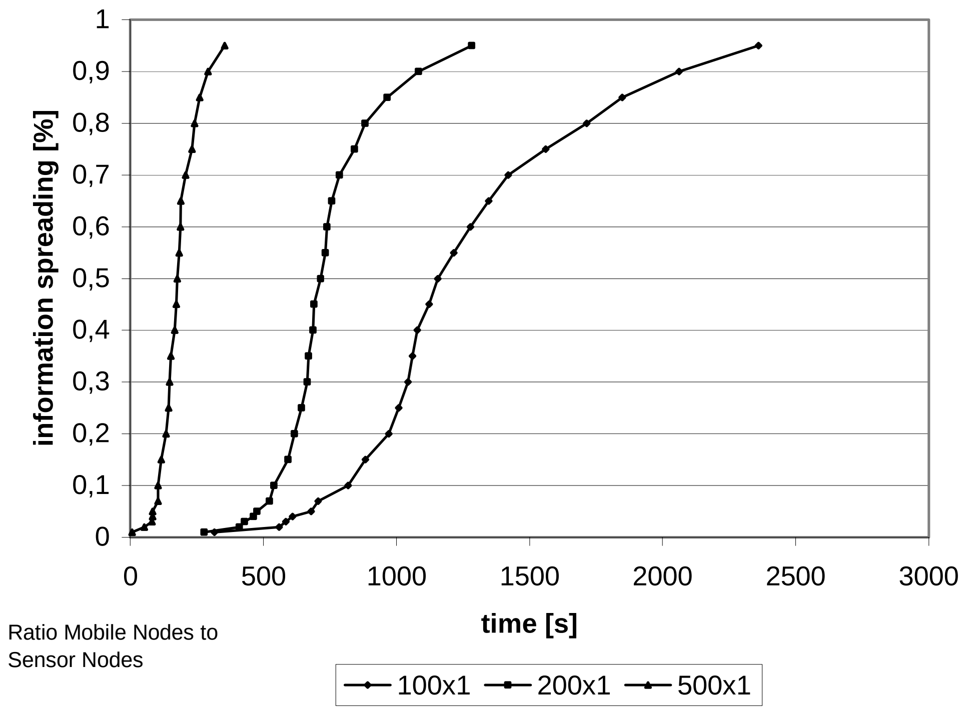


Figure 4.10: Information Spreading over Time for Graph Walk Movement with One Sensor Node

4.9. Discussion

The simulation results show that, assuming realistic values for the transmission range and the density of mobile nodes, the distribution of an update in the vicinity of an information source is a matter of minutes. Reaching an almost complete spreading of information can also be achieved within less than an hour.

The difference between the simulation results based on different mobility models shows the importance of using realistic mobility models in order to get realistic simulation results.

4.10. Related Work

Applications based on message exchange in infrastructure-based systems such as the Usenet have been used for a very long time. More recently, peer-to-peer file sharing application like Gnutella [Kan01] have become popular. However, as we are interested in systems based on ad hoc networks with mobile nodes, we want to mostly restrict our discussion in the following on systems fitting those characteristics.

In the area of collaborative wearable computing, the ad-hoc exchange of information between mobile users during chance encounters has been investigated. Application scenarios including the exchange of tasks between user agents [KSS+99] and the dissemination of trust information [SKJ+00] have been simulated. In those scenarios the information distributed is much more specialized than in ours and exchange of information depends much more on the individual users themselves, even though the underlying mechanisms are very similar.

Much work has been done on routing in MANETs, where messages between sender and receiver are exchanged on [DGH+87] an unstable path built of mobile nodes (see, for example, [PB94] or [Joh94]). Usually, however, a fully connected path from the sender to the receiver is required to be able to forward a message. In [VB00] a general routing protocol for partially connected networks is discussed, which, similar to our approach, uses the moving mobile nodes to relay messages. Their results show, that such an approach is feasible, transmitting 100% of the messages in most cases in reasonable time.

Recently, information diffusion has been discussed in the area of sensor networks. There, information is exchanged between a number of randomly placed non-mobile nodes, which acquire a model of their environment using built-in sensing systems (e.g., for seismic data or images). Algorithms in this area have to be able to cope with the failure of single sensors. Different variants of broadcast algorithms have been discussed for such sensor networks with the goal of reducing bandwidth and energy consumption [XWC02]. In [KHB99] a family of negotiation-based protocols for sensor networks, called SPIN, are discussed. It is shown that they perform better regarding performance and energy consumption than the more simple broadcast protocols.

More closely related to our approach regarding the dissemination of data is the 7DS system [PS01]. Their underlying data model relies on a hierarchy of web-caches and the information can be accessed via a client/server-based approach from an infrastructure if available. If network partitions occur, the mobile nodes rely on their cached data, which can be updated similar to the diffusion algorithm that we have presented here. Cooperation among the mobile nodes allows the access of information in other caches. Queries trigger a diffusion process of data through the mobile nodes which update their cache with the requested information. The mobility model is a strict random waypoint model neglecting spatial constraints. However, their objective relates to the fragmented data storage. Replication of the data on every node and restricting the coverage of the information dissemination, e.g., to an area or a number of nodes, is not an issue there.

4.11. Conclusion and Outlook

In this chapter we have shown that the dissemination of data in large MANETs is feasible. Simulation results show that information can be spread among several hundred users in a city center scenario within approximately 10 to 60 minutes from its initial creation at the sources. This time interval is appropriate for many types of information that may be of interest to a pedestrian walking through a city, such as information about current events or specials on sale. A *1-to-many* message exchange application, like our Usenet-style prototype for PDAs, can be implemented using devices and technologies which will soon be deployed among many (millions of) users.

It can be concluded that information can be made available in MANETs through the dissemination of messages using a diffusion algorithm. This means that in a lot of cases it is not necessary to access an infrastructure in order to obtain information concerning the current proximity.

After showing the general feasibility, there are still many questions that remain unanswered. Our simulations show that the time it takes to disseminate information among users varies depending on how the mobile nodes, i.e., the users, move around. This strengthens the need for realistic user mobility models in order to obtain a reliable performance prediction of new systems prior to deployment.

More simulations are needed to show, if limiting the scope of the information by hop counts successfully approximates the locality of information as we expect. In any case, this solution gives only a very coarse resolution of locality. A more sophisticated solution here is to employ a more detailed world model [BBR01] and location sensors, e.g., GPS, to locate the mobile nodes. Then, the spatial scope of an information could be specified directly and precisely, e.g., information could be interesting for users on the same floor only, but not for those on the floor below, yet the people there might be closer concerning the communication if only distance is taken into account.

In our simulations we have assumed that every user is interested in all channels, i.e., we completely replicated all messages. If we assume a large variety of topics for channels in a real-world system, it is evident that complete replication does not work due to resource restricted devices on the one hand and users that are not willing to carry unwanted information on their devices on the other. A technical compromise is that every user allots some portion of his systems memory and communication time for information that is not of (high) interest to him. The question here is: how much is technically necessary for the system to work and how much are users willing to contribute.

Since our system is based on the assumption that the number of participants is large, the social situations in which a user sees a clear benefit and therefore uses this interaction style has to be investigated. This, we believe, can only be answered by appropriate user studies.

5. A Protocol for Data Dissemination in Frequently Partitioned Mobile Ad Hoc Networks

Distribution of data in mobile ad hoc networks is challenged when the mobility of nodes leads to frequent topology changes. Existing approaches so far address either the network partitioning problem or are capable of handling large amounts of data, but not both at the same time.

In this chapter a novel approach is presented which is based on a negotiation scheme enhanced by an adaptive repetition strategy. Different strategies for the selection of repeated data are presented and evaluated. Simulation results show a reduction of data transfer volume compared to hyper-flooding by 30 to 40% even in the presence of frequent network partitions.

5.1. Introduction

Mobile ad hoc networks (MANETs) are going to be a reality in the near future with more and more mobile devices, e.g., PDAs or cell-phones, being equipped with short range radio technology, e.g., as Bluetooth or 802.11. In our daily environments such MANETs will not only contain the mobile nodes which are typically carried by their users but also incorporate devices being fixed in the infrastructure, such as sensors or information provision points, e.g., info-stations. Applications in such environments can make use of the information being available through the sensors and other nodes. Examples are tracking applications in production plants capturing the location of production material and the state of manufacturing machines, communication on a construction site, missions from civil services, e.g., collaborative fire-fighting, but also convenience applications such as smart city/shopping guides.

Typically, information in such networks itself is spatially scoped, i.e., only from interest within a distinct area nearby the information source. Sensor networks, i.e., ad hoc networks with typically stationary nodes, can setup links between information sources and sinks. Mobility challenges the information dissemination in such networks, since network partitions cannot be treated as errors because they happen regularly. In order to supply applications on nodes with information of their environment a robust mechanism to deliver data is needed. In order to increase availability of data, replication is a candidate to achieve this goal with a trade-off to consistency.

In this chapter we present an algorithm for updating replicated data on mobile nodes which is gathered by information provided by sensors. We refer to such data as *model-data*, since the sensor information provides a model of real-world's state. The consistency of the replicated data is weak, due to unpredictable network partitioning, aiming at delivering the most current state of an information entity and not providing single-copy consistency. Current information shall replace older one and inconsistencies are tolerated as long as the most current information will finally be propagated. Using a hyper-flooding [OT98] approach as the foundation of a three-way-handshake protocol enables our protocol to overcome network partitions. The negotiation of transferred data leads to a significant reduction of the data transfer volume compared to plain hyper-flooding by 30 to 40%.

Next we will introduce the system model. After a discussion of existing flooding techniques for data propagation in ad hoc networks our algorithm is described. Performance results from simulations are presented based on two scenarios before the discussion of related work and an outlook to future work concludes the chapter.

5.2. System Model

The system consists of two kinds of nodes: observer nodes and mobile nodes. *Observer nodes* are equipped with a synchronized real-time clock (e.g., GPS clock) or an appropriate clock synchronisation algorithm [Roem01], and sensors allowing to make observations in their proximity that describe properties of the real world. Every *observation* represents a state change of an *object* that has a unique object ID (*oid*), and has a time to live (*TTL*) that depends on the type of observation. Each observation is timestamped with t_{obs} by the observer node to indicate when the observation was made. Additional information (*info*) may be added by the observer node to describe precisely what kind of state change was observed, e.g., the position (state change described in *info*) of a person (object) or the temperature inside a room. The tuple (*oid*, *TTL*, t_{obs}, *info*) is called *meta-data* because it describes the „what and when" of an observation. The actual distinction of objects on the sensor level is not part of this paper.

Mobile nodes maintain a local copy of the most recent state of all objects, observed within a distinct area. The copies of state information on mobile nodes form a replicated database. The replicated database maintains weak consistency where mobile nodes may keep and use stale information, but any update made to a local copy will add more recent information to the database. The size of such a database is limited due to the locality of information and the resource restrictions of the mobile devices.

The synchronized clocks of observer nodes are necessary to be able to compare two or more independent observations of the same object accurately. The high accuracy of, for example, GPS clocks of approximately 360ns [GPS] is sufficient to distinguish many observations made in the real world, e.g., people's movements. It would, for example, not be accurate enough to observe the direction of a light beam passing two independent observer nodes equipped with a light sensor. In general the accuracy needed is driven by the type of observations that need to be made.

Mobile nodes use local real-time-clocks (RTC) to determine when the TTL of an observation record expires. Those clocks do not have to be synchronized, since they are only used to measure how long a record has been kept locally. Assuming a typical clock skew of a simple hardware RTC of 5 to 15 seconds per day [DALLAS], it would be sufficient to synchronize a few times a day (e.g., when passing any observer node) in order to correct the clock drift and to measure the time a record has been kept accurately enough.

All nodes are equipped with a symmetrical short range RF communication technology that offers a device discovery mechanism and allows two way communication. The RF technology is used to locally broadcast messages, i.e., every neighbor in the transmission range of the sender may receive the message. Additionally, we assume that the MAC protocol follows a CSMA/CA approach that detects collisions. Mobile nodes and observer nodes thereby form a MANET which is assumed to be partitioned very frequently due to short transmission ranges and the mobility of nodes.

5.3. Forwarding Strategies

For the task of distributing observations to mobile nodes a robust forwarding mechanism is needed that can cope with the frequent topology changes and network partitions in a MANET. The evaluation of flooding in such environments [HOT+99] has shown that it provides a good basis for distributing information in highly dynamic and sparsely populated MANETs. Different possibilities for flooding have been proposed and shall be briefely described here since they will be used for our algorithm presented in Section 5.4.

Plain Flooding: The basic version of flooding is a robust way to broadcast information in a network. Every node forwards an incoming message unless it has done so before or some knowledge of the network diameter is available to add a maximum hop count to the message. Although this is very reliable, plain flooding cannot cope with network partions or very high mobility [HOT+99].

Selective Flooding and Gossiping: Selective flooding has been proposed to reduce the number of messages in comparison to the plain flooding approach. The general idea is that a node forwards a message only to a subset of its neighbors [Tan96]. Gossiping is a variant of selective flooding where the message is sent to a subset of neighbors that is chosen randomly [HKB99]. This reduces the number of messages sent with the trade-off of being less robust, especially in networks with a low node density. Selective flooding is based on plain flooding and thus does not cope with network partitions.

Hyper Flooding is a modification of flooding proposed in [OT98]. It allows nodes to forward a message more than once if the set of neighbors changes within a given validity period of the message. This improves the delivery performance over plain flooding in scenarios with frequent topology changes (e.g., due to high mobility) and network partitions that are rejoined within the validity period of the message.

5.4. Negotiation-based Ad hoc Data Dissemination Protocol: NADD

This section describes an algorithm suitable for exchanging observation data in MANETs with frequent topology changes. First, data structures relevant to the algorithm are explained. Second, the algorithm itself is described. Crucial to the algorithm is when and which data is (re-)sent. A deeper discussion of selection strategies of data to be resent is presented.

5.4.1. Data Structures

Every observer node stores an observation record for each object that is currently within its observation range. An *observation record* contains the following elements:

- Object ID (*oid*) of the observed object
- Time-to-live (*TTL*) of the observation
- Timestamp of the last observation of a state change (t_{obs})
- Information that indicates the replication progress of a record (*d*)
- Additional meta-data (*info*)
- State of the observed object

The *oid* kept in the observation record is a unique identifier for a real-world object such as a room or a person. Additional meta-data may be added to describe in more detail what kind of information is represented in the record, e.g., the temperature in a room or a person's position. In the context of this paper, different *oid*s represent information about distinguishable objects. The type of information represented is of no further concern for this paper. The TTL is initialized by the observer node and is continuously decremented by each node that holds a copy of the record. Its initial value depends on the type of observation made (e.g., part of the meta-data) and is supplied by the observer node. The observation time $\mathrm{t_{obs}}$ is recorded by the observer node that has created the observation record originally, i.e., that has actually made the observation. In case of multiple observers of the same object nodes create different records about the same object. These records can be ordered due to the assumption that all observers have synchronized clocks. The precise description of *d* is given in Section 5.4.2.2.

5.4.2. Protocol

In the proposed protocol, messages are sent from a sender to all direct neighbor nodes (local broadcast). The mechanism used to forward observations is implemented using a three-way-handshake where observations stored locally in a node's database (DB) are *advertised* in ADV messages, *requested* in REQ messages from nodes that do not have the advertised information in their DB, and *sent* with DATA messages by the advertising node as shown in Figure 5.1. Since the state information provided by observer nodes may become large, this approach has the advantage that state data is only exchanged if at least one neighbor node requests it. Additionally, the three-way-handshake allows the optimization of advertising many observation records in a single ADV message.

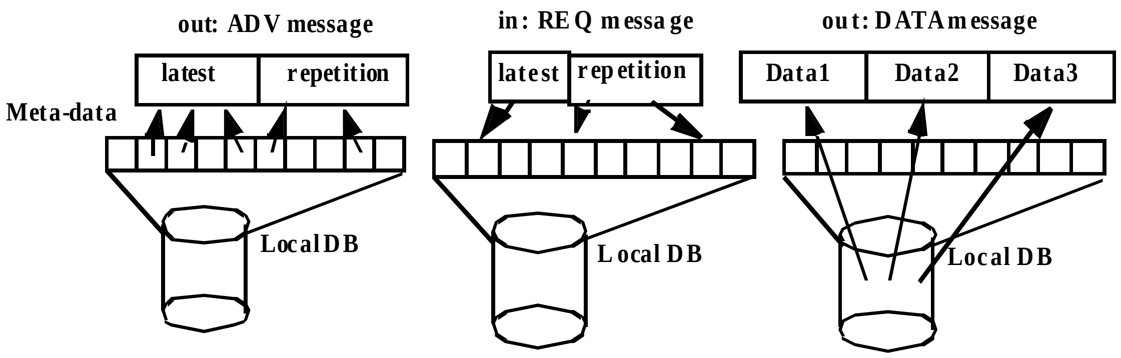


Figure 5.1: Interaction pattern of a node while advertising

An ADV message contains multiple tuples *(oid, t_{obs}, TTL, d)* describing information available in the DB of a node. A REQ message contains multiple tuples *(oid, t_{obs})* of observation records needed by a node in response to an ADV message. A DATA message is a set of observation records that have been requested by any neighbor. Figure 5.3 shows an overview of the protocol in pseudo-code.

5.4.2.1. Interaction Between Nodes

A new information entity, i.e., a new or updated observation record that was either received by a mobile node or observed by an observer node, is offered to all neighbors of such a node by sending an ADV message. Any neighbor node may send a REQ message in return to indicate that it is interested in some of the data. On receiving a REQ message, a node broadcasts the corresponding state information. The protocol as described so far uses plain flooding on top of a three-way handshake. This results in the disadvantages of not overcoming the boundaries of network partitions as mentioned in Section 5.3. To disseminate information across partitions an approach similar to hyper-flooding is added: whenever a node discovers a new neighbor, it is allowed to *re-advertise* observations as long as the TTL has not expired. The TTL is decremented continuously by each node that holds a copy of an observation record. If the TTL equals 0, the item is removed from the DB.

The number of items that can be advertised in a single ADV message is limited to keep messages short and thus to reduce the probability of collisions on the MAC layer. On the other hand, replication performance is improved by letting a node send more than one ADV. In our algorithm nodes may ask any node that replies to their ADV message with a request to issue another ADV message. In the current implementation a node always requests another ADV message with each REQ message sent. This process stops if no items offered in an ADV message are requested or - obviously - when the two nodes leave each others communication range. This mechanism is backed up by the creation of ADV messages if the set of neighbors of a node does not change for a predefined period of time. Figure 5.1 gives an overview of the basic interaction scheme.

5.4.2.2. Advertising Strategies

For a large number of different observations the size of each DB replica will soon be large, making it impossible to advertise all observation records in a single ADV message. Therefore an advertising node has to be able to select a sub-set of the data from its local DB when composing an

ADV message. This leads to the problem of finding an appropriate *selection strategy* that ensures a reliable overall replication process.

As a first approach we applied a strategy mix where information that has never been advertised by a node is selected to be advertised first. If this number is smaller than the number of items an ADV message can hold, the remainder of the ADV message is filled with advertisements of data that has already been sent based on a round-robin strategy in the database. This ensures that new data has priority over data that has already been offered.

In a second class of strategies, we replaced the round-robin selection with a more sophisticated *demand driven selection policy*. When a new record is created by an observer node, it is important to give priority to the propagation of this record in order to support its replication. An approximation for that property can be made locally on any node by taking into account the number of data messages including the particular record, that have already been sent by the node. A low number of such messages indicates that not many replicas have been initiated by the node and therefore priority has to be given to that record when sending advertisements. On the other hand, this indicator is not sufficient when the record has already been replicated on almost every node. In this situation a node that received a copy of an almost completely replicated record *r* late will prefer such a record over a record *r′* that has been replicated only a few times, since the number of data messages that include *r* will soon be outrun by those that contained *r′* and will hardly increase. This is due to the fact that almost no other node will request *r* and more nodes will request *r′*. To take this into account we keep the difference *d=#adv-#data* for every record *r*, where *#adv* is the number of ADV messages sent that included *r* and *#data* is the number of DATA messages. A large *d* indicates that the record has been advertised and only relatively few requests were received that lead to DATA messages. A small *d* indicates that an item has been requested regularly in response to advertisements. To approximate the global replication progress of a particular observation record, the value of *d* calculated by other nodes is taken into account on the reception of every ADV and DATA message. The node receiving such a message re-calculates its own

$d_{new}=(alpha*d_{old})+(1-alpha)*d_{remote}$,

where d_{old} is the previous local value for the observation and d_{remote} is the value for the same record on the node that sent the message. *alpha* is a weight, with $0<alpha<1$ that defines how much remote information is taken into account.

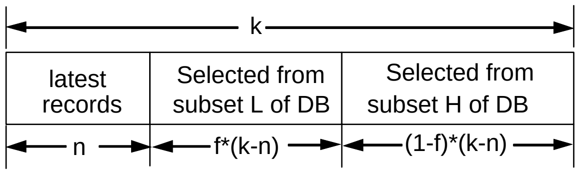


Figure 5.2: Structure of an ADV message for the demand driven selection strategy

Figure 5.2 shows the structure of an ADV message for the second selection strategy determined by a tuple *(k, f, g)*. The message can contain at most *k* entries, where *n* are occupied by new information, just as in the round-robin selection strategy. The remainder is split into two parts, determined by the fraction f with $0<f<1$. The DB is split into two subsets *L* and *H* with

$$(DB = H \cup L) \wedge (H \cap L = \emptyset)$$

L contains a fraction *g* of all records in the local DB such that *d* for all records in *L* is smaller than the lowest *d* of any record in *H*. Records from *L* and H are selected randomly (uniform distribution) to fill f*(k-n) and (1-f)*(k-n) slots in the ADV message respectively. If any subset contains less messages, the remainder will be filled with information from the other set.

5.4.2.3. Randomized Messages Transmission

To reduce the number of messages and to avoid broadcast storms [NTC+99], *randomization* is used to delay messages before they are sent. ADV messages issued by mobile nodes are delayed to avoid that a group of nodes advertises the same observation at the same time and location. REQ messages are delayed, because it is sufficient that one node requests an observation, while other nodes can pick up the DATA message without requesting it. DATA messages are delayed to avoid that many nodes answer a REQ message. Delaying messages in the described way results in a flavor of selective flooding, since not every node that receives a new information entity re-advertises it. Whenever the TTL of an observation expires any node that holds it, drops it.

```
ON_NEW_DATA or ON_NEW_NEIGHBOR:
  a = prepareAdvMsg() // compose ADV message a from local DB
  schedule_for_send(a) // send within a randomized time interval
ON_RCV_ADV(m: AdvMsg):
  p = 1 // probability for ADV requesting
  r = prepareReqMsg(m, 1) // build REQ based on local DB and ADV diff
  if r contains at least one request then
     schedule_for_send(r) // send within a randomized time interval
ON_RCV_REQ(m: ReqMsg):
  d = prepareDataMsg(m) // prepare DATA based on incoming REQ
  schedule_for_send(d) // send within randomized time interval
  if m.sendAnotherAdv then // additional ADV prepared on demand
     a = prepareAdvMsg()
     schedule_for_send(a)
  fi
ON_RCV_DATA(m: DataMsg):
  store(m) // update local DB with requested data
ON_IDLE: // send messages from send buffer
  if first_item( fifo_send_queue ).send_time <= current_time
  then
     send_and_remove(first_item(fifo_send_queue))
  fi
schedule_for_send(m: msg) // send buffer with randomized schedule
  rnd = random_int(k*msg_time, 2*k*msg_time)
  if isempty(fifo_send_queue) then
     append(fifo_send_queue, {current_time+rnd, m})
  else
     append(fifo_send_queue, {last_queue_time+rnd, m})
  fi
```

Figure 5.3: Pseudo-code of our NADD

5.5. Simulation

The proposed algorithm was tested in simulations to evaluate its performance with respect to replication latency, i.e., the time until a certain fraction of the data is replicated on all nodes, and the message overhead imposed by the algorithm.

In order to compare the discussed selection strategies, optimal selection is simulated. Nodes only advertise data that is missing in the database of other nodes. This ensures maximum efficiency in the data exchange, which is only influenced by the mobility of the nodes and the underlying communication technology.

5.5.1.1. Simulation Environment

The simulations were done using a discrete time-step approach. At the MAC layer a simple carrier sense, collision avoidance mechanism (CSMA/CA) prohibits one node to send if it is within the radio range of another node that is already sending. In this case the message is scheduled to be resent later. If retransmission fails for the third time the message is dropped. If both senders are out of each others radio range, simultaneous transmissions are allowed, though the message does not reach receivers in the intersection of both ranges. If two or more senders start sending simultaneously, again messages in the intersection of any two radio ranges are extinguished and do not reach their receivers. ADV and REQ messages have a size of 32 bytes per item advertised or requested. DATA messages have a size of 512 bytes per item transfered. The transmission speed is 128 kbit/s with 10 m transmission range. Mobile nodes follow the random waypoint mobility pattern [BMJ+98] with a pedestrian speed of 3-5 km/h and intermediate stays of 72-120 seconds. Observer nodes are placed in a regular grid and remain stationary during a simulation run. The total area simulated is 100 m by 100 m to represent a large building.

Scenario	Strategy	Max. ADV size	Remark
RR-6	round-robin	6	
RR-12	round-robin	12	
SEL-6	selection	6	*alpha=0.5, f=0.65, g=0.5, k=6*
SEL-12	selection	12	*alpha=0.5, f=0.65, g=0.5, k=12*
OPT-6	optimal knowledge	6	
OPT-12	optimal knowledge	12	

Table 5.1: Scenario Overview

In all scenarios observers can advertise at most 6 or 12 observation records per ADV message. This represents a message size of 224 (=32+6*32) or 416 bytes for ADV and REQ messages and a maximum of 3104 (=32+6*512) or 6176 bytes for DATA messages, respectively. Thereby messages for advertisements and requests are short to keep the probability of collisions low. The TTL of all observation records is set to a value that does not invalidate the item during the time of the simulation. All updates were done by the observer nodes at the start of the simulation. Future investigation will have to evaluate the effect of temporally overlapping replication processes.

All scenarios contain 10 mobile nodes and 9 observer nodes. Each observer node makes 80 observations, resulting in a database size of 720 observation records. The scenarios vary in the selection strategy chosen for advertisements and the maximum number of entries in an ADV message. All simulations were run for 3600 seconds. Table 5.1 gives an overview of the scenarios evaluated.

5.5.2. Replication Latency

This section presents the growth of the database copies carried on mobile nodes over time. The results of Figure 5.4 show the replication latency for the scenarios where at most 6 items can be advertised in an ADV message. With the optimal strategy OPT-6 it takes approximately 800 seconds to perform a complete replication on all nodes. This result solely influenced by the mobility of the mobile nodes, since each advertising node is assumed to know the contents of the database of the node it is offering data to. The round-robin strategy RR-6 uses a simple advertising schedule that only depends on what has locally been advertised before. This results in a very slow propagation, because the advertising behavior of other nodes is not taken into account at all. The demand driven strategy SEL-6 shows improvements over the round-robin strategy and results in a faster data replication, especially in the time span where 40% to 80% of the data is replicated.

Compared to the results described above, the scenario SEL-12, which uses 12 entries per ADV message shows a significantly faster growth of the database copies in the time span where 60% to 95% of the data has been replicated on each node. The optimal scenario OPT-12 shows the same behavior as its counterpart OPT-6, because it is also only limited by the mobility of the nodes. The round-robin strategy shows about the same replication latency in both cases, but varies in the message overhead as described in the next section

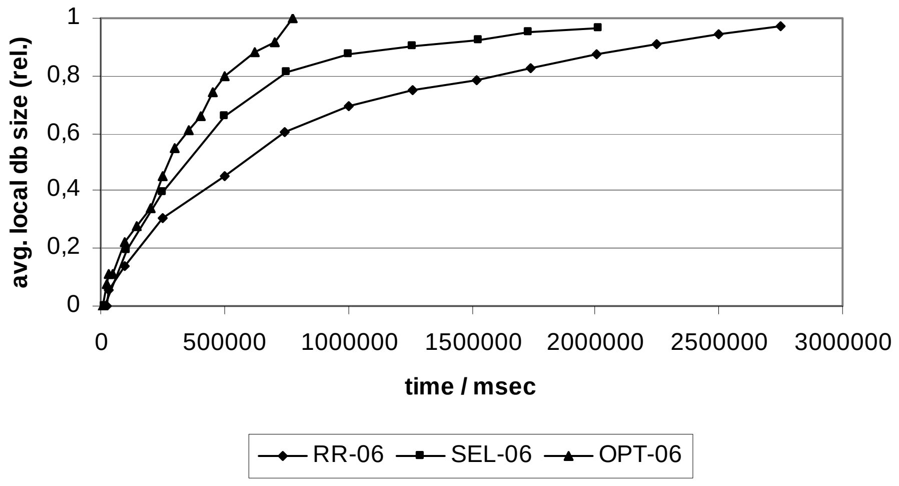


Figure 5.4: Average replication latency with ADV size 6

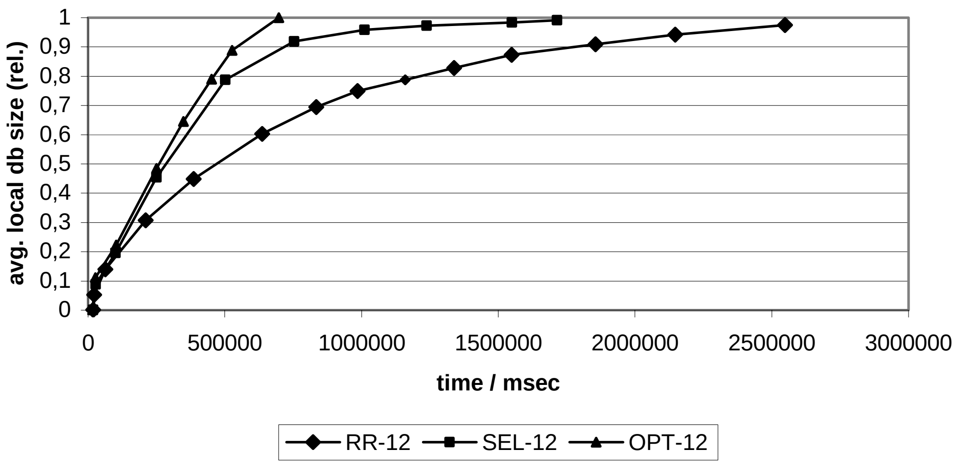


Figure 5.5: Average replication latency with ADV size 12

5.5.3. Message Overhead

This section discusses the message overhead imposed by our protocol. The results of Table 5.2 give an overview of the average number of messages and their total size per node sent by each node in the different scenarios. The message sizes show the average transfer volume per node divided into ADV, REQ, and DATA volume sent. Here the messages sent have been weighted according to their size, where one ADV or REQ entry has 32 bytes, and one DATA item has 512 bytes. Each message has a constant overhead of 32 bytes. It has been assumed that every message includes the maximum of 6 or 12 entries. The optimal strategy has the lowest message overhead, since it only advertises data if necessary. It does not show the same results for ADV, REQ, and DATA messages since messages can be lost due to collision on the MAC layer and the mobility of nodes. The round-robin strategy needs about twice as many ADV messages compared to OPT. Many of those messages do not contain data that is needed and therefore only little more REQ messages are sent compared to OPT. The SEL strategy has the highest message overhead because many ADV messages contain information that is requested and therefore additional ADV messages are triggered. On the other hand this strategy shows a very good replication latency, as stated above.

Table 5.2 shows how much transfer volume would have been needed if, instead of the three-way-handshake, only DATA messages would have been used to propagate the observation records (i.e., in a hyper-flooding approach without negotiation). The advantage of the three-way-handshake over the plain data message approach with respect to transfer volume is 30-40% (see Table 5.2 and Table 5.3).

	Num ADV	**Num REQ**	**Num DATA**	**Size ADV**	**Size REQ**	**Size DATA**	**Total size**
RR-6	225	83	88	50kB	18kB	269kB	337kB
SEL-6	497	241	247	110kB	54kB	768kB	932kB
OPT-6	101	70	75	22kB	15kB	233kB	270kB
RR-12	168	47	50	70kB	19kB	340kB	429kB
SEL-12	377	181	176	157kB	75kB	1091kB	1323kB
OPT-12	54	37	38	22kB	15kB	237kB	274kB

Table 5.2: Message overhead in number and size of messages

Assumed transfer volume using DATA instead of ADV messages	**Num DATA**	**Vol/ kB**
RR-6	225	682
SEL-6	497	1506
OPT-6	101	306
RR-12	168	1013
SEL-12	377	2273
OPT-12	54	325

Table 5.3: Transfer volume with DATA messages only

5.6. Related Work

The SPIN protocol family [HKB99] uses a a three-way-handshake protocol similar to our protocol. SPIN addresses sensor networks, i.e., ad hoc networks with stationary nodes. Since it does not take temporary network partitions into account and therefore does not deal with the resulting problem of choosing a selection strategy the advertisement of data will only work in environments with low node mobility.

In various situations it has been proved that flooding is a robust mechanism to distribute information to all nodes in MANETs. In [HOT+99] flooding has been evaluated as a basis for multicast protocols in MANETs. Hyper-flooding has been proposed as a method to overcome network partitions in ad hoc multicast routing if, besides other parameters, the TTL for a message and the approximate network diameter are known [OT98]. In our protocol, the replicated model data already has a TTL included in its meta-data. The diameter of the network is not needed because nodes can decide to drop information solely based on the TTL, since model data is assumed to be valid for a long period of time in comparison to messages used in routing protocols. Additionally, our protocol does not perform hyper-flooding on a per-message basis but on the basis of a three-way-handshake, where advertisements are hyper-flooded by re-advertisements according to the selection strategies.
In [VB00] an epidemic protocol was introduced to solve the routing problem in a partially connected network. They use a similar mechanism to exchange information between two neighbor nodes. However, their goal is to deliver messages to any node without establishing a route between sender and receiver and not the replication of model data. The data considered is typically short-lived, i.e., if routing of a message fails a retransmission is issued.
A combination of so called rumor-mongering and anti-entropy is used in [DGH+87] to replicate information in databases in wired networks. In our protocol, we combine new information and, if free space is available in an ADV messages, older information. This results in a partial anti-entropy session, because some differences between hosts are resolved with new information first (i.e., rumors) and older information (i.e., part of the anti-entropy).

In [XWC00] the distance between any two versions of a data item and the communication cost is used as the basis for a cost model in order to determine the estimated benefit of forwarding the data. In this approach it is necessary that every node has a notion of „distance" which depends on the semantics of the data. The authors also make the assumption that only a single node updates a particular object.

5.7. Conclusion

We presented a protocol for information dissemination in mobile ad hoc networks. The protocol replicates information stored in local databases of nodes. In order to reduce the data transfer volume, negotiation is used to advertise and request data among mobile nodes. Network partitions, as they appear due to node mobility or low node density, can be tolerated since data is advertised more than once. The selection strategy that determines which data is re-advertised, influences the performance of the protocol with respect to the propagation latency and the data transfer volume. The demand driven selection policy shows a reduction of the data transfer volume by 30 to 40% compared to a plain hyper-flooding approach which does not use negotiation. The replication latency performs nearly optimal till 80% replication of the data is achieved and slows down for the last 20%.

So far, we have investigated the impact of different data selection strategies on replication latency and message overhead. In future work, we will investigate what parameters can be used to adjust the hyper-flooding nature of advertisements, e.g., depending on the node density, in order to achieve further reduction of advertisement messages in dense networks. Mobility models of mobile nodes have impact on the performance of routing protocols [THB+02]. We will examine the impact of mobility models on our protocol and the improvements that can be made if such knowledge is exploited.

6. From Home To World: Supporting Context-Aware Applications through World Models

In the vision of pervasive computing smart everyday objects communicate and cooperate to provide services and information to users. Interoperability between devices and applications not only requires common protocols but also common context management. In this chapter we discuss requirements on the context management based on the Georgia Tech's Aware Home environment and the global context management perspective of the Nexus project. Our experiences with integrating the Aware Home Spatial Service into the Nexus platform show how federation concepts and a common context model can provide applications with uniform context information in different administrative and application domains.

6.1. Introduction

Pervasive computing has drawn increasing attention of researchers in the past years. As a result, a multitude of applications has been developed. These applications cover different domains, such as tourist guides [CDM+00a],[AAH+97] indoor information systems [Cooltown],[CKW01] and smart environments, e.g., the Georgia Tech Aware Home [KOA+99], to name a few. A variety of supporting infrastructures have been proposed, which facilitate the development of applications. However, these infrastructures mostly address a distinct application domain, such as context processing based on sensors [SDA99] or providing application-specific context [STM00], [CDM+00a].

However, one cannot deploy any of the above-mentioned applications and expect them to cooperate or share resources. Also, when new services, hardware or environmental information such as maps become available to an application, other existing applications can not automatically use them. Interaction between different applications based on their context, e.g., identity, location, or state, is not possible if they do not rely on a common representation of this context.

In this chapter we discuss the requirements on application-independent context management in order to provide a platform where applications can seamlessly share their context. Two approaches for such platforms are presented. One addresses the context management in the scope of a smart home environment based on the Georgia Tech's Aware Home project [AAH+97] while the other approach addresses global context management from the perspective of the Nexus project [HKL+99]. We have integrated both approaches and discuss our experiences regarding the context representation. Crucial to the integration of both approaches is a common context model and a federation concept for the local context management.

The chapter is structured as follows. First we motivate the requirements on context management to achieve interoperability based on typical applications in a smart home environment. Following to that, the related work is presented. In section 6.4 and section 6.5, we describe the context management for the Aware Home and the Nexus platform. An assessment of context modeling and the platform integration is given in section 6.6 based on the integration of the Aware Home Spatial Service into the Nexus platform. Finally, we close with a summary and outline future work.

6.2. Requirements

In this section we derive requirements based on a scenario and discuss what follows from that for context modeling.

6.2.1. Scenario

Consider the following scenario: You have a home that can sense what its inhabitants are doing, adapt to them and support them with their task; e.g., the Smart Intercom allows you to reach every other person in the house - regardless of where he or she is located. So if you are 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 currently acceptable to contact him 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.

Now you want to add another application to your home: A smart doorbell. 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 you can decide 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.

Also, you want to add a smart alarm system. 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 accordingly. Then it would provide the authorities with information about the situation within your house - e.g., the layout of the house, where the inhabitants and where the intruders are located.

In the remainder of this section, we discuss requirements for the data model of these applications and for the infrastructure that supports them.

6.2.2. Derived requirements

Based on the scenario we derive requirements regarding the information model, information access, consistency, the abstraction from resources, and interoperability.

Information Model:

The applications in the scenario 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 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 consistent among all applications. If real world objects or resources are moved or removed, the information in the infrastructure 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. First, 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. Secondly, 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. Thirdly, 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.

6.2.3. Context modeling

The information model as discussed in the previous section is concerned with context data. A number of different definitions for context exist, e.g., [CK00][DA99]. Our view on context is similar to the definition presented in [DA99]: *Context is the information which can be used to characterize the situation of an entity. Entities are people, locations, or objects which are considered to be relevant for the behavior of an application. The entity itself is regarded as part of its context.*

When context is stored in a context model it is necessary that applications can access it. As the examples have shown, applications typically consider information about a distinct location or an entity, e.g., a user or object. Hence, such a model must allow for access to context information based on location and the identity of objects. Additionally, context models can incorporate the time dimension, i.e., capture history or provide a prognosis. 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. Based on the selection of objects via identity, location, and time further information, such as the activity of a user or the state of an object, can be derived. Using location as index imposes new challenges on the representation of the index and its processing. The underlying spatial structure has to be reflected in the index.

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 our context models take the spatial structure of the real world as their basis, we also call them *world models*.

6.3. Related work

Context-aware applications have attracted the interest of researchers over the past years. A number of different applications have been developed exploring different application domains. Tourist guides, e.g., Guide [CDM+00a] or CyBARguide [AAH+97], provide information to mobile users about typically stationary objects in their vicinity, such as distinct sights. Indoor information systems, e.g., Cyberguide [AAH+97], ETH World [ETH], or conference room reservation [CKW01], provide similar services in the indoor domain.

Another class of pervasive computing applications presents location-dependent information to users. Virtual Information Towers [LKR99] provide information for a distinct area through posters, while Stick E-Notes [Pas97] and Geo-Notes [EPS+01] offer a Post-It metaphor allowing users to leave messages at a distinct location. These systems typically are not concerned with the relations between different mobile objects, e.g., users, and they only require simple spatial models. Abstraction from resources is typically not necessary, since the information is not captured via sensors and actuators are not supported. Interoperability is typically not an issue in such systems.

For another class of applications, an underlying spatial model is required. The Teleporting project [HHS+99] requires a spatial model in order to redirect output to the devices that are closest to users. Similar to that, the Family Intercom [NKO+01] also requires spatial knowledge. While the Family Intercom does not intend to share its spatial model, the Teleporting project has developed a platform which provides a data model linked to locations obtained via fine-grained location and spatial monitoring systems. This platform already meets some of our requirements, but its scalability is targeted at larger buildings. There is no given structure of data objects though, leading to reduced interoperability.

The Context Information Service (CIS [JS03]) of the Aura project [GSS+02] also aims at providing context information. The underlying location model is based on a mapping onto geometric coordinates [JS02] thus not allowing purely symbolic coordinates. The information model is targeted at technical information of the network and its entities, such as printers or bandwidth, and reflects their properties via meta data. The location of people is also provided. This centralized architecture is limited to building size scalability although caching techniques provide improved performance.

Other kinds of infrastructure are targeted at distinct domains. The Context Toolkit [SDA99] provides abstractions to integrate various sensor devices and facilitates sensor fusion. Guide [CDM+00a] also provides an infrastructure which enables classes of applications from the navigation or outdoor domain to rely on an extensible data model. However, the location model is cell-based and depends on the communication infrastructure.

Most of the above-mentioned approaches do not allow for flexible integration of different spatial models. Such spatial information is represented by location models, reflects application requirements, and depends on the modeled environment [BBR02]. Easy Living [BMK+00] is targeted at the indoor domain and provides an infrastructure for spatial access to information similar to QoSDream [NC01]. Abstractions of hardware providing context information or a common data model are not addressed by either approach.

As we have seen, there are a variety of different approaches which address some of the requirements derived in Section 2. A comprehensive solution is still missing. In particular, the scalability of the infrastructure and interoperability of applications are still open research questions. The answers to these questions are crucial to provide users with applications which operate in varying environments.

6.4. AHSS - A local architecture

We have designed and implemented a context management infrastructure, the Aware Home Spatial Service (AHSS), at the Georgia Tech Aware Home, based on the requirements presented in section 6.2 and the feedback from the developers at the Aware Home. One of their main requirements for developing applications was flexibility. As their research environment is very dynamic, they did not want to be obstructed by too inflexible frameworks and standards, preferring direct interaction and agreement between developers instead. Scalability, on the other hand, is not such a big issue for prototype applications in the Aware Home. Therefore, the Aware Home Spatial Service is targeted at small to mid-size smart home environments.

The "intelligence" in such settings is usually not concentrated in mobile devices like PDAs, cellular phones or wearable computers. It is typically in stationary systems in the house that serve the area of the house. In the smart home we are considering, a variety of platforms are in use. The main language for most systems in our setting is Java. Some application areas like computer vision necessitate the use of other languages like C or C++.

6.4.1. Spatial 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, similar to the one proposed in [BBR02] that we have adapted and extended by topographic aspects.

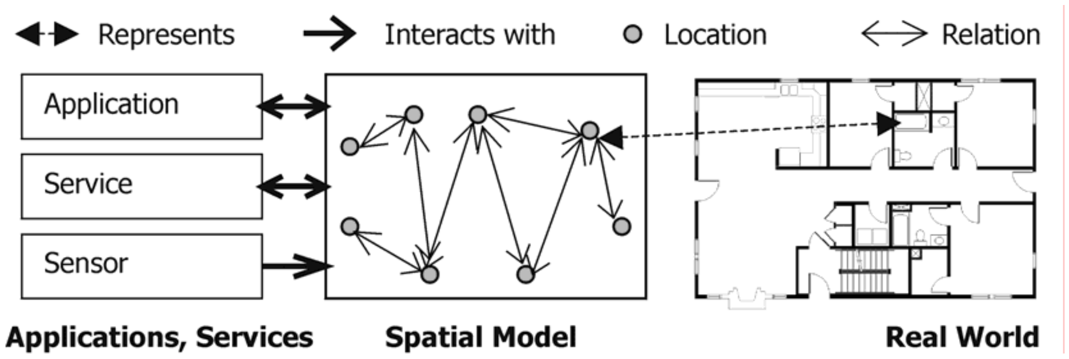


Figure 6.1: AHSS conceptual view

As shown in Figure 6.1, 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, 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 [OpenGIS]. 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, to allow modeling of explicit spatial relations. They have a type and a weight that can be used to model distances that are not geometric: two rooms might be very close from a geometric point of view, but if there is a wall between them, or 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 (LOD) that 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, e.g., the layout of the rooms in the house. Figure 6.2 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.

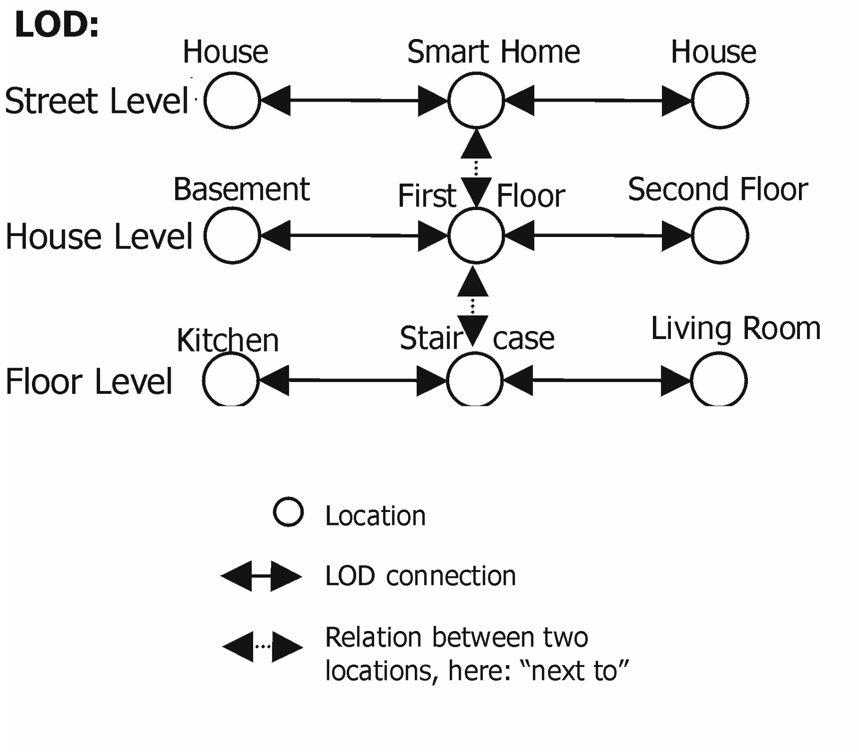


Figure 6.2: Levels of detail in the AHSS spatial model

6.4.2. System architecture

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, that stores the current spatial model (see Figure 6.3). This centralization allows us to keep the architecture of the system relatively simple. However, it should be noted that a centralized design is not required by our spatial model. Sharing and consistency of our model could also be provided by a distributed infrastructure.

The spatial server has to store a potentially large amount of information that has a spatial component in a persistent way. Instead of implementing a proprietary data storage, we rely on a database system. As we have seen in section 6.2.3, the information we are dealing with is structured by space. Therefore, the AHSS spatial server uses a spatially-enabled database. This allows us to model the spatial components of our data much more naturally. It also improves query performance because many spatial databases support spatial indexing. There is a variety of commercial spatially-enabled databases available (e.g., IBM's Spatial Extender for DB2 or Oracle Locator), but those products tend to have a large footprint and are generally expensive. We decided to use the open source database PostGIS [PostGIS], a spatially enabled version of PostgreSQL.

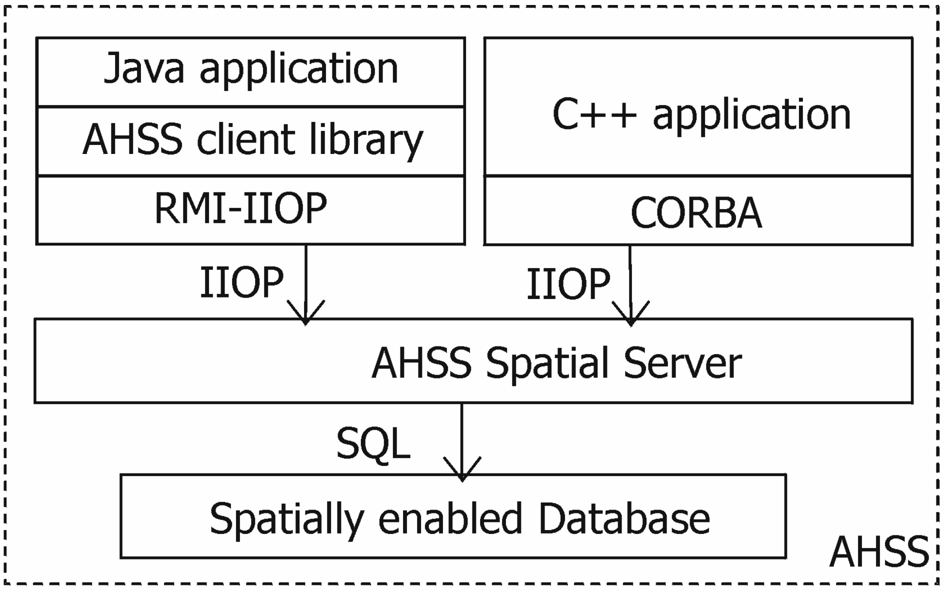


Figure 6.3: Architecture of the AHSS

The communication between the spatial server and its clients utilizes IIOP. This allows Java clients the access via RMI/IIOP, while other clients, e.g., written in C++, can use CORBA. Using RMI-IIOP callbacks, the infrastructure also offers an event concept. Applications can define events on locations in the spatial model. Other applications can then subscribe to those events, and get notified when the events are triggered.

6.4.3. Experiences

The Aware Home Spatial Service provides a simple infrastructure to model context information that is structured along the dimension of space. It allows us to store spatial information in a central infrastructure and to reuse it across multiple applications. Applications can exchange information through the spatial model and they can notify each other using events.

As shown in Table 6.1 AHSS meets our requirements as they were introduced in Section 6.2.

Requirements	
Requirement	Realization
Information Model	Support for location and identity as indexes - allows queries for stationary and mobile objects based on the spatial layout of the smart home.
Information Access	The spatial layout of the AHSS allows the calculation of spatial relations, e.g., the distance between two objects. Querying and updating of information is supported by the database.
Consistency	The centralized architecture of the AHSS provides a consistent view on the modeled information.
Abstraction from Resources	Since applications only rely on the information stored in the AHSS the information source, e.g., a sensor device or a fusion of sensor data, as provided by the Context Toolkit [SDA99], is shielded from the information consumer. Resource discovery is automatically provided by processing queries on the database which is constantly updated.
Interoperability	Applications sharing information stored in the AHSS rely on the same information model and thus become interoperable. However, the AHSS does not define any interoperability protocol between applications themselves.

Table 6.1: Requirements

Although all of our requirements are met, there are some issues worth noting. The AHSS does not rely on an explicit model of the context information that is managed. Different instances of an AHSS, as they can occur in different administrative settings, will therefore not automatically become interoperable. Hence, the interoperability of application gets restricted to a distinct information model of the underlying AHSS.

However, the AHSS is a suitable platform for local context management. In the next section we discuss a global context management architecture. A common context model allows the integration of different local context models into a federated, global context management platform. In Section 6 we show how AHSS can be integrated into such a global context model.

6.5. Nexus - A global architecture

If we want to extend the scope of interoperable, context-aware applications beyond a closed environment like a single home, we need a common context model that is suitable for a wide range of context-aware applications. To achieve this goal, we need 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).

Since no single person or corporation will model the entire world in detail, our approach in the Nexus project at the University of Stuttgart is to investigate how context-aware applications can be supported by the federation of local context models. Hence, the infrastructure should allow for the integration of context models from different providers.

The federation of different context models from a variety of providers requires a common information model which states the type and structure of information objects as well as the access to the objects, i.e., a unified model of object identities and the location model.

For the Nexus platform, the information model is called Augmented World Model. The Augmented World Model is a global object-based ontology 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) [LKR99] or virtual Post-Its [Pas97] that are used to represent digital information in the world. The Augmented World Model is not stored explicitly. Instead, it is a federated, global view on all compliant local context models. Those local models are called augmented areas (AA) (see Figure 6.4). 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. 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).

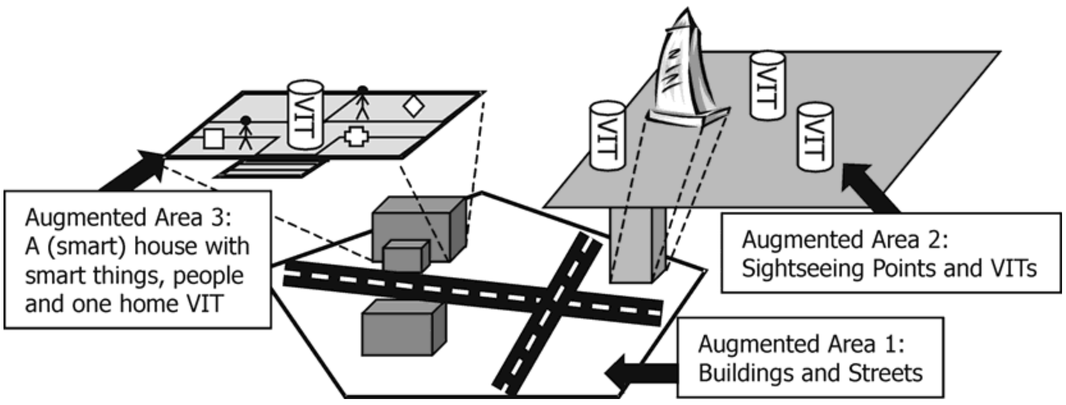

VIT
VIT
VIT
VIT
Augmented Area 3: A (smart) house with smart things, people and one home VIT
Augmented Area 2: Sightseeing Points and VITs
Augmented Area 1: Buildings and Streets

Figure 6.4: Augmented Areas

The SCS defines the types and attributes of objects of the Augmented World. These are ordered in a hierarchical "is-a" relationship (inheritance). In Figure 6.5, 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 Nexus Object Locator (NOL). Objects of the type *spatialobject* all have a geographical position and optionally an extent. Because the spatial context is our primary index and therefore is crucial to performance, we distinguish between static and mobile objects.

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. We have designed that model by doing a use case analysis on different applications [NM01]. An AA provides the attribute, the type and the meaning of the attributes of its objects.

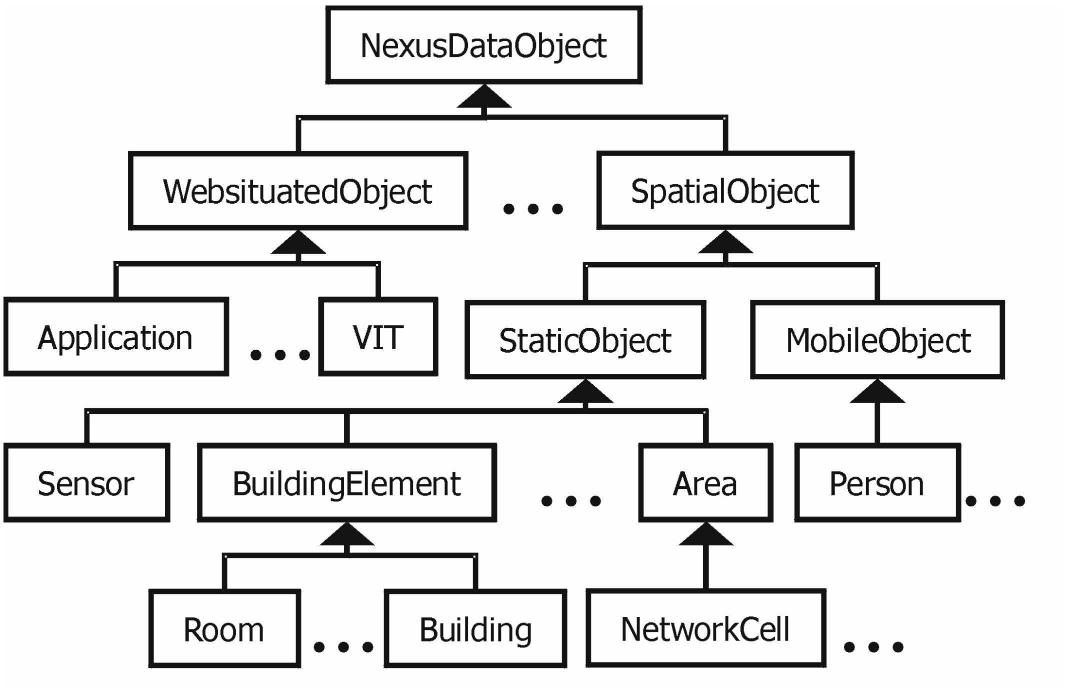


Figure 6.5: Excerpt from the top part of the Standard Class Schema

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 into an AA, then she can extend the SCS type Room with the necessary additional attributes (see Figure 6.6). An advanced LivingRoom application can now access this information, whereas normal applications knowing only the SCS could treat the object as its parent class Room.

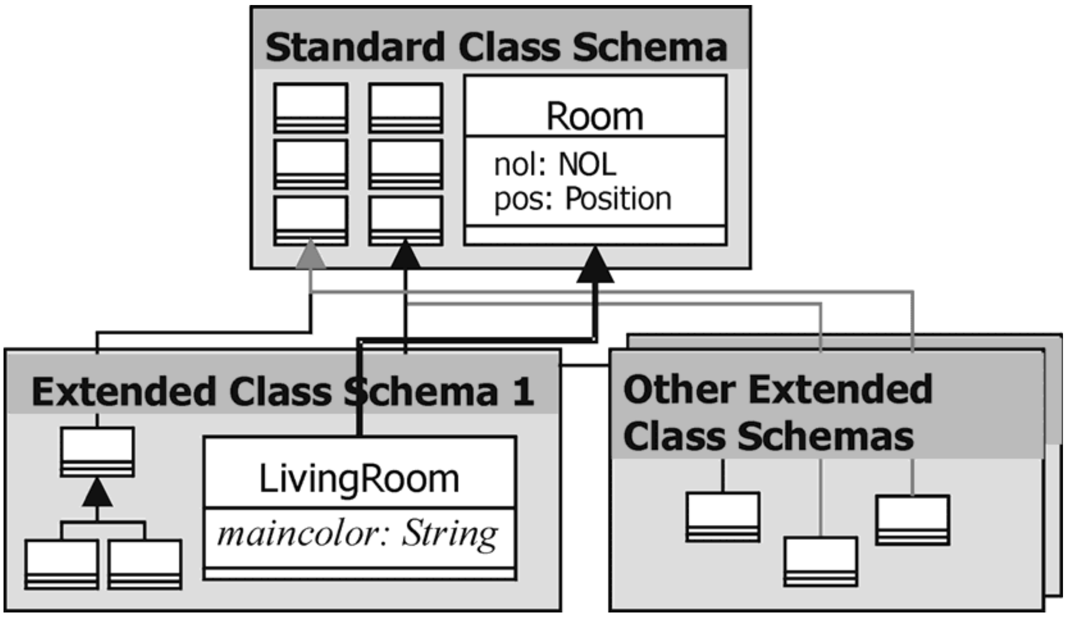


Figure 6.6: Standard and extended class schema

6.5.1. Platform architecture

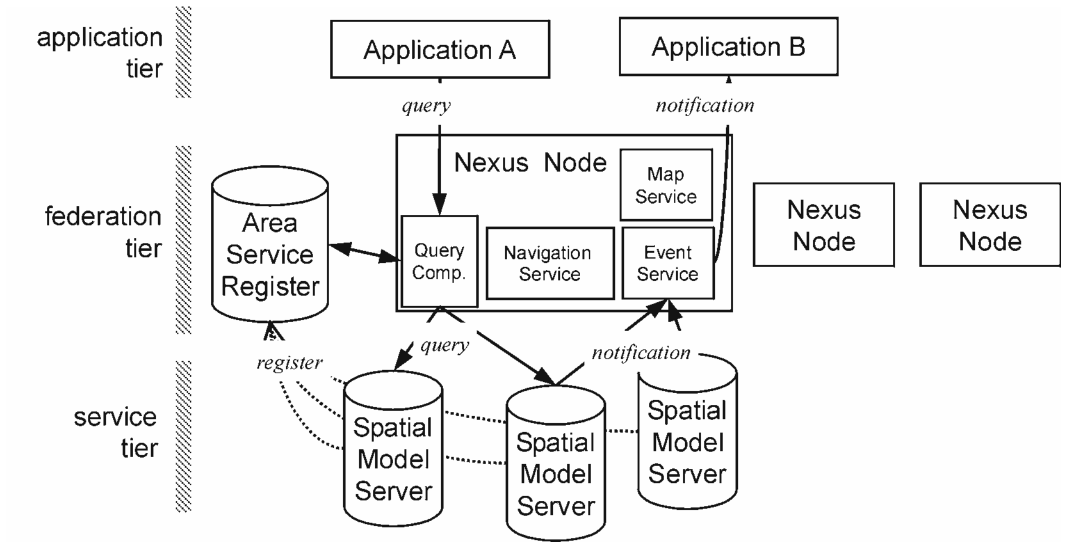


Figure 6.7: Architecture of the Nexus platform

As we have seen, 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 we will describe an open infrastructure that is able to manage the Nexus Augmented World Model. As depicted in Figure 6.7, the Nexus Platform 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.

6.5.2. Spatial model servers

Spatial Model Servers (SpaSes) host Augmented Areas. Objects are modeled in AWML (Augmented World Modeling Language) and queried using a simple spatial query language called AWQL (Augmented World Query Language), which is used to specify the objects of interest and to filter the attributes. AWQL supports spatial predicates (overlaps, inside) and nearest neighbor queries as well as data manipulation (insert, update, delete). AWML and AWQL are both XML languages. Note that AWQL 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, we use a full-grown database with a spatial extension (IBM DB2 7.1 + Spatial Extender). AWQL can be easily transformed to SQL by an XSLT style sheet. Mobile objects are managed by the Location Service [LR02]. 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.

6.5.3. Nexus nodes

The Nexus nodes in Figure 6.7 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 Augmented World 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 augmented areas and stores the address of the server, the available object types and the extent to the AA. The Nexus node computes the targeted region and the object types of the application query and queries the ASR to find out which Spatial Model Servers cover 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. More details about the federation tier can be found in [NGS+01].

6.5.4. 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 their own interface. In Figure 6.7, you can see three different value-added services of the Nexus platform: the event service monitors spatial events, combining basic events into more complex events. This allows the processing of spatial predicates, such as "two of my friends meet". The map service provides maps based on a selected area and the navigation service provides a navigation route from a starting point to an endpoint.

6.6. The NexusScout application

In this section we briefly describe NexusScout, a location-based application that runs on the Nexus platform. It is based on the Virtual Information Tower application [LKR99]. We have added several advanced functionalities that show the power of the platform and are useful for mobile, context-aware applications. NexusScout runs on a notebook and we have also just ported it to a PDA. It uses WLAN for wireless communication. Outdoor positioning is done via GPS, while indoor positioning uses infrared beacons.

The NexusScout provides maps to users showing their position. Virtual Information Towers (VIT) provide information (web pages) that is relevant at the given location. Based on the Nexus Augmented World Model nearest-neighbor queries for objects, e.g., restaurants, are possible. The integration of the navigation and map services allows to use the

Neuxs as navigation system as well as register spatial predicates, e.g., "notify me when my colleague enters the building".

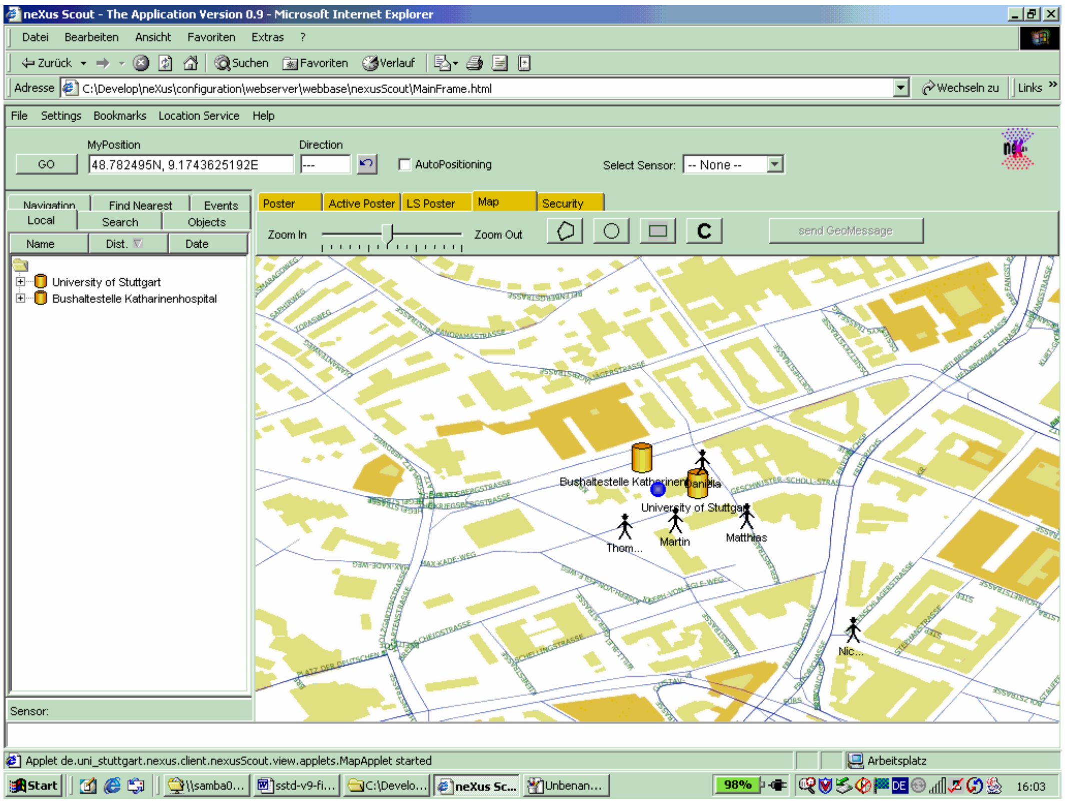

neXus Scout - The Application Version 0.9 - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Adresse C:\Develop\neXus\configuration\webserver\webbase\nexusScout\MainFrame.html
File Settings Bookmarks Location Service Help
MyPosition
GO
48.782495N, 9.1743625192E
Direction
AutoPositioning
Select Sensor: -- None --
Navigation
Find Nearest
Events
Local
Search
Objects
Name
Dist.
Date
University of Stuttgart
Bushaltestelle Katharinenhospital
Poster
Active Poster
LS Poster
Map
Security
Zoom In
Zoom Out
send GeoMessage
Bushaltestelle Katharinen
Daniela
University of Stuttgart
Thom...
Martin
Matthias
Nic...
Sensor:
Applet de.uni_stuttgart.nexus.client.nexusScout.view.applets.MapApplet started
Arbeitsplatz
Start
16:03

Figure 6.8: Screenshot of the NexuScout

6.7. AHSS in Nexus

As we have seen in Section 6.4.3, a local context model like AHSS is suitable for small- to mid-size home environments, where flexibility is very important to quickly develop and evaluate prototype applications, and where developers can easily interact and agree on the modeling. However, if we have applications for which the context is to be shared over larger areas and possibly multiple administrative domains, e.g., multiple aware homes, the AHSS is no longer sufficient, as it does not provide the necessary scalability and agreements between developers on how the world should be modeled. Our approach to mitigate this problem is to integrate AHSS into our Augmented World Model.

6.7.1. Conceptual integration

We decided to integrate AHSS into Nexus as a spatial 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. 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 our approach, we map Nexus objects to AHSS Locations. 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. We are now going to describe the changes we have made to AHSS in order to integrate it into Nexus.

6.7.2. Technical integration

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 we had to implement an AWQL interface 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 Augmented World Model (AWM), the Standard Class Schema plus appropriate Extended Class Schemas, which are both modeled in XML. We have therefore added an AWM interpreter which reads the appropriate AWM specified in the query. It then handles the AWM 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 6.9 gives an overview of the necessary changes to the AHSS architecture.

6.7.3. Experiences

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. For example, the NexusScout can use AHSS without any modifications. 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.

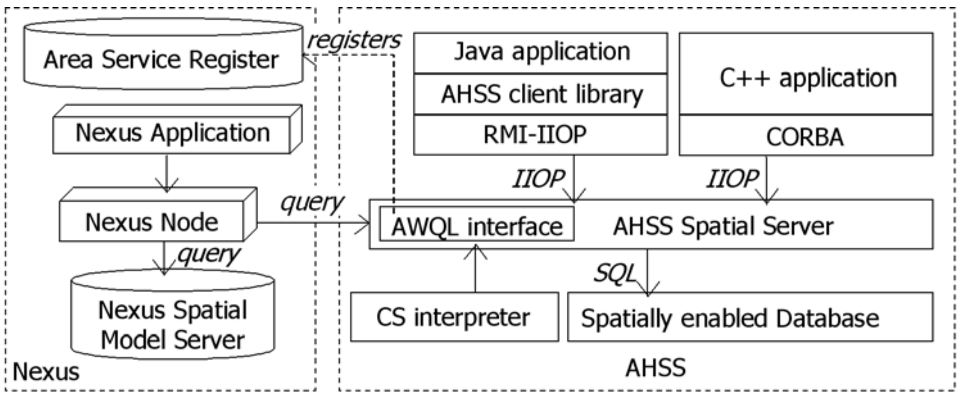


Figure 6.9: The extended AHSS architecture

6.8. Conclusion and future work

In this chapter we have shown that shared context models are a suitable basis for building context-aware applications that operate in the same environment and rely on the same context information. We have presented the Aware Home Spatial Service (AHSS) that was designed for a single smart home environment. For context management on a global scale, we have developed the Nexus platform that federates different context models based on a common standard for modeling objects and a topographic modeling of space. Finally, we have shown how an existing local spatial model like AHSS can be integrated into the Nexus platform.

The current version of the extensible Nexus standard class schema is only the first step towards defining a common standard for context models in general. The integration of further application models currently used in smart environments will lead to a better understanding of world models and how to build them. To support further classes of applications we have to go beyond modeling sensor information as model data and integrate hardware abstractions and discovery mechanisms into our model, e.g., provide access to standardized interfaces for complex actuators and sensors like cameras. So far the Nexus platform has focused on providing efficient and scalable access to two dimensional topographical world models. In the future, we will investigate complex three dimensional models and also full-fledged support for topological models. Federating different models modeling the same real world objects ensures internal consistency of the model information. Consistency between the model and the real world is also an important focus of our future research.

For a more widespread use of world models in smart environments tools for creating such models have to be developed. Having larger smart environments allows more useful evaluations in real world situations. We plan to investigate security, privacy and acceptability issues in such settings.

7. Middleware and Application Adaptation Requirements and their Support in Pervasive Computing

Pervasive computing environments are characterized by an additional heterogeneity compared to existing computing infrastructures. Devices ranging from small embedded systems to fullfledged computers are connected via spontaneously formed networks. In this chapter we analyze requirements of applications and system software to cope with the dynamically changing execution environment. Based on our microbroker-based middleware BASE a component framework for pervasive computing supporting application adaptation is proposed.

7.1. Introduction

In the recent past, middleware platforms have been the target of researchers in order to provide flexibility with respect to the configuration of the middleware itself. Requirements on such reconfigurable middleware systems arose mainly from the domain of Quality of Service (QoS) management. Different application requirements on non-functional aspects, such as QoS, lead to mechanisms of the middleware to ensure a distinct QoS property.

The vision of ubiquitous or pervasive computing adds new complexity. Our everyday environment becomes populated with smart everyday items. That is, processors are integrated into the environment and allow to access information related to the real world as well as to control distinct functionality. The end systems in such scenarios are far more heterogeneous than in classical computing environments. Sensors will only need limited computing and communication capabilities and other devices will be dedicated to a single purpose, i.e., a presentation system in a video projector might contain a fullfledged computer but its software is specialized for presentation management. Besides the involved devices, the communication technology will differ as well, ranging from infrared connections over radio links to computers connected via static links. The resulting network topologies will frequently change due to user and device mobility. Information and services available are bound to the location of the device, e.g., temperature information or a presentation system of a far away place are typically less interesting than those available nearby.

As a result, the requirements on adaptation and configuration of the underlying middleware as well as those from the applications change compared to those requirements already present in classical middleware systems. In this chapter we will present an example scenario, derive and motivate requirements on middleware configuration support and application adaptation. Our approach to support these requirements via a microbroker based middleware – BASE – and a component model based on an application framework is then presented. After a discussion of related work the chapter closes with a summary and outlook on future work.

7.2. System model

This section will first present a pervasive computing scenario and two possible applications before the system model is defined.

7.2.1. Scenario

Let us consider a scenario as it is common in the envisioned pervasive computing systems. Present in such a scenario are embedded and specialized devices, e.g., sensors providing information about temperature, position of users or specialized systems, such as the before mentioned presentation system. All these devices are equipped with wireless communication. Along with these stationary devices, mobile devices which are typically carried by users are present. Such devices could be handheld devices, such as personal digital assistants (PDAs) or cell phones, but in the future there might be smart clothes as well. The computing environments of today will not vanish or be substituted by these devices but complement such systems. In order to motivate the use of such environments to applications two possible applications are sketched:

Support of senior citizens: in order to support the life of the elderly in their home, their body functions and positions might be captured and evaluated at a designated home server. If a change in the health condition occurs, information how to behave is presented through audio or video devices in the room where the person currently is located. In serious health conditions an ambulance is called and provided with the health status of the person.

Office support: the status of rooms and objects could be monitored by sensors and propagated into the vicinity. Users nearby are thus provided with environmental information as well as vacancies of meeting rooms etc. Additionally, locally available services become accessible when a user is nearby, e.g., a presentation system is only of use, when the user is in the same room to make use of its output.

Before we will derive requirements from these scenarios the underlying system model will be presented.

7.2.2. System Model

Pervasive computing environments can be classified by the involved devices and the network characteristics. Furthermore, applications depend on the abstractions provided by the underlying operating system or middleware – which is referred to as system software. We will briefly sketch the characteristics of these three topics in the remainder of this subsection.

7.2.2.1. Devices.

As stated above, devices range from sensors over specialized systems to full fledged computers and mobile devices. Besides their processing and storage properties – which may differ widely – devices provide different capabilities which can be used by applications running on these devices. Examples are sensors, e.g., temperature as well as positioning, display or input capabilities, or some controlling capability, such as dimming the light or adjusting the blind of a window.

The availability of a device capability might be restricted in space and time. A GPS sensor is not likely to work within a building and at night sensors based on daylight will stop operating.

7.2.2.2. Network.

The wireless connections between the devices differ with respect to the underlying technology and their characteristics. The most profound difference to classical computing environments is the spontaneous nature of such networks, which are formed by nodes which are temporarily in each others communication range. Obstacles, user mobility, and power saving are common events which lead to a reconfiguration of a spontaneous network.

As a result, services located on a device that is not in the current spontaneous network of a client, are not available. This prevents the usage of centralized lookup or trading services. During the interaction with a service, the device providing the service might leave the network. Since devices can be equipped with different network interfaces, spontaneous networks will overlap, i.e., some devices might be reachable via more than one network interface at a time.

7.2.2.3. System software.

The support of system software clearly may differ widely. Specialized operating systems for embedded devices, common operating systems with middleware support, or completely proprietary solutions are present. From an application point of view, the abstractions how to interact with remote services – a typical middleware responsibility – and how to access device capabilities, which is an operating system task, are important in order to be comprehensive and yet easy to use.

Another relevant issue in distributed systems in general is interoperability which is typically achieved by relying on interoperability protocols. Interoperability protocols reflect the communication model of the application as it is supported by a middleware. Remote method invocations are reflected by request/response messages while events can be realized by oneway messages containing the event. Requirements on the underlying transport, such as an error-free connection-oriented channel, lead to a restricted usage if only oneway communication – via an optical link, or connection-less communication – is available.

7.3. Requirements

In this section we will derive requirements on the adaptation of applications and its support by system software and component models.

Applications are considered to be executed in a distributed way. Standalone applications could require adaptation support as well, e.g., when a device capability becomes unavailable (a GPS sensor indoor), but these kinds of adaptation requirements are a subset of the more general ones of distributed applications.

7.3.1. Application adaptation requirements

Applications in a spontaneous networking environment have to cope with:

Changing service and device capability availability: With devices becoming available their services should be used by an application. As well, if a service becomes unavailable, an alternative service should be selected. This will only work, if applications are composed of services with clear dependencies. If alternative levels of an application are defined which require different services the application can continue as long as at least on set of services is available. Clearly, this cannot be supported by the middleware alone but requires an appropriate framework for applications. The same mechanisms can be used to address fluctuating sensor availability on a device.

Different abstractions for programming of device capabilities: Middleware and operating system abstractions for remote services and device capabilities are typically different, e.g., proxy objects vs. system calls. This hardens adaptation, since the switch of a local to a remote device capability cannot be done with the same programming interface.

7.3.2. System software adaptation requirements

System software in a spontaneous networking environment has to support:

Device lookup and service discovery for spontaneous networks: The device lookup depends on the underlying network characteristics and thus requires distinct lookup mechanisms for each supported network interface. Additionally, services might require distinct interoperability protocols which also depend on the network interface and hence the service lookup will have to take this into account. The detection of lost devices and thus the unavailability of all services in use on that devices have to be signalled to the application or an application framework.

Flexible protocol support and selection: If a network interface looses its connection to another device, communication should be upheld if other network interfaces can provide a communication channel. Switching between different interoperability protocols over networks with different characteristics however requires adaptation if the underlying transport does not fullfil requirements of the interoperability protocol, e.g., IIOP requires connection-oriented error-free/signalling communication.

Decoupling of application communication model and interoperability communication model: In order to allow different communication links for outgoing and incoming messages, the application communication model, e.g., RPC or events, should be kept independent from the communication model of the possible interoperability protocols. For example this allows communication over infrared via sending out a request as an event and receiving the reply as a reply message over an RPC interoperability protocol based on TCP and IEEE 802.11.

Uniform abstraction for device capabilities and services: This allows applications to access remote capabilities in the same way as local ones. Moreover, a uniform abstraction to access services and device capabilities allows to mask the heterogeneity of devices.

Flexible integration of adaptation mechanisms: Since different application requirements will need support through mechanisms, e.g., to migrate a component to a remote host to increase the application performance or to migrate it to the local node in order to save energy, different mechanisms should be easily integrated, configured, and used either directly by an application above the system software or by an application framework.

A system software offering the above mentioned support is not sufficient to help application programmers to conquer the heterogeneity and dynamics of pervasive computing environments. Instead of programming towards middleware mechanisms and selecting distinct mechanisms manually, application programmers should rely on high level policies which will result in combinations of mechanisms of the system software. Examples for such policies are 'EnergySaving', leading to fostering local execution of application components and restricting radio communication that is costly in terms of energy, or 'IncreasingAvailability', which would make extensive use of remote services in order to allow the application execution – as a tradeoff to energy.

What is needed in addition to a middleware supporting the requirements stated above is an application framework that will provide benign abstractions for choosing appropriate adaptation policies. In order to support such a framework, we have developed a microbroker-based middleware, BASE [BSG+03], which meets these requirements. Currently we are developing a component system based on BASE which will allow the specification of component dependencies.

In the following we will sketch the design of BASE and the component system, which we are currently developing.

7.4. BASE a Microbroker based Middleware

Our middleware BASE is intended to be a minimal platform suitable for small embedded systems but extensible to make use of abstractions available on resource-rich environments. BASE provides application programmers with suitable abstractions to conquer the heterogeneity in pervasive computing environments. Another objective of BASE is to form a foundation for an adaptation supporting component framework. We will briefly sketch the overall architecture of BASE. More detailed information is available in [BSG+03].

The major design decision in BASE was to choose a microbroker design. Device capabilities as well as local and remote services are uniformly accessible via invocation objects, which carry the target object, method-name, parameters, and a service context indication special handling of the invocation, such as QoS parameters. The microbroker takes incoming

invocations and dispatches them to either a local service via a skeleton, a remote service via a transport module which connects the local and remote device, or to a device-local device capability. Hence, remote device capabilities can be accessed as services as well.
Invocation objects can be created manually or – if a service provides a stub object – through a proxy (stub object) as conventional middleware systems typically provide. The microbroker is responsible for synchronizing the caller and issue invocations and receive possible replies as well as an invocation. This allows the application to choose different communication models, such as remote procedure calls (RPC), deferred synchronous RPCs, or events via stub objects. Furthermore, the utilization of different interoperability protocols becomes possible. Interoperability protocols typically reflect the applications communication model. However, since the microbroker maps the application communication model to an exchange of invocation objects, different protocols can be used as long as they accept an invocation object and transfer it. Using the same interoperability protocol for outgoing and incoming invocations is not necessary, since the microbroker keeps track of expected responses (modelled as incovations as well). A scenario where a node uses two communication technologies for the outgoing request and the incoming reply is depicted in Figure 7.1.
BASE allows the integration of transport plugins during runtime. The dynamic invocation creation along with local service registries provide a simple reflection mechanism.
The BASE prototype has been implemented in Java, making it suitable for a variety of Java-enabled embedded systems, e.g., mobile phones or the TINIBoard. A minimal configuration of BASE requires 130 KBytes of memory. Due to buffer usage this can increase to a maximum of about 400 KBytes. Still, this makes BASE suitable for many small embedded Java-

based systems. The extensibility of the microbroker allows the integration of features available on resource-rich computing environments.

7.5. PCOM

The functionality provided by BASE offers a basic abstraction to ease application development. Still, additional mechanisms on top of BASE are needed to enable the automatic adaptation of applications during runtime in order to react to the changing availability of services or device capabilities according to the current application execution policy, e.g., to minimize the energy usage or to maximize the dependability of an application.

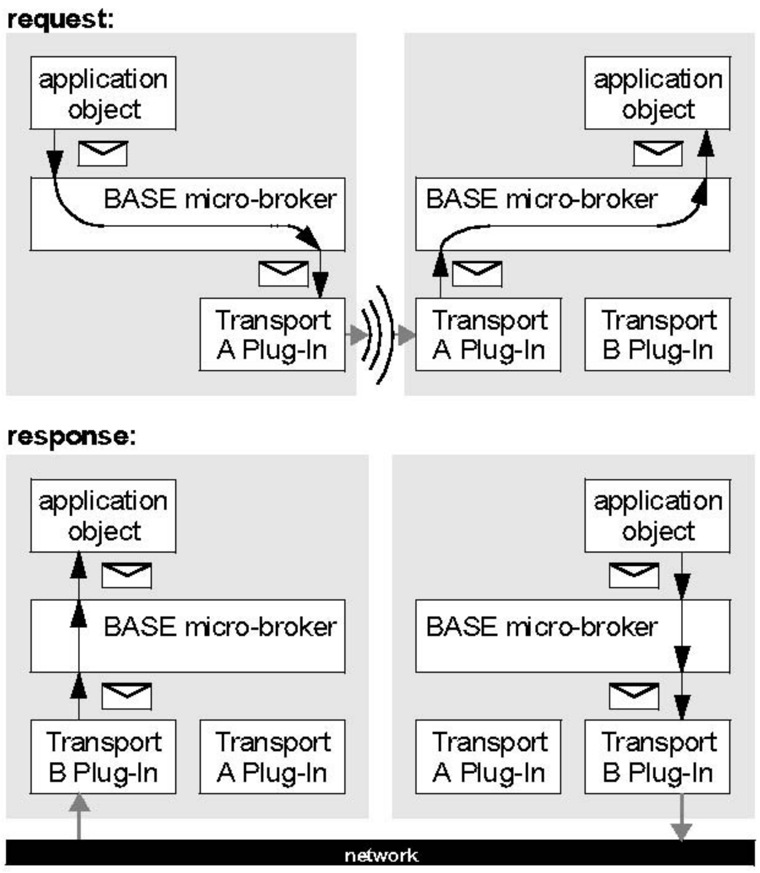


Figure 7.1. Request/response interaction in BASE

To achieve this, we propose an application model based on a component system (named PCOM). The application model specifies the architectural building blocks (modeled by components) and their interdependencies (modeled by contracts between components). At runtime, this specification is mapped to a concrete set of component instances where all mandatory contracts are fulfilled. Hence, PCOMcomponents offer a distinct functionality via contractually specified interfaces (following the definition of [Szy98]). The functional properties of the contract are modelled in the interface itself whereas additional properties, e.g., dependency on another component, QoS requirements, or behavioral contracts via preand postconditions, are explicitly modelled as contract types. This concept has been proposed in the realm of traditional component systems, e.g., [BJP+99, WBG+01]. Contract types are templates for contract instances as well as the components are templates for component instances. When components are instantiated, contract types are mapped to concrete contracts which either offer the desired property, e.g., negotiate a distinct QoS property or bind to another component, or indicate a contract violation. An application is modelled via a special component (the so-called application anchor) which specifies the set of necessary subcomponents. These components depend on each other according to the specified contract types.

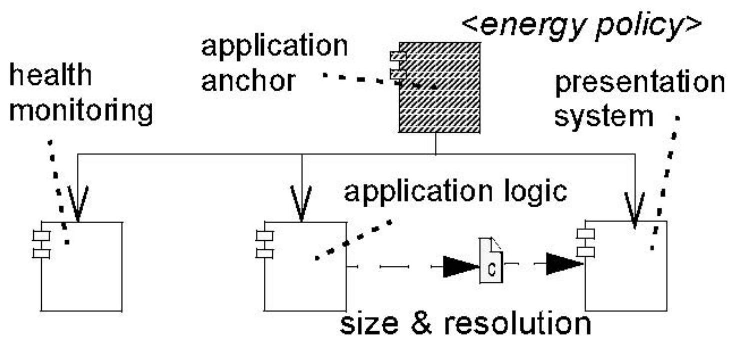


Figure 6.2. Health monitoring application in PCOM.

A simplified example for this is given in Figure 6.2. Here, a health monitoring application is shown, which outputs information and advices whenever a suitable display is in the vicinity. This application is formed by its application anchor and three subcomponent instances:

The *health monitoring* component is used to retrieve sensor information such as blood pressure, pulse, etc.

The *presentation system* component is responsible for presenting advices for certain health conditions, e.g., to calm down, take a distinct kind of medicine, on a display nearby.

The *application logic* component depends on these two components. It receives sensor information from the health monitoring component and derives advices, which it sends to the presentation system component. For simplicity, only the dependency between the application logic and the presentation component is shown. It is modelled as a contract which requires distinct size and resolution of the presentation system.

Additionally, a policy regarding energy consumption is shown, which is assigned to the application.

The application specification has to be mapped to instances on devices which realize the components. The different components are mapped to specific services residing on potentially different devices. Contracts between components have to be negotiated when a binding is established. For an example, before using or acquiring the display component a negotiation ensures that the resolution and size fullfil the contract.

The policy specifying the energy consumption is taken into account by the underlying framework when tasks that need a lot of energy, e.g., performing calculations or accessing remote components, are executed. The policies lead to configurations of the underlying BASE which will enforce them, e.g., in selecting the transport requiring the least energy.

The mapping of application policies, the contracts, and the binding of components deployed across different devices shall be provided by the framework. Currently, we have implemented BASE and designed the above sketched application model. Our next steps involve the design of the underlying framework as well as the mapping of contracts and policies to the services and mechanisms provided by BASE.

The overall framework will allow adaptation of applications by activating those applications where the application anchor contract is satisfied. That is, all dependencies of the applications can be fullfilled according to the application policies and contracts involved. Adaptation is supported by the mechanisms of the underlying middleware and the selection of alternative contracts. The execution context of an application is determined by the services available on nearby devices and the associated component instances from the application specification.

7.6. Related Work

7.6.1. Middleware Systems

In the past, a multitude of different middleware systems has been developed (e.g., [OMG02a, SunRMI]) shielding application programmers not only from distribution of services but also different operating systems or hardware architectures. Conventional middleware systems are designed for mostly stable environments, in which service unavailability can be treated as an error, making these systems unsuitable for spontaneous networking environments.

The latter can be achieved by extending conventional middleware systems to dynamically reconfigurable middleware systems (e.g., [BG00, BCR+00, RKC01]), which are able to adapt their behavior at runtime, e.g., how marshalling is done. Still, most existing reconfigurable middleware systems concentrate on powerful reconfiguration interfaces and not on supporting small, resource-poor devices.

The resource restrictions of such devices prohibit the application of a full-fledged middleware system. One way to address this is to restrict existing systems and provide only a functional subset (e.g., [OMG02b, RSC+99]) leading to different programming models or a subset of available interoperability protocols. Another option is to structure the middleware in multiple components, such that unnecessary functionality can be excluded from the middleware dynamically. One example is the Universally Interoperable Core (UIC) [RKC01]. Like BASE, UIC is based on a microkernel that can be dynamically extended to interact with different existing middleware solutions. However, different communication models or different protocols for outgoing and incoming messages are not supported.

7.6.2. Component Systems and Pervasive Computing

Component systems strive for independence of software components from underlying platform properties in order to allow their reuse. One way to achieve this is to model explicit context dependencies, e.g., via contracts between components or contracts between the component container, such as in J2EE [SunJ2EE]. Typically, the inter-component contracts can be negotiated and various solutions exist, to ease the integration into the application framework, such as the aspect-oriented programming paradigm [BG00, BA01]. While such approaches can be appropriately used to handle the inter-component contracts the component container contract typically relies on a fixed common abstraction, making it unfeasible for pervasive computing environments where the container contract can change.

In the realm of ubiquitous computing the first approaches for component based systems are emerging. While Pebbles [Oxygen] is at a stage where it is hard to judge which requirements will be met, the Aura project [CGS+02] proposes a component framework similar to ours. The resource dependency of the Aura system is not addressed by the underlying middleware but by hand tailored resource monitors. Hence, only a comprehensive support of adaptation at the application layer, not on the middleware layer, is intended. Similar to Aura, One.world [GAB+00] and the Gaia system [RC00] shift the complexity of applicication adaptation to the programmer. Support of the underlying middleware is only provided with respect to communication issues.

7.7. Conclusion and Outlook

Pervasive computing environments differ from existing ones in the increasing heterogeneity of devices and networks. The spontaneous networking leads to situations, which are treated as errors in classical computing, but require distinct precautions since they can happen regularly. Based on typical scenarios we have derived a system model for pervasive computing and the support from system software and application adaptation. We have presented an extensible middleware platform which already provides basic abstractions to ease application development. The automatic adaptation of applications should be supported by a component model based on a framework. The basic abstractions of our middleware BASE can be used to build a framework for a component model. A contract concept is not only used to specify required properties for component interaction but also to indicate application configurations leading to a component-based application model. Adaptation of application is reduced to validating required contracts and activating applications where all contracts are fullfiled. Contract enforcement and mechanisms to adapt are provided by BASE.

Currently, we have designed and implemented BASE. We are building prototypes for applications using BASE in order to gain experience on how the framework can support our application model. In the next steps of our work we will aim at completing the framework.

8. BASE - A Micro-broker-based Middleware For Pervasive Computing

Pervasive computing environments add a multitude of additional devices to our current computing landscapes. Specialized embedded systems provide sensor information about the real world or offer a distinct functionality, e.g., presentation on a "smart wall". Spontaneous networking leads to constantly changing availability of services. This requires middleware support to ease application development. Additionally, we argue that an extensible middleware platform covering small embedded systems to full-fledged desktop computers is needed. Such a middleware should provide easy-to-use abstractions to access remote services and device-specific capabilities. We present a micro-broker-based approach which meets these requirements by allowing uniform access to device capabilities and services through proxies and the integration of different interoperability protocols. A minimum configuration of the middleware can be executed on embedded systems. Resource-rich execution environments are supported by the extensibility of the middleware.

8.1. Introduction

Existing middleware platforms are characterized by their precautions to overcome heterogeneity of computer systems with respect to the hardware platforms and programming languages. However, the computer systems on which applications are executed are mostly homogeneous according to their processing and storage capabilities. The vision of ubiquitous or pervasive computing [Wei91] creates a world populated not only by computers as we know them today but also with sensors and smart "everyday items". The heterogeneity added by these smart things is characterized by an additional property: the embedded systems integrated in the environment are typically tailored to distinct purposes. Hence, not only processing and storage capabilities differ widely but local device capabilities, such as different sensor types for temperature, pressure or positioning, are also device-specific. Communication between the different end-systems can take place over different kinds of network interfaces, such as infrared communication or radio links, e.g., Bluetooth or IEEE 802.11, and additionally via different interoperability protocols, such as IIOP, RMI, or simple event-based protocols.

The availability of resources, remote ones as well as local ones, can change over time, due to network connectivity as well as sensor-specific properties, e.g., it is unlikely that a GPS-based positioning system will work indoors.

In order to provide application programmers with support for conquering the additional complexity in pervasive computing environments, we have developed a micro-broker-based middleware. Our middleware will serve as a foundation for applications as well as component systems, hence the name BASE. Key features of BASE are the uniform access to remote services and device-specific capabilities, the decoupling of the application communication model and the underlying interoperability protocols, and its dynamic extensibility supporting the range of devices from sensors to full-fledged computers.

The chapter is structured as follows. Next, we will motivate the requirements for such a middleware and introduce an example scenario. Existing approaches are classified and discussed in the related work section before we will sketch the overall design rationale of our approach BASE. Some implementation details of BASE and an evaluation will be presented before we close the chapter with a conclusion and outlook on future work.

8.2. Requirements

In order to clarify our system model and derive our requirements, we want to sketch a small scenario. In a future “pervasive computing world”, a building, e.g., an office, contains a huge number of highly specialized and therefore very heterogeneous computing devices. While some of them are stationary, e.g., placed in a room, others are carried by users, e.g., as wearable computers. Devices range from small embedded sensors to classic stand-alone computers. Clearly, the resources and capabilities of such devices differ widely, due to cost and size restrictions. Note, that the capabilities of a mobile device can also change dynamically. As an example, a GPS-sensor will stop functioning when entering a building. To summarize, a pervasive computing environment consists of a multitude of heterogeneous devices, both stationary and mobile, with different and dynamically changing capabilities and specific ways to access them.

One essential device capability is the ability to communicate and interact with other devices. This is achieved by forming spontaneous networks with changing members due to the communication range. Following [KF02] we prefer to use the term ’spontaneous’ instead of ’ad-hoc’ as ad-hoc tends to be restricted to specific lower level functionality like routing. The network interfaces used are highly heterogeneous ranging from infrared communication over radio links to wired connections. Interoperability protocols are tailored to specific requirements as well, e.g., a sensor does not need to implement a complex interoperability protocol but can simply emit its data periodically as events. To summarize, devices interact by forming spontaneous networks using different network interfaces and interoperability protocols. Membership in these networks is temporary and network related properties like communication cost and bandwidth change dynamically.

Distributed applications in this scenario are structured into application objects, or services, interacting with each other. Services in turn use device capabilities or further services, which are provided by either the local device, or by remote interaction with other devices. From the application's point of view, one of the main challenges is to use services and capabilities with changing availability. As we have seen, this is true for local, e.g., GPS, as well as remote cases, e.g., due to reachability. In addition, even a service that is both functional and reachable can become unavailable. Take for example a presentation system integrated into a video projector. If the user leaves the room, the presentation system becomes unavailable, because the user cannot see its output anymore.

Existing middleware platforms typically address portability of applications via standardized interfaces for remote service interaction, e.g., via stub and skeleton objects, and interoperability of applications across different middleware platforms via interoperability protocols. We derive three additional requirements:

1. **Uniform programming interface:** while classical middleware addresses uniform access to remote services the additional heterogeneity of specialized device capabilities requires similar abstractions, e.g., proxy objects, in order to access different device capabilites in a uniform way independent of the underlying platform.

2. **Flexible protocol support:** the service model of a middleware, e.g., remote procedure call or events, is typically reflected in its underlying interoperability protocol, e.g., using request/response messages or emitting event messages. The devices and systems in the above-mentioned scenario would need the integration of a variety of such service models which are reflected by their correspondent interoperability models. A

decoupling of the service model from the interoperability model used by the middleware can help to bridge these interoperability domains. Additionally, this allows different communication paths for the incoming and outgoing messages. As an example think about two devices communicating via infrared in order to save energy. If the infrared link breaks due to obstacles or distance and a wireless radio link still exists, communication can continue. This can be either achieved by providing one interoperability protocol over different network interfaces or by the abstraction of different interoperability protocols which allows flexible usage of existing technologies.

3. **Tailorable:** To be useable on all kinds of devices found in future scenarios, the middleware has to be tailorable to the device at hand, a sensor device as well as a mainframe. The core functionality should be small enough to be executed on a sensor platform, but easily extensible to use the capabilities of resource richer devices.

Nowadays middleware platforms already provide high abstractions for programming distributed systems. Some platforms are already targeted to the above mentioned scenarios. The next section will discuss related work before we will present our approach.

8.3. Related Work

8.3.1. Conventional Middleware Systems

Device heterogeneity is not a unique characteristic of pervasive computing, but can be found in conventional systems, too. Different middleware systems like CORBA [OMG02b], Java RMI [SunRMI] or DCOM [EE98] have been developed to provide a homogeneous access to remote entities independent of e.g., operating systems or hardware architectures. Typically, these middleware systems try to provide as much functionality as possible, which leads to very complex and resource consuming systems, that are not suitable for small devices. Approaches to solve this problem exist and are discussed below. Conventional middleware systems are designed for mostly stable network environments, in which service unavailability is a rare event and can be treated as an error.

8.3.2. Dynamically Reconfigurable Middleware

Extending conventional middleware systems to dynamically reconfigurable middleware systems (e.g., [BG00], [BCA+01], [BCR+00], [Led99], [RKC99], [RKC01]) enables such middleware to adapt its behavior at runtime to different environments and application requirements, e.g., how marshalling is done. Still, different communication models or different protocols for outgoing and incoming messages are typically not supported. As one exception, the Rover toolkit [JTK97] provides this functionality for its queued RPC (QRPC) concept, layered on top of different transport protocols. However, Rover only supports the QRPC and addresses potentially disconnected access to an infrastructure and not spontaneous networking.

A further difference from BASE is that most existing reconfigurable middleware systems concentrate on powerful reconfiguration interfaces and not on supporting small, resource-poor devices. A notable exception to this is UIC [RKC01], which is discussed below.

8.3.3. Middleware for Resource-Poor Devices

The resource restrictions on mobile devices prohibit the application of a full-fledged middleware system. One way to address this is to restrict existing systems and provide only a functional subset (e.g., [OMG02a], [RSC+99], [TAO]) leading to different programming models or a subset of available interoperability protocols. Another option is to structure the middleware in multiple components, such that unnecessary functionality can be excluded from the middleware dynamically. One example is the Universally Interoperable Core (UIC) [RKC01]. UIC is based on a micro-kernel that can be dynamically extended to interact with different existing middleware solutions. Still, the used protocol stack is determined before the start of the interaction and cannot be switched between request and reply as in BASE and abstractions are only provided for remote services.

8.3.4. Middleware for Pervasive Computing

Most pervasive computing middleware systems (e.g., [ACH+01], [CPW+99], [Moz98], [RC00]) try to establish some kind of integrated, preinstalled technical infrastructure in a physical area, e.g., a room or building, often called an intelligent environment (IE), in which the user and his/her mobile devices are integrated on-the-fly when entering the area. The IE offers a huge variety of different capabilities and middleware services that can be used, once the device of the user is integrated.

As an example, the goal of the Gaia system [RC00] is to enhance physical spaces with computers to ActiveSpaces. Gaia provides an infrastructure to spontaneously connect devices offering or using services registered in Gaia. To integrate existing systems, like CORBA, interaction between application objects is done via the Unified Object Bus [RC01], which is layered on top of these systems. As essential system services, such as discovery and lookup, are provided by the Gaia infrastructure, mobile devices cannot cooperate autonomously without the infrastructure.

In contrast to this, we aim at supporting the cooperation of nearby devices, i.e., using only temporarily available hardware and software capabilities of nearby devices, independent of the presence of an external infrastructure. An infrastructure, such as an IE, may be included into a spontaneous network as temporarily available services, but the other way round - without the infrastructure - spontaneous networking requires additional support.

8.4. BASE

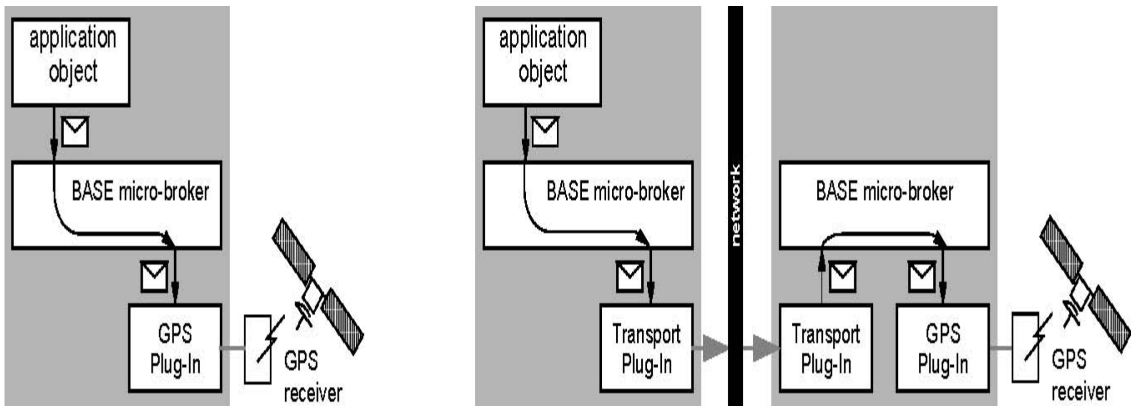
Before we describe the architecture and implementation of BASE, we first want to motivate our design rationale.

8.4.1. Design Rationale

One key idea behind BASE is the uniform abstraction of services as well as device capabilities via proxies as the application programming interface. Consequently, the middleware delivers requests to either device services in the middleware or transport protocols. Allowing different communication models with respect to the transactional pattern (request/response, event, synchronous, asynchronous, etc.) results in the middleware to provide the synchronization independent of the underlying protocols. Our approach is inspired by micro-kernels as they were introduced into the realm of operating systems (e.g., [RJO+89], [TKR+91]) and had some first applications in the middleware area as well (e.g., [PR00], [RKC01]). Only minimal functionality, i.e., accepting and dispatching requests (so-called invocations), is located in the micro-broker. Interoperability protocols as well as object lifecycle management can be added as additional services, realized as plug-ins.

The micro-broker accepts requests represented as so-called invocation objects. In the following, we will refer to the invocation object when talking about an invocation. An invocation is composed of a source and a target address, an operation with parameters, and additional information concerning the handling of the invocation. The micro-broker dispatches the invocation to either a local service, a local device capability or a transport plug-in, which transports the invocation to a remote micro-broker. Transports which receive an invocation or a reply to a previous invocation – also represented by an invocation – submit them to the micro-broker to initiate the dispatching to the corresponding local service or device capability. Invocations can be either generated by proxies, representing a service or a device capability, or manually by the application programmer, e.g., like the request object in the dynamic invocation interface in CORBA [OMG02b]. Figure 8.1 depicts the micro-broker in a typical setting, where invocations are dispatched to (a) device capabilities and (b) transport plug-ins for the remote processing on other nodes. Remote service interaction follows the same pattern and is depicted in Figure 8.4.

Let us briefly argue why we have chosen this approach. Clearly, the requirement for uniform access of device capabilities as well as remote services can be easily established by our approach.



(a) accessing a local device capability (b) accessing a remote device capability
Figure 8.1: Local and remote capability usage.

The micro-broker allows the flexible integration of new transport plug-ins and device capabilities by simply registering a new entity which accepts an invocation. This allows to provide access to all features available on resource-rich computer systems. The minimal functionality of the micro-broker itself allows the deployment of the middleware on resource-poor devices as well. To sum up, the uniform programming abstraction is provided by the service abstraction for remote service access and device capabilities. Together with the extensibility of the micro-broker this fullfils the first and third requirement that we have identified. The micro-broker allows in- and out-going messages over different transport protocols that can be dynamically loaded and configured through the invocation abstraction, which satisfies the second requirement. Although our implementation does not rely on reflection, the dynamic composable invocations along with the service registries provide means for reflection about services registered with the middleware.

The prototype of BASE is implemented in Java but relies only on features available in the Java Microedition. This allows the deployment on small Java-based embedded systems (e.g., [Loo01]) or specialized Java processors (e.g., [JStamp]). The proliferation of end-systems besides classical computers capable of executing Java, such as cell-phones or PDAs, and the aforementioned embedded systems make Java a suitable starting point providing a uniform abstraction for our middleware.

The benefit of our micro-broker approach compared to existing middleware platforms is the minimal footprint needed for a basic configuration which qualifies it for small embedded systems as well as the extensibility providing the means to use features of more sophisticated computers. The configurability that reflective middleware typically provides is also supported by BASE. A major difference to existing middleware platforms is the support of different communication models, such as RPC or events with different synchronization semantics, by the micro-broker, which allows these communication models over a variety of different interoperability protocols. Typically, the main communication model of a middleware is reflected in its interoperability protocols, e.g., CORBA's IIOP reflects the RPC by request/response messages. The BASE micro-broker only requires a transport plug-in to marshal and send an invocation. If responses are expected they may be received by any other transport plug-in.

8.4.2. BASE Architecture

Figure 8.2 depicts the overall architecture of BASE. Four layers are involved. The micro-broker is the central part of the system, consisting of the invocation broker and two registries for local services and devices which can currently be reached.

The micro-broker accepts invocations which are either manually assembled or generated by a stub-call. Additionally, an invocation can be used to access the registries for service lookups.

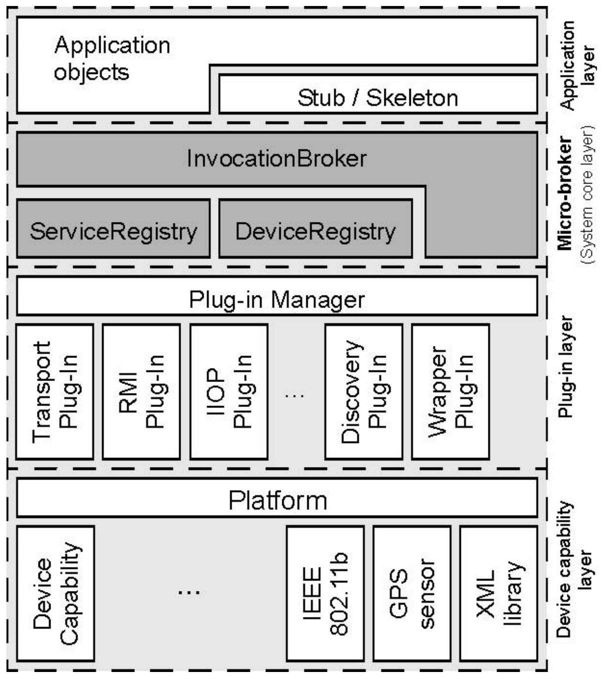


Figure 8.2: BASE architecture.

The plug-in layer maintains plug-ins which represent the entities capable of receiving invocations. Examples for plug-ins are transport protocols or encapsulations of device capabilities, such as sensor systems like positioning or temperature, or other services depending on the device, like input/output capabilities such as printing or video projection. Plug-ins typically involve interaction with the underlying operating system or directly with the hardware to offer access to a device capability or transport. The invocation broker accesses the plug-ins via invocations. Thus the underlying platform is encapsulated by the plug-ins. The device capability layer represents the device platform by its supported hardware and software.

In the remainder of this section the layers sketched above are discussed in more detail starting with invocations, the invocation broker, registries, stubs and skeletons, and the plug-in layer.

8.4.2.1. Invocation

Invocations are similar to dynamic invocation interface requests in CORBA. Figure 8.3 shows the elements of an invocation. Naturally, an invocation is represented as an object. Device and service IDs are used to denote a sender and receiver of an invocation. Services are given unique IDs that are local to a device. This ID is combined with a unique device ID to form a globally unique ID. The message IDs are needed for synchronization issues and are described in the paragraph discussing the invocation broker. A service context field allows the specification of additional parameters that indicate properties relevant to the processing of the invocation in the middleware such as synchronization issues or Quality of Service parameters. Basically, the context is a name-value list where parameters can be added freely. The payload contains the operations and parameters. In the case of event-based communication no receiver needs to be specified and the operation denotes the event-type on which applications can subscribe. The parameters then carry additional information of the event. In point-to-point communication the operations and parameters are interpreted as a remote method invocation.

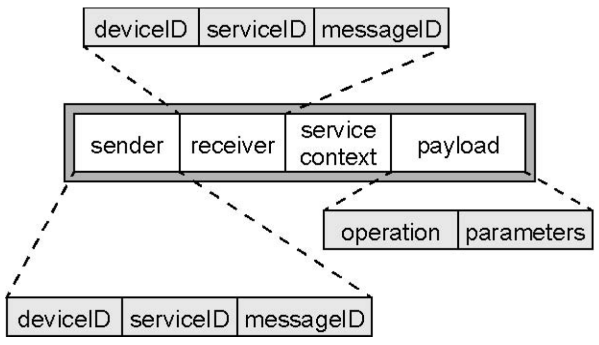


Figure 8.3: Invocation object structure.

8.4.2.2. Invocation Broker

Central to the system core, the invocation broker realizes the core functionality of the micro-broker. Invocations are accepted and dispatched. In order to separate the control flow between application and the processing of an invocation in a plug-in, a thread pool is maintained. Incoming calls are entered into the invocation table, assigned a message ID in order to identify parallel invocations of the same client. The context field contains, among other information, the communication model, i.e., synchronity and transactional pattern (request-response/event) of the invocation. Depending on the communication model, the invocation broker blocks the incoming thread in case of a synchronous invocation. A new thread from the thread-pool is taken and the delivery of the invocation to the responsible plug-in (see below) is executed. After the plug-in has processed the invocation by either a local action, e.g., retrieving a sensor data, or a remote action, i.e., marshalling and sending the request to a remote peer, the thread returns and is added to the threadpool again. In case of a remote processing, an invocation may be sent back to the initial caller. The invocation broker receives the invocation from a plug-in for remote interaction, which may be different from the one that has processed the outgoing invocation, as shown in Figure 8.4.

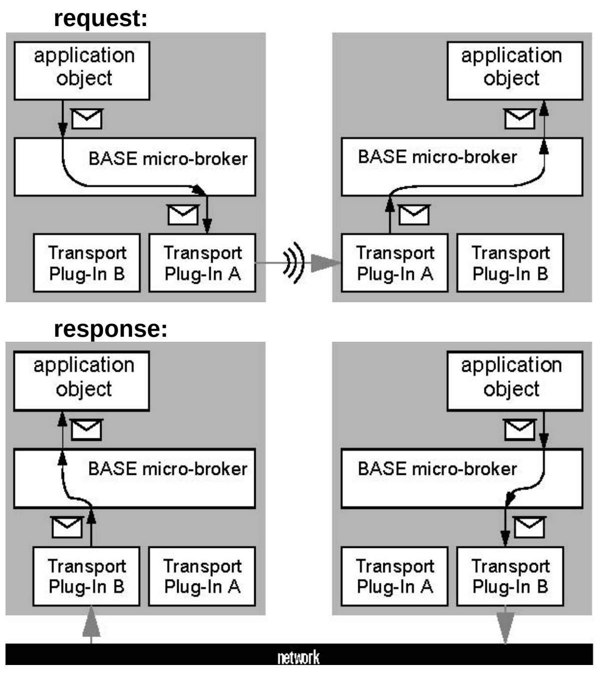


Figure 8.4: Request / response in BASE.

The invocation carries the target object and its message ID. If a message ID is contained in the receiver field of the invocation, this indicates that a caller is either blocked or awaiting an asynchronous delivery of the invocation. In case of a blocked call the waiting thread is freed and the invocation is provided as return. In the asynchronous case the invocation broker takes a thread from the thread-pool and calls up the application through a callback. In this case the message ID is used to designate the application callback registered at the invocation broker.

Notice that the explicit handling of synchronization depending on the communication model retrieved from the service context is a major design decision in BASE. This decouples the communication model from the underlying interoperability protocols. A request/response based communication model can be realized over two event-protocols as well as an event can be sent as a single request in an RPC-based interoperability protocol. An interaction can take place over different transport plug-ins for out-going and incoming invocations.

So far, BASE only supports a limited number of communication models, but an extension to different synchronization models, see e.g., [OMG98], can easily be established with the underlying concept.
In order to determine the target of an invocation or to provide applications with service lookup two registries are maintained and described below.

8.4.2.3. Service and Device Registry

The service registry maintains all locally available services on a device. Services - as mentioned before -can be either application objects offering a service or device capabilities. Applications can query for available services by either specifying a name or the functional properties, i.e., the interface. Hence, a simple name and trading service is provided. Due to the nature of spontaneous networks, the availability of a lookup service cannot be assumed. The device registry maintains a list of all currently reachable devices and the transport plug-ins which provide the access to another device. If multiple transport plug-ins are possible for the same device, they are also entered into the list. This allows for a simple service lookup in the vicinity of a device. If a service request cannot be fullfilled locally, registries of nearby devices are queried and the result presented to the application.
The information of the device registry is also used by the invocation broker in order to determine which transport plug-in should be used. First, without any further information, any of the available transport plug-ins can be used. As long as there is a connection between two devices, i.e., the device is listed in the device registry and at least one transport plug-in is provided, invocations can be exchanged. Notice, that even if the transport plug-in by which a request invocation has been sent becomes unavailable replies can be received, if another transport plug-in exists. The service context sent with an invocation can be used to control the selection of specific transport plug-ins, e.g., in order to save energy or require a

distinct bandwidth. We plan to extend this concept by strategies which will provide application-specific selection of transport plug-ins according to policies, e.g., energy awareness.

Although the current implementation of the service and device lookup is rather simple, the underlying concept is designed to be extensible allowing the integration of other lookup mechanisms, e.g., Jini [Wal99] and UPnP [Mic00].

8.4.2.4. Stubs and Skeletons

A common abstraction in middleware systems are local proxies for remote entities providing local access for application objects - stubs representing the remote service to clients and skeletons issuing local calls to services. In BASE, stubs and skeletons rely on the invocation abstraction. Stubs generate invocations upon method calls and skeletons generate local method calls upon a received invocation. Notice, that the generation of an invocation does not result in the marshalling of the parameters. This is a responsibility of the transport plug-ins. Invocations are used here to provide a common concept for interaction with the micro-broker. Applications can, however, omit the use of stubs and skeletons and compose and interpret invocations directly.

In contrast to systems like Jini [Wal99], where stub and skeleton can include a service specific protocol stack this is not provided in BASE. Instead a service specific protocol would be realized as a transport plug-in and thus become re-usable for other services as well.

8.4.2.5. Plug-In Manager

The plug-in layer is essential for the abstraction BASE presents to an application developer. Plat-form-specific capabilites, e.g., device capabilities and transports, are represented as plug-ins and become accessable to the application programmer as services. The plug-in manager allows the dynamic loading and integration of new plug-ins.

Device capabilities are registered at the local service registry, and transport protocols at the invocation broker itself.

Plug-ins provide an abstraction of device-specific resources. Depending on the platform interface that allows the access of the device capability layer they can be portable among devices. Thus, an application on top of BASE will only interact via invocations, either dynamically constructed or generated by stubs, with device-specific capabilites.

Transport plug-ins are responsible for accepting an invocation, marshal it, and transmit it as a protocol data unit to a remote peer, which then constructs an invocation by demarshalling it. The simplest transport plug-in would use object serialization to marshal an invocation into a byte-buffer and send the buffer via a transport protocol, e.g., TCP/IP. Other transport plug-ins could rely on existing interoperability protocols and marshal and represent the invocation accordingly, e.g., map it to a request-message in IIOP and marshal the parameter by CDR, which allows interoperability with CORBA-based systems.

As long as the context of an invocation does not require a distinct transport plug-in, the invocation broker may use any transport plug-in to send an invocation to a remote device. The device registry maintains a list of all currently available transport plug-ins to a specific device. Hence, communication can take place as long as at least one transport plug-in allows the communication.

8.5. Implementation Status and Evaluation

This section will present the current status of our prototype implementation and discuss memory size and execution performance measurements.

8.5.1. Implementation Status

Our prototype has been implemented in Java to rely on its platform-independence. Although, for small devices C or C++ would seem to be a better choice at first, we found that Java allows us to run our middleware on a multitude of different devices, if the used Java features, like reflection, etc. are carefully restricted. A Tini minicomputer for example can execute only a subset of Java Version 1.1. Other devices, like smart phones or PDAs are limited to the Java Microedition [SunJ2ME].

So far, our prototype implements the basic concepts. Namely the invocation broker, the service and the device registry are implemented. The invocation broker handles different synchronization concepts and the service context is used to indicate the synchronization of RPC calls. For synchronous invocations, stub and skeleton support is implemented. Two transport plug-ins are realized so far, one based on the Java standard serialization mechanism on top of TCP/IP and a second based on Java RMI. Others are under way. The plug-in manager is implemented and allows the dynamic and static configuration of a BASE system.

8.5.2. Memory Size

The memory footprint of a minimal BASE configuration is crucial in order to allow the installation on small or embedded devices. We have measured the memory footprint of such a configuration, containing the micro-broker (invocation broker and registries) plus a TCP-based transport plug-in. The measurements where done using the IBM J9 implementation of the Java Microedition, more specifically the Java Microedition with the Connected Device Configuration and the Foundation Profile. First, in order to determine the memory footprint without additional dynamic memory consumption, i.e., BASE in idle mode, we use the Windows Task-Manager, as suggested in [Wk00]. In this mode 132 KByte are used. During runtime, when invocations are exchanged, the system uses up to 420 KBytes, which was measured using the J-Sprint profiler [JSprint].

8.5.3. Execution Performance Overhead

To measure the execution performance overhead introduced by the additional communication via the BASE micro-broker, we compared a BASE configuration sending invocations via a Java RMI transport plug-in with a pure Java RMI-based system. The measurements were conducted for a synchronous RPC communication by transmitting invocations for an operation testOperation, that takes a single string input parameter and returns immediately. The string size was either 0 or 1000 characters. This was done for local as well as remote invocations. The results are shown in Figure 8.5 and Figure 8.6. Each value given is the average of 12750 measurements. Measurement was done in 50 rounds with *roundnumber* ×10 invocations per round, leading to a total number of $\sum_{i=1..50} 10 \times i = 12750$; measurements.

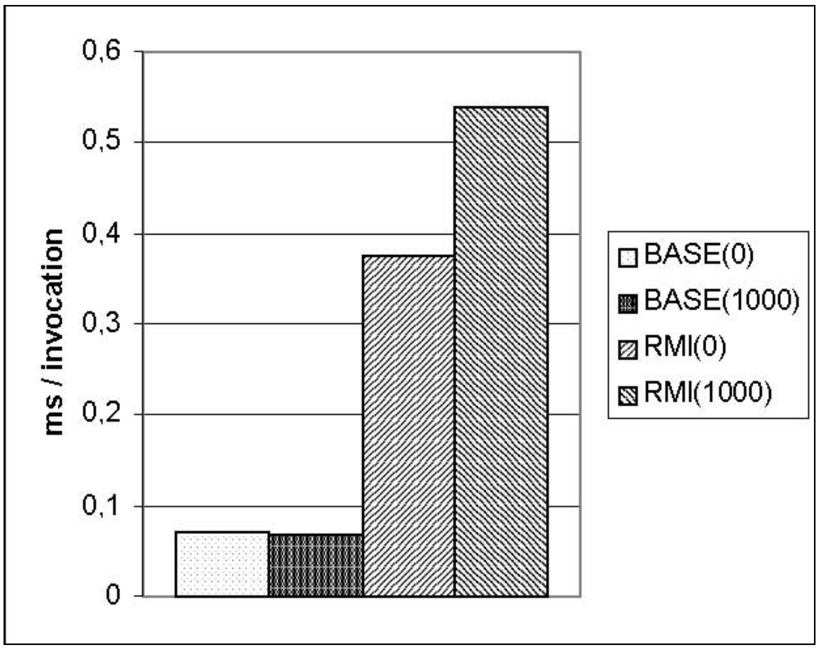


Figure 8.5: Local communication performance.

8.5.3.1. Local Invocations.

In the local case, BASE is clearly faster than RMI. This is due to the fact, that RMI in this case uses the loop-back interface including the RMI and TCP protocol stack while the BASE micro-broker forwards the call directly to the service skeleton and does not use the RMI-based transport plug-in at all.

8.5.3.2. Remote Invocations

In the remote case, BASE introduces an additional performance overhead of about 20%. Taking into account the creation of invocations from the stub objects and their interpretation by the skeletons, this seems acceptable. However, the absolute end-to-end latency measured for BASE is about 4 ms per remote invocation with a string size of 1000, which is rather long. Therefore, we did some additional measurements to compare this to the end-to-end latency of pure RMI, i.e., calling the remote operation directly through RMI without marshalling the invocation object. The pure RMI call only needed about 0,95 ms or 25% of the time BASE needed. This is due to the fact that we have used the standard Java object serialization mechanism in our prototypical RMI plug-in to marshal the invocation object. Note, that this is not a problem of the micro-broker itself, but of the current RMI plug-in implementation. Currently, other transport plug-ins are under development to overcome this performance bottleneck.

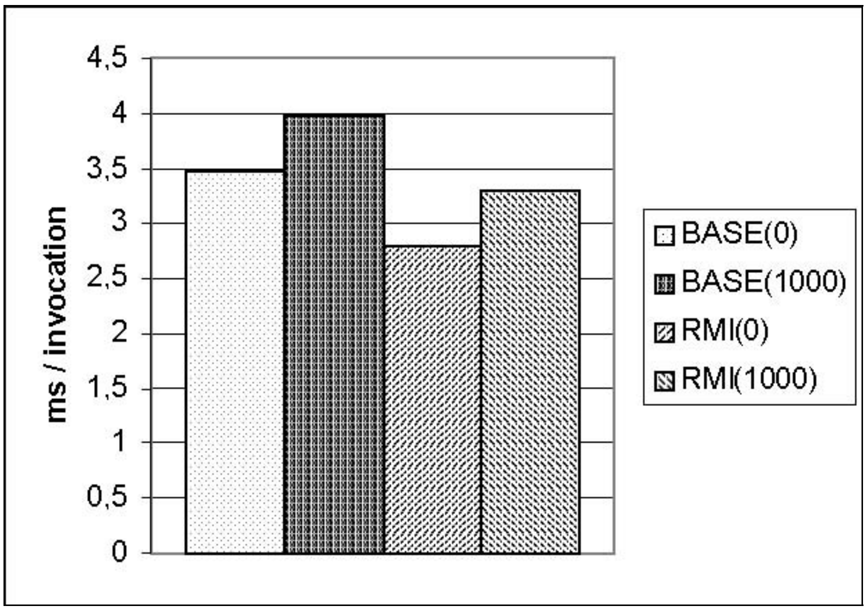


Figure 8.6: Remote communication performance.

8.6. Conclusion and Future Work

We have presented the concept and design of BASE, a flexible middleware supporting the additional requirements of pervasive computing environments. Based on a micro-broker design, BASE allows minimal installations on embedded devices or specialized platforms as well as the integration of features available on resource-rich devices, such as personal computers. Application programmers can rely on a uniform abstraction to access remote and local services as well as device-specific capabilities. Thus BASE supports the portability of applications across heterogeneous devices. The middleware shields applications from the multitude of different communication technologies and interoperability protocols by separating the communication model of the application and the interoperability protocols used. This allows the usage of nearly arbitrary interoperability protocols.

The current implementation status of BASE is promising. Currently we are adding further support for different interoperability protocols and port BASE to some specialized devices. Further experience will be gained from doing prototypical implementations of pervasive computing applications in our lab.

Using BASE as a middleware already will ease the design and implementation of applications. In further research directions we want to design a component system based on BASE that will support the adaptation of applications due to their execution environment. BASE will be extended by mechanisms to enforce adaptation strategies in the component framework, such as migration or service selection strategies. The extensibility of the micro-broker approach seems to be a good BASE here.

9. PCOM – A Component System for Pervasive Computing

Applications in the Pervasive Computing domain are challenged by the dynamism in which their execution environment changes, e.g., due to user mobility. As a result, applications have to adapt to changes regarding their required resources. In this chapter we present PCOM, a component system for Pervasive Computing. PCOM offers application programmers a high-level programming abstraction which captures the dependencies between components using contracts. The resulting application architecture is a tree formed by components and their dependencies. PCOM supports automatic adaptation in cases where the execution environment changes to the better or to the worse. User supplied as well as system provided strategies take users out of the control loop while offering flexible adaptation control.

9.1. Introduction

Pervasive Computing is characterized by the interaction of a multitude of highly heterogeneous devices, ranging from powerful general-purpose servers located in the infrastructure, to tiny mobile sensors, integrated in everyday objects. Devices are connected to each other on-the-fly using wireless communication technologies like Bluetooth, IEEE 802.11 or IrDA and share their functionality. A sensor could for instance use a nearby display to present its data to the user.

Developing and executing applications in such environments is a non-trivial task. Apart from the device heterogeneity, the hardware and software resources, i.e., devices and services, available to an application are highly dynamic, due to factors like user mobility, fluctuating network connectivity or changing physical context. This forces applications to adapt themselves constantly to their ever-changing execution environments. User-interaction, e.g., for adaptation control or administrative tasks, should be minimized, thus removing the user from the control loop [WPT03].

To ease application adaptation, we have developed BASE, a flexible middleware for Pervasive Computing environments (see e.g., [BSG+03] for details). It provides adaptation support on the communication level by dynamically (re-) selecting communication protocol stacks, even for currently running interactions.

BASE offers no support for adaptation at higher levels, e.g., by automatically reselecting services and devices. Therefore, we have designed and developed PCOM, a light-weight component system on top of BASE. PCOM allows the specification of distributed applications that are made up of components with explicit dependencies modeled using contracts. An application can be executed if all of its components can be executed – either local or remote – meaning that all dependencies between components can be fulfilled. In order to automatically choose alternatives if multiple suitable components are available, strategies are employed. This allows adaptation without prompting the user. The main contribution of this chapter is the definition and evaluation of this light-weight component system for strategy-based adaptation in spontaneously networked Pervasive Computing environments.

The remainder of the chapter is structured as follows. Next, we will present our system model and briefly sketch BASE. Models for application adaptation are discussed in section 9.3. The requirements on application adaptation, especially those that are not fulfilled by BASE, are derived in section 9.4. Section 9.5 presents the architecture of PCOM, its application model and the mechanisms that enable adaptation. As an indication for the validity of our approach, an evaluation of PCOM, including a comparison of application adaptation in BASE and PCOM is given in section 9.6. After discussing related work in section 9.7, we conclude the chapter and provide an outlook on future work in section 9.8.

9.2. System Model

Our work focuses on spontaneously networked Pervasive Computing environments in which devices are connected on-the-fly, typically using some kind of wireless technology. Such environments are highly dynamic. Connections between devices are not permanent, the topology of the network is constantly changing, and there is no central or coordinating element. We do not assume the presence of a smart environment like Gaia [RC00], Aura [GSS+02] or iROS [JFW02]. Although such an infrastructure could be available at certain times, devices cannot rely on it.

In our system model communication and thus interaction is restricted to devices that are currently reachable by the network (e.g., due to communication technology). As a result, systems in these environments are inherently location-aware as communication is typically spatially limited. The devices have different specializations and resource limitations. Besides resource-poor and specialized devices such as sensor nodes, resource-poor general purpose devices could be present, e.g., PDAs. Also resource rich-devices can either provide a general purpose platform or they can provide single services such as a presentation system. Due to the lack of a central or coordinating element, applications are dynamically composed of services provided by devices that are part of the currently reachable environment. As an example, consider an instant messaging application that requires an input service such as a keyboard or a touch screen to write messages and an output service to display messages, e.g., a monitor, a video projector or an audio channel. During start up, the application scans the current environment for available services and connects to suitable instances. At execution time, the application uses the services and adapts to changes regarding their availability or quality. Possible adaptations could include for instance the reselection of the output service whenever it becomes unavailable.

9.2.1. BASE.

In order to provide basic support for services that enable such applications, we have developed BASE. BASE is written in Java using the Java 2 Micro Edition with the Connected Limited Device Configuration (CLDC). It assists application programmers by providing mechanisms for device discovery and service registration that can be used to locate and access local as well as remote device capabilities and services. Since the availability of services and capabilities can fluctuate in spontaneously networked environments, BASE provides a simple signaling mechanism to determine their availability. Communication protocols and device capabilities can be extended flexibly, since BASE is structured as an extensible micro-broker. This allows the middleware to run on resource-poor devices and benefit from resource-rich devices. In the context of this work, BASE is used as underlying communication middleware, offering communication and discovery on a wide range of devices. More information on BASE can be found in [BSG+03] and [BS03b].

9.3. Adaptation Models

To provide application adaptation support for Pervasive Computing systems, three main levels of support can be distinguished. This classification is similar to the one given in [OGT+99].

Manual adaptation: here, adaptation is done by the end user. If an adaptation is performed, the system presents different choices and the user selects the most appropriate one. For the instant messenger described previously, this means that the user has to explicitly select the output or input service used by the application, whenever a used service becomes unavailable or a new service is discovered. Clearly, this is time-consuming and irritating, especially for environments with a high level of dynamism and a large number of different devices and services.

Application-specific automatic adaptation: to lessen the involvement of users, application adaptation should be executed with as little user interaction as possible. This can be realized by shifting the adaptation decision into the application. As a result, the system must support adaptation by signaling changes in the environment and the application programmer has to explicitly handle resource availability on a per-resource base, leading to complex and error-prone adaptation routines. Regarding the instant messenger scenario the programmer must provide routines that reselect the input and output service whenever the used services become unavailable. Such a reselection may be necessary at any point during the usage of a service. Therefore, the code of the application will be cross-cut by adaptation routines that are effectively reducing its readability and maintainability.

Generic automatic adaptation: at the highest level of support, application adaptation is done without stressing users or application programmers. The programmer only specifies the functional and non-functional properties of services required by the application and the user controls the adaptation process by stating adaptation goals. Thereafter, the system monitors service availability and selects the optimal services. The programmer of an instant messenger simply specifies the parameters of the input and output service, e.g., minimum screen resolution, and the user defines the adaptation preferences, e.g., highest available resolution. At runtime, the system automatically tries to find services with an acceptable quality. In cases where multiple services fulfill the requirement, the system performs the selection based on the preferences of the user.

9.4. Requirements

BASE offers generic automatic adaptation support at the communication layer. With PCOM we aim at providing further generic adaptation support at the application layer. PCOM should enable application programmers to extend the system with application-specific adaptation logic if needed. This enables a rather straight forward specification of application dependencies along with standard adaptation strategies resulting in a simple core system which can be customized to the needs of an application programmer. From these objectives the following requirements can be derived:

Application specification: applications should be specified in terms of their required services. Services should clearly denote their dependencies to other services and the platform. Non-functional properties of the dependencies should be explicitly stated. The composition of an application from services should allow the specification of alternatives in order to support the system to automate adaptation decisions.

Service monitoring: the system has to monitor the availability of services in order to detect currently used services that change their non-functional properties or become unavailable as well as to detect new services.

Strategy based adaptation: the system has to provide means for automatic adaptation of an application. If alternatives of services are present in the current execution environment, strategies decide which service to select. Besides standard strategies, e.g., to optimize energy consumption, user-defined policies should be integrated. At the core of adaptation, the application lifecycle and the lifecycle of single services have to be managed.

Minimalism and extensibility: to meet the resource heterogeneity of Pervasive Computing the resulting system has to be minimal with respect to required resources, e.g., processing power and memory, and it has to be extensible to exploit the advantages of resource-rich devices.

9.5. PCOM

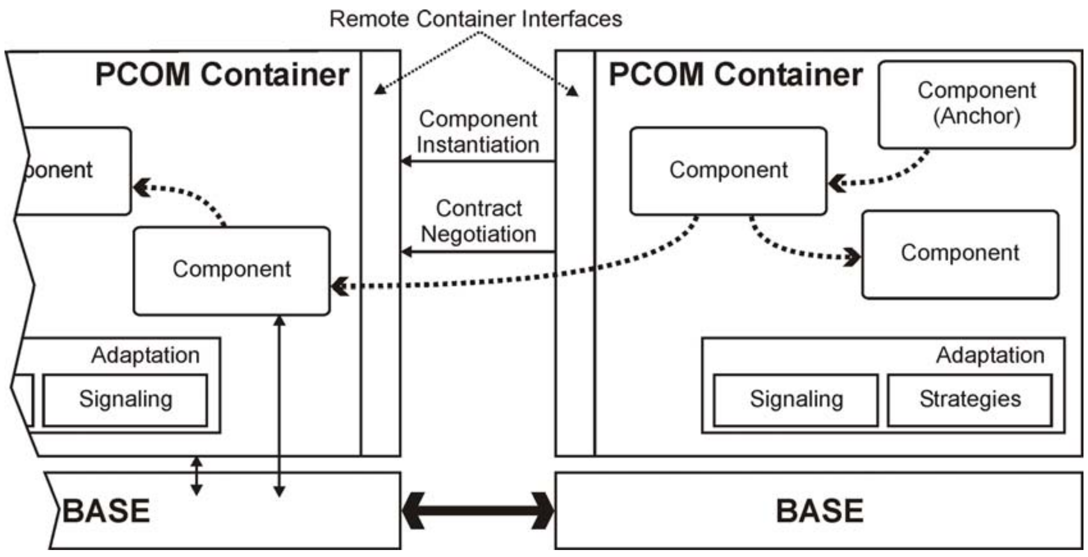


Figure 9.1: PCOM Architecture

In the following we will present our component system PCOM (see Figure 9.1). PCOM provides a distributed application model and supports automatic application adaptation based on signaling mechanisms and adaptation strategies. Applications are composed of interacting entities, so-called components, which dependencies are explicitly specified as contracts. The PCOM container hosts components, manages their dependencies, and thus acts as a distributed execution environment for applications. Each container defines a remote container interface that exports locally available components by their contracts and allows remote containers to negotiate new contracts and access the components. To reuse the communication and discovery capabilities of our middleware BASE, the container is implemented as a single service on top of BASE. As a result, a container is automatically capable of detecting and using other containers.

In the following we will further describe our application model and present components along with their contracts. After that, we discuss application adaptation in PCOM and its realization.

9.5.1. Application Architecture

Applications in PCOM are composed of components, that interact with each other in order to fulfill their dependencies. Components are atomic with respect to their distribution but can rely on local or remote components, resulting in a distributed application architecture.

An application is modeled as a tree of components and their dependencies where the root component (the so-called application anchor) identifies the application. The application tree reflects the dependencies between components where the successors of a component identify its dependencies in order to fulfill the service. PCOM uses a tree as application model, because arbitrary graphs cause several complications. For instance, the multiple use of the same component requires merging probably conflicting requirements. As another example, cycles of the graph could cause infinite loops during the composition of applications.

The life cycle of an application is reflected by the life cycle of its application anchor. Next, we will explain components in more detail, including the modeling of dependencies via contracts and their life cycle.

9.5.2. Components

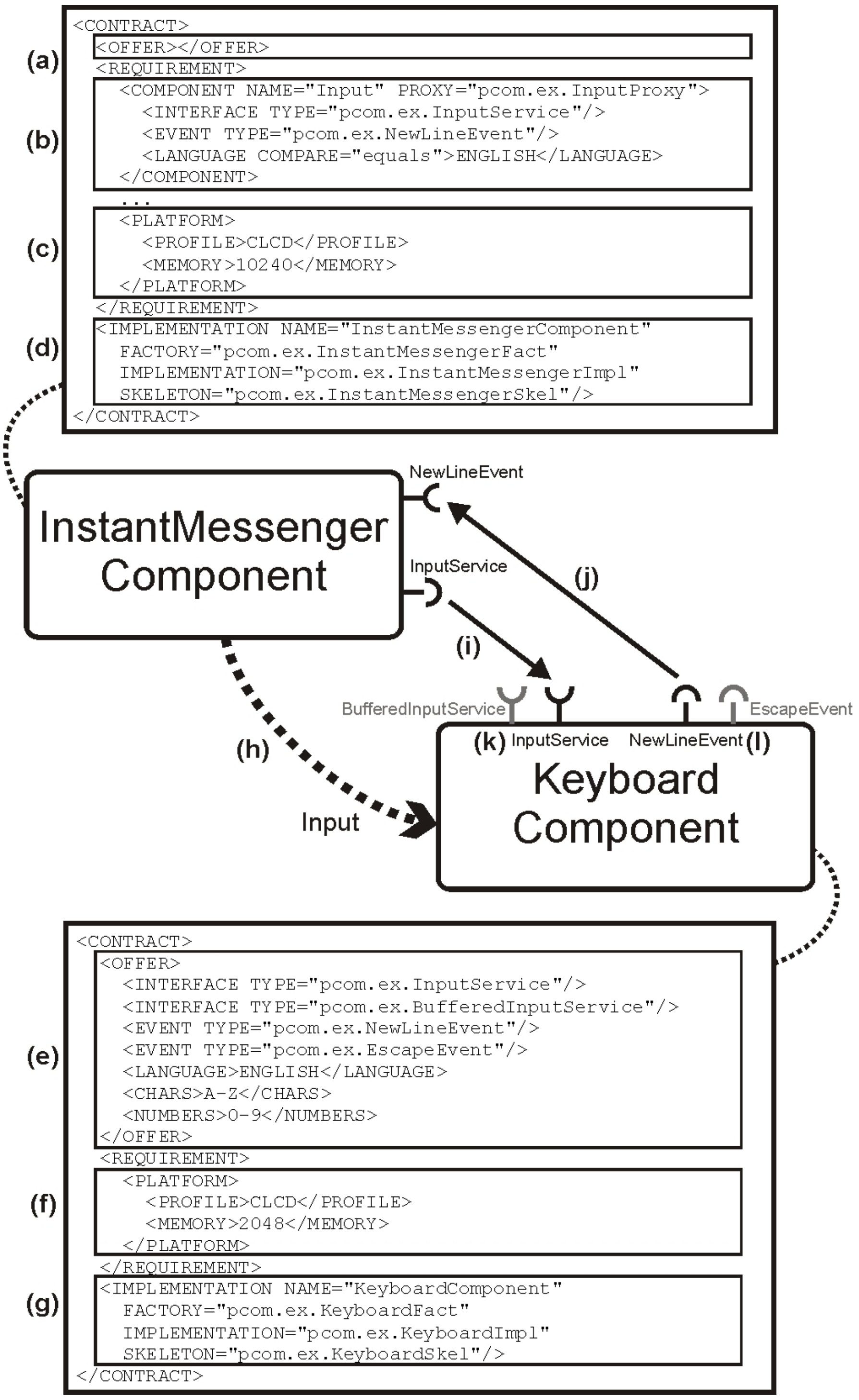
Components in PCOM are units of composition with contractually specified interfaces and explicit context dependencies. PCOM's components enclose contracts that describe their offered functionality and requirements regarding the platform and other components. Components are atomic with respect to distribution and may use other components in order to provide their service. Note that PCOM does not regulate the granularity of components. Therefore, the granularity could range from single functionalities to complete applications.

9.5.2.1. Contracts.

Contracts consist of two distinct parts: The first part specifies the corresponding component's requirements on the executing platform, e.g., required libraries or memory. The second part specifies the functionality provided by the component and its dependencies on other components. A dependency between two components has a direction and reflects the fact that one component either requires certain service interfaces (pull) or listens to some events provided by another component (push). Thus, PCOM supports push and pull communication models between components.

In order to describe dependencies, contracts in PCOM specify the service interfaces and the events that are offered and required by a component. Along the syntactical interface specification of events and services that define a functional dependency, non-functional parameters can be added to express further properties, such as a screen-size, energy consumption or performance related parameters. In contrast to the functional specification that is known at compile time, non-functional parameters can vary at runtime and might depend on the offer of components that are used to satisfy the dependencies. Thus, non-functional parameters can be either static or dynamic.

At runtime, contracts in PCOM are represented as object graphs. To ease the specification of these graphs, we use a compiler to transform an XML document into code that creates the desired structure. This representation is used for the comparison of offers and requirements. By applying them, it is possible to determine whether the offer of one component can be used to satisfy the requirements of another component. Due to the possibly large number of comparison operators that is needed to support arbitrary non-functional parameters, the underlying object model provides only a small set of operators that can be extended by application programmers.



```
<CONTRACT>
  <OFFER></OFFER>
  <REQUIREMENT>
    <COMPONENT NAME="Input" PROXY="pcom.ex.InputProxy">
      <INTERFACE TYPE="pcom.ex.InputService"/>
      <EVENT TYPE="pcom.ex.NewLineEvent"/>
      <LANGUAGE COMPARE="equals">ENGLISH</LANGUAGE>
    </COMPONENT>
    ...
    <PLATFORM>
      <PROFILE>CLCD</PROFILE>
      <MEMORY>10240</MEMORY>
    </PLATFORM>
  </REQUIREMENT>
  <IMPLEMENTATION NAME="InstantMessengerComponent"
    FACTORY="pcom.ex.InstantMessengerFact"
    IMPLEMENTATION="pcom.ex.InstantMessengerImpl"
    SKELETON="pcom.ex.InstantMessengerSkel"/>
</CONTRACT>
```

```
<CONTRACT>
  <OFFER>
    <INTERFACE TYPE="pcom.ex.InputService"/>
    <INTERFACE TYPE="pcom.ex.BufferedInputService"/>
    <EVENT TYPE="pcom.ex.NewLineEvent"/>
    <EVENT TYPE="pcom.ex.EscapeEvent"/>
    <LANGUAGE>ENGLISH</LANGUAGE>
    <CHARS>A-Z</CHARS>
    <NUMBERS>0-9</NUMBERS>
  </OFFER>
  <REQUIREMENT>
    <PLATFORM>
      <PROFILE>CLCD</PROFILE>
      <MEMORY>2048</MEMORY>
    </PLATFORM>
  </REQUIREMENT>
  <IMPLEMENTATION NAME="KeyboardComponent"
    FACTORY="pcom.ex.KeyboardFact"
    IMPLEMENTATION="pcom.ex.KeyboardImpl"
    SKELETON="pcom.ex.KeyboardSkel"/>
</CONTRACT>
```

Figure 9.2: Exemplary Contracts

9.5.2.2. Example.

Figure 9.2 shows XML-based contract specifications for an exemplary instant messenger component and a keyboard component. First, we will have a look at the messenger's contract. It specifies that the messenger component does not offer any service to other components (a) and that it depends on an input component offering a given service interface and event type (b). Additionally, the messenger's contract states the non-functional requirement that the input component's language must be English (b). Next, the platform dependency declares, that the messenger must be executed by a container that has at least 10 Kbytes of free memory and provides a CLDC (c). The last section of the contract contains information about the component's internals used by the container (d).

In contrast to the messenger's contract, the keyboard component's contract specifies an offer that consists of two interfaces and two events (e). Additionally, the offer also contains non-functional attributes that describe the available keys and the supported language. Apart from the requirements on the platform (f) the keyboard does not have any requirements. Again, the last section of the contract contains information about the component implementation (g).

At runtime, these XML-based contracts are transformed into an object model that allows matching the instant messenger component's requirements with the offer of the keyboard component. As the keyboard offers all required functional and non-functional features, it can be used to satisfy the messenger's dependency. After the components have been combined at runtime (h), the instant messenger component is capable of placing calls to the interface provided by the keyboard component (i) and the keyboard component can send the requested event to the instant messenger (j). The additional interface (k) and event (l) of the keyboard component will never be used.

9.5.2.3. Component Lifecycle

To consistently embed components into applications, the container defines and manages the lifecycle of components. Conceptually, this lifecycle consists of the two states STARTED and STOPPED. The state transitions are controlled by the container. The container loads a component by first loading the object graph that represents its contract. It then determines whether it can fulfill the component's requirements towards the platform. If they can be satisfied, the container adds the contract to the set of exported contracts. Initially the component rests in the STOPPED state. Once a component is about to be embedded into an application, the container tries to resolve and initialize the component's dependencies by selecting suitable components to fulfill them. This initial resolution of dependencies can be seen as a special case of adaptation. A more detailed description of the selection process is given in subsection 9.5.3. After all dependencies are fulfilled, the container triggers a transition to the STARTED state. In this state, the component provides its functionality and the container provides signaling and adaptation support. When the state changes to STOPPED, the container releases all resources held by the component.

9.5.2.4. Contract Exchange and Negotiation

As soon as a component is about to be executed, the container has to determine whether its dependencies – both, functional and non-functional – can be satisfied. In order to find components that can potentially be used to satisfy a dependency, the container sends the contract that contains the requirements to the containers available in the environment. These containers reply with the contractual offers of their components that could fulfill the requirements.

As mentioned earlier, there are non-functional parameters that a component cannot determine without knowing the components that are used to satisfy its dependencies. In order to determine such parameters, PCOM containers also support a negotiation phase that recursively determines the non-functional parameters of a component without starting it. To enable this, containers rely on so-called factories that are representatives for locally installed components. Factories provide the capability to determine the actual value of a non-functional parameter based on the set of components that is currently available. While PCOM provides a simple standard factory, application programmers can provide component-specific factories by declaring them in the component contract's implementation section (see Figure 9.2 (d)).

The algorithm for contract negotiation is a post-order traversal of the tree of matching offers and requirements, where factories implement the functionality that determines the values of non-functional parameters from the available offers.

9.5.3. Adaptation

In ever-changing environments, component-based applications have to deal with fluctuating availability and quality of components. Changes regarding the availability and quality of components can either have a positive or a negative impact on the application. This means that the quality of a used component's functionality can either increase or decrease during the execution. Also, used components might become unavailable and new components that could deliver a required functionality might be discovered at any time.

In order to adapt to fluctuations, a component has to have means of detecting changes with respect to quality and availability of other components that either depend on or are required by the component.

PCOM defines three signaling mechanisms that detect changes regarding availability and quality.

9.5.3.1. Signaling Mechanisms

The first signaling mechanism is targeted at the availability of used components. Whenever a used component becomes unavailable, a so-called *communication listener* is notified. Application programmers can register communication listeners for every dependency of a component. As PCOM uses a soft-state lease mechanism to maintain the dependencies between components, the detection of an unavailable component is either a result of an unsuccessful call placed by the using component or by a heart-beat message sent by the runtime system.

The second mechanism detects the availability of new components. In order to receive notifications about components that could potentially be used to replace a currently used component, programmers can define *discovery listeners* for each dependency. Whenever BASE detects a new device, PCOM checks whether the device hosts an instance of PCOM. If a new instance is discovered, PCOM determines whether the new components could be used to replace a dependency of a locally executed component. The comparison of the requirements of a running component and the offer of a newly discovered component is solely based on the static parameters of the offer, significantly reducing the discovery overhead. Once a discovery listener is called, an adaptation strategy can decide, if a full negotiation of the dynamic parameters should be done. Hence, negotiation is performed only if an application may profit from a component change.

The last signaling mechanism provided by PCOM aims at fluctuations in the quality provided by a component. As mentioned above, non-functional parameters can change over time. Therefore, PCOM allows application programmers to specify *contract listeners* that are notified whenever a parameter changes.

9.5.3.2. Options for Adaptation

Application programmers can use the described signaling mechanisms as hooks to specify their own actions for adaptation or use system provided mechanisms. PCOM offers two generic mechanisms: *execution discontinuation* and *component reselection*. Application programmers are provided with means to implement further options, e.g., modifying contracts or retransmitting messages in case of a transient network partitioning.

The first generic adaptation mechanism is simply the discontinuation of an executed component. Whenever an executed component is no longer able to provide its functionality, it can stop its execution. This will result in an event that is received by the communication listener of the using component. With respect to the application model defined by PCOM, this means that a problem in a component is escalated to the next, i.e., higher, level of the tree. The escalation continues until a component resolves the conflict by either reselecting a component (see below) or applying a user-defined strategy. If the escalation leads to the discontinuation of the application anchor, the execution of the application stops.

The second generic mechanism supports the reselection of components at runtime. This is enabled by two features. First, components specify their dependencies explicitly which allows matching a contractually specified requirement and its corresponding offer. Second, PCOM allows the definition of strategies that prioritize possible components based on user preferences. Therefore, if a component initiates the reselection of a certain dependency, PCOM can automatically determine the possible replacements that match the programmer's requirements. If there are several possible replacements, a user defined strategy is applied to select the best replacement according to the user's current selection goals. Clearly, a simple reselection will only be possible if the corresponding component is stateless. For stateful components, the application programmer still has to provide additional routines that establish the desired state. Nevertheless, the programmer does not have to implement the reselection algorithm and can use the signaling mechanisms to add an application-specific adaptation routine.

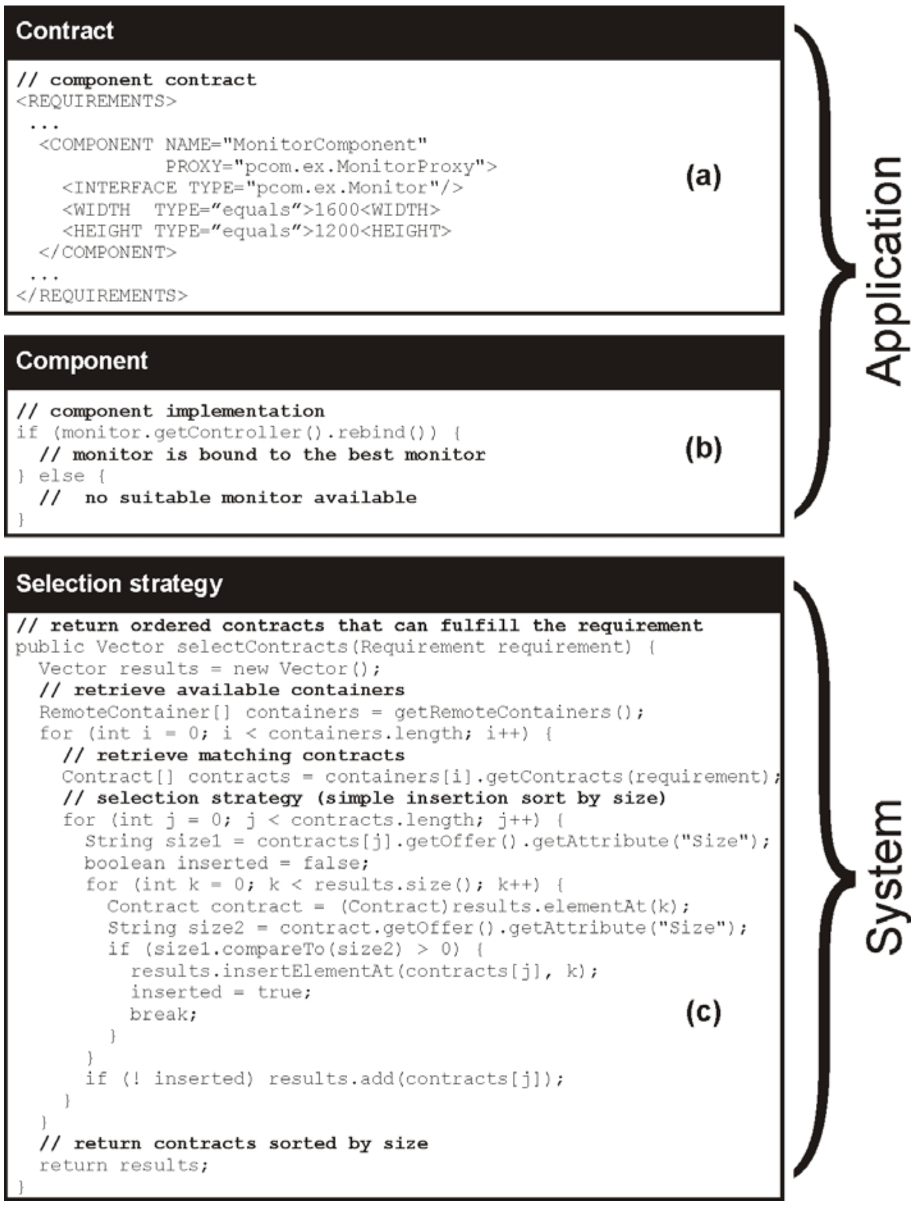
So far we have seen, how PCOM allows for generic application adaptation support via predefined as well as user-supplied strategies. The container realizing PCOM's runtime system resides on top of BASE, our middleware for Pervasive Computing. In the next section we will compare the abstractions provided by PCOM with the support BASE offers. The additional overhead for communication and application adaptation is presented based on measurements.

9.6. Evaluation

As stated in Section 9.4 the main requirements on PCOM are application specification and support for strategy-based adaptation. In PCOM these requirements are realized through components with contractually specified dependencies. As shown in Section 9.5.3, a crucial task for adaptation is the (re-)selection of services. Therefore, we will evaluate the service selection in PCOM and BASE. We compare the necessary tasks of a programmer and the assistance for service selection provided by PCOM and BASE. Next, the time needed for service selection is presented which includes contract evaluation, communication, and component instantiation. Finally, the additional requirements of PCOM regarding remote communication, memory, and computing power are discussed.

9.6.1. Service Selection

Selecting a service that will be used by an application comprises two fundamental tasks. First of all, an application has to determine the set of services that is available in a given environment. Thereafter, it has to determine the suitability of each service and select the best service possible.


Contract
(a)
Component
(b)
Selection strategy
(c)
Application
System

```
// component contract
<REQUIREMENTS>
 ...
  <COMPONENT NAME="MonitorComponent"
             PROXY="pcom.ex.MonitorProxy">
    <INTERFACE TYPE="pcom.ex.Monitor"/>
    <WIDTH  TYPE="equals">1600<WIDTH>
    <HEIGHT TYPE="equals">1200<HEIGHT>
  </COMPONENT>
 ...
</REQUIREMENTS>
```

```
// component implementation
if (monitor.getController().rebind()) {
  // monitor is bound to the best monitor
} else {
  //  no suitable monitor available
}
```

```
// return ordered contracts that can fulfill the requirement
public Vector selectContracts(Requirement requirement) {
  Vector results = new Vector();
  // retrieve available containers
  RemoteContainer[] containers = getRemoteContainers();
  for (int i = 0; i < containers.length; i++) {
    // retrieve matching contracts
    Contract[] contracts = containers[i].getContracts(requirement);
    // selection strategy (simple insertion sort by size)
    for (int j = 0; j < contracts.length; j++) {
      String size1 = contracts[j].getOffer().getAttribute("Size");
      boolean inserted = false;
      for (int k = 0; k < results.size(); k++) {
        Contract contract = (Contract)results.elementAt(k);
        String size2 = contract.getOffer().getAttribute("Size");
        if (size1.compareTo(size2) > 0) {
          results.insertElementAt(contracts[j], k);
          inserted = true;
          break;
        }
      }
      if (! inserted) results.add(contracts[j]);
    }
  }
  // return contracts sorted by size
  return results;
}
```

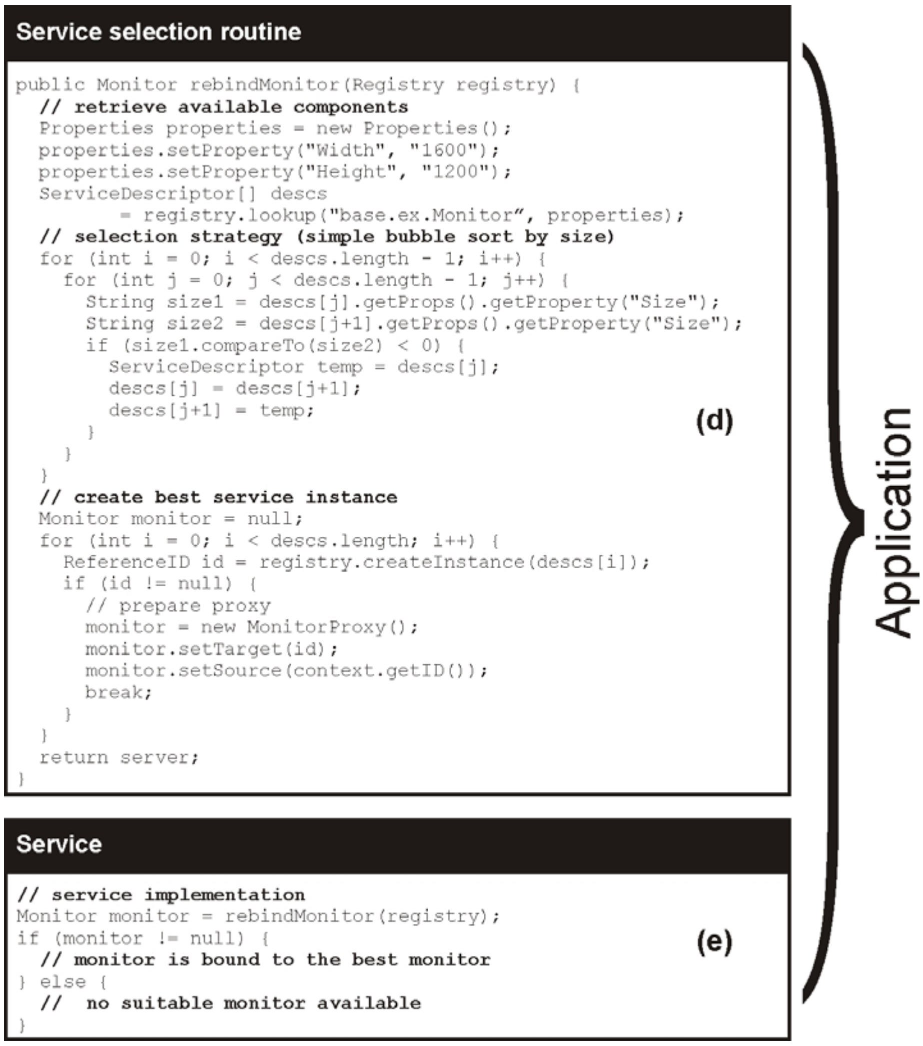
Figure 9.3: Component Selection in PCOM

To allow determining the suitability, BASE and PCOM support non-functional parameters that allow a more detailed description of services.

The suitability of a service could recursively depend on the suitability of the services used by it. As mentioned earlier, PCOM supports negotiation of dynamic parameters to model such dependencies. But since BASE does not deal with dynamic parameters, we restricted all parameters used during the evaluation to parameters that are static and thus, do not require negotiation.

Figure 9.3 shows the units of PCOM that are involved in the component selection process. An application programmer specifies the requirements of a component using a contract (a). At runtime, PCOM provides the application programmer with a handle for each component requested by the contract. Using this handle, a programmer can simply initiate the (re-) selection by calling the rebind-method (b). Typically, this method will be called within one of the listeners discussed above. When a reselection is initiated, PCOM uses contract matching to find suitable components and it uses a strategy to prioritize possible replacements (c). The distinction between contract and strategy separates the requirements that must be met to ensure the desired component behavior from user preferences. Notice, that (a) and (b) are supplied by a programmer, while (c) is a configurable and thus re-usable strategy that is integrated in the system.

Figure 9.4 shows how a similar behavior can be implemented using BASE. An application programmer provides a selection routine for the required service that specifies its properties and priorities (d). Whenever a reselection must take place, the application calls this routine (e). In contrast to PCOM, the selection routine provided by the application programmer encapsulates both, service requirements and preferences.


Service selection routine
(d)
Service
(e)
Application

```
public Monitor rebindMonitor(Registry registry) {
  // retrieve available components
  Properties properties = new Properties();
  properties.setProperty("Width", "1600");
  properties.setProperty("Height", "1200");
  ServiceDescriptor[] descs
         = registry.lookup("base.ex.Monitor", properties);
  // selection strategy (simple bubble sort by size)
  for (int i = 0; i < descs.length - 1; i++) {
    for (int j = 0; j < descs.length - 1; j++) {
      String size1 = descs[j].getProps().getProperty("Size");
      String size2 = descs[j+1].getProps().getProperty("Size");
      if (size1.compareTo(size2) < 0) {
        ServiceDescriptor temp = descs[j];
        descs[j] = descs[j+1];
        descs[j+1] = temp;
      }
    }
  }
  // create best service instance
  Monitor monitor = null;
  for (int i = 0; i < descs.length; i++) {
    ReferenceID id = registry.createInstance(descs[i]);
    if (id != null) {
      // prepare proxy
      monitor = new MonitorProxy();
      monitor.setTarget(id);
      monitor.setSource(context.getID());
      break;
    }
  }
  return server;
}
```

```
// service implementation
Monitor monitor = rebindMonitor(registry);
if (monitor != null) {
  // monitor is bound to the best monitor
} else {
  //  no suitable monitor available
}
```

Figure 9.4: Service Selection in BASE

The comparison of these two implementations shows that - from an application programmer's point of view - using a service in BASE is more complex than using a component in PCOM. While application programmers in BASE have to provide the functionality for searching and selecting required services, programmers in PCOM are provided with handles that hide the details of this selection. Instead of providing the specific algorithm that searches and prioritizes components, they simply specify the parameters that denote application-specific requirements and thus, they do not have to reason about user preferences. This means an additional flexibility which would be hard to achieve in a BASE implementation. Note that other, more complex features like contract negotiation or PCOM's signaling mechanisms are even harder to implement on top of BASE because of the lack of dynamic attributes.

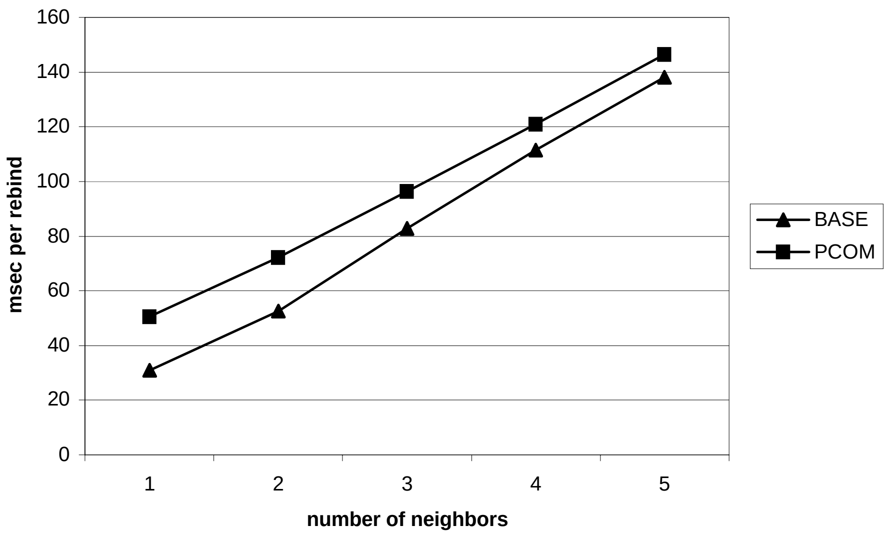


Figure 9.5: Component vs. Service Selection

Clearly, the extraction of functionality for selection causes an additional performance overhead. To quantify the impact on performance, we measured the time for a reselection in PCOM and in BASE. Figure 9.5 shows the average time for reselecting a service respectively a component (using the strategies and algorithms described in Figure 9.3/9.4) in cases where suitable components (or services in BASE) were available on 1 to 5 remote systems. The measurements have been conducted on PCs (Pentium III/600MHZ) connected with a 100 MBit network in order to show the fundamental effort without experiencing additional delays, such as Bluetooth discovery. The numbers shown in Figure 9.5 are the result of measuring 10 independent runs with 100 reselections each and varying the number of devices offering services (BASE) and containers (PCOM). To reduce fluctuations as far as possible, we disabled Java's just-in-time compiler. The remaining fluctuations were below 10 percent of the average time of a run and are most likely side-effects of the operating system's scheduler and Java's built-in garbage collector.

The total selection time is determined by the time for obtaining offers from neighbors, choosing an offer, and instantiating the chosen service or component. While the time for obtaining offers and choosing an offer increases linearly with the number of neighbors the instantiation of the chosen offer is constant. The measurements in Figure 5 show that, although reselection in PCOM is slower than in BASE, the relative overhead decreases with the number of neighbors. This is due to the higher cost for instantiating a PCOM component compared to a BASE service. The absolute overhead for a selection of approximately 30 ms however, is unlikely to be a bottleneck for realistic applications.

In addition to these measurements on resource-rich devices we have performed experiments on a JStamp embedded system[5] connected by a 19200 baud serial line. The average selection time was 3300 ms, which still may not impose serious problems, since a constant change of an application configuration, such as switching a monitor, will be annoying to the user.

In summary, comparing service and component selection shows that separating requirements and preferences using contracts and strategies is not for free. Although the overhead is noticeable, we believe that the gained flexibility is worth the performance penalty.

9.6.2. Communication

In order to compare the communication performance in BASE and PCOM, we measured the cost for a single message transfer using both systems. Our measurements showed that PCOM basically does not induce overhead on calls between components as it does not introduce

[5] http://www.jstamp.com

indirections in the dispatch chain. This in turn is a result of carefully integrating proxies and skeletons of BASE and PCOM.

In terms of general communication overhead, three mechanisms introduced by PCOM require additional remote communication. In contrast to services in BASE, components in PCOM use a soft-state protocol to detect the (un-)availability of components. This protocol transparently exchanges additional keep-alive messages if no other messages have been exchanged during a lease period. These messages represent an additional communication overhead for components that communicate infrequently. The second mechanism that introduces new messages is the discovery listener as it retrieves relevant contracts from devices that have been newly discovered. The last mechanism that requires additional remote communication is the contract listener. It creates a message for every modification of an offer or a requirement that is specified in a contract.

Clearly, all three mechanisms do not only create overhead, but do also provide necessary features. It is conceivable that realistic applications in dynamic environments must rely on soft-state protocols to reduce the amount of wastefully reserved resources. Similarly, components that have changing requirements or offers need to communicate them. Finally, optimization of executed applications requires notification about changes that could have positive impact.

Obviously, all three mechanisms could also be implemented in the application space, but it is questionable whether the possible performance benefit would outweigh the memory and engineering overhead of implementing all mechanisms within each component.

9.6.3. Resource Overhead

Apart from the cost of single mechanisms, PCOM has additional memory and processing requirements. In terms of memory usage, PCOM adds 30-40KB on top of 90-120KB required by BASE, resulting in a total memory usage of 120-160KB. With respect to processing, component instantiation and contract evaluation as well as all three mechanisms described in the previous section lead to increased requirements. The overhead for comparing contracts and instantiating components has already been discussed in the comparison of service and component selection. The processing requirements for the other mechanisms vary heavily depending on the applications and the environment and thus are hard to quantify.

In summary, the evaluation shows that the application of PCOM is not for free, but our results are promising. Compared to the baseline memory requirements of BASE, PCOM adds only little additional overhead. With respect to communication, the requirements do not change. Although the reselection overhead is more noticeable, we believe the gained flexibility is worth the cost.

9.7. Related Work

We will discuss related work in the areas of component systems, architectures for adaptation and evolution as well as recoverable computing, and pervasive computing.

Component Systems: Szyperski defines components as units of composition with contractually specified interfaces and explicit context dependencies only along with other properties [Szy98]. This definition conforms to our definition introduced in section 9.5. Existing component systems, e.g., CORBA CCM [OMG02c], Enterprise Java Beans [SunEJB], conform to this definition by introducing container abstractions to decouple components from the underlying platform and by providing – at least functional – contracts between components via interfaces. Such systems typically provide persistency and transactional behavior and are targeted at enterprise software rather than on resource constrained and dynamic environments, such as Pervasive Computing.

Adaptation Architectures and Recoverable Computing: The self configuration of software is addressed by a number of projects in the research area of application architectures. In contrast to our work, these projects typically consider adaptation to be a rather rare event, caused by errors or changes in the software's mission.

The *Weaves* approach [OGT+99] provides a general graph structure to model component dependencies. This leads to complex algorithms and additional specifications to support adaptation decisions. Therefore, this approach is too heavy-weight for resource poor devices and frequent adaptations.

The *recursive restartability approach* [PBB+02], proposed in the domain of recoverable computing, uses a tree-based application model quite similar to the PCOM model. Still, this model is specifically designed to allow the restart of failing components. The partitioning of the application follows the encapsulation of restartable units – not units of composition – and the only supported adaptation is a component re-instantiation. PCOMs application model is different in that it models the functional and non-functional properties of inter-component dependencies.

Pervasive Computing: The necessity of application adaptation is realized by a variety of projects that differ widely in their support for adaptation and the abstractions provided to application programmers. The system model considered is often based on smart environments, providing a set of services, such as lookup and persistent storage to devices that connect temporarily or permanently to the smart environment. In contrast to this, our system model does not assume connectivity to a smart environment but spontaneous connectivity to devices in the vicinity.

The *iROS* [JFW02] application model consists of atomic application parts which communicate via an event heap, realized as a tuple space. The event heap decouples distributed parts of an application. If functionality is not present, the request in the event heap is purged using an aging mechanism. Adaptation of applications is implicit, as functionality is only presented to the user if the application receives an answer to its request in the event heap.

One.world [GAB+00] is also based on a tuple space to allow communication between distributed parts of an application via events. Applications are composed of nested environments. Environments isolate applications from each other and serve as containers for persistent data. Conquering failure and selective availability is supported by providing mechanisms for application-specific automatic adaptation, such as migration or checkpointing along with persistent storage. Generic automatic adaptation is not supported.

Gaia [RC00] provides an application model based on a generalized model view controller pattern. An abstract definition of required functionality is mapped to the services available in a distinct smart environment (an active space). A coordinator component ensures that the application is executed as long as their integral parts are available. Adaptation is mainly considered to happen when a user moves to another active space and the matching of non-functional parameters is solely used to create a mapping between them.

The application model of *Aura* [GSS+02] provides a high level, user oriented task scheduler. Like PCOM, Aura aims at providing generic automatic adaptation support, but assumes a variety of services, e.g., remote communication, distributed file system, between remote Aura environments. PCOM is intended for environments, where this cannot be assured.

9.8. Conclusion

In this chapter we have presented PCOM, a light-weight component system supporting strategy-based adaptation in spontaneous networked Pervasive Computing environments. Using PCOM, application programmers rely on a component abstraction where interdependencies are contractually specified. The resulting application architecture is used for strategy-based adaptation of applications. Our results so far are promising. Based on our middleware BASE, PCOM adds only little memory overhead and basically no runtime overhead on communication. Overhead is introduced by the instantiation of components resulting in higher reselection time. However, this overhead decreases with the number of involved nodes. We conclude that providing a component abstraction along with generic adaptation support is possible with reasonable overhead even for resource-restricted devices.

Besides evaluating PCOM on a variety of different devices and communications technologies in our lab, we are currently evaluating PCOM's abstractions by developing further and more complex applications. From the gained experiences, we expect to identify additional generic adaptation mechanisms. Furthermore, we are working on generic adaptation mechanisms that will allow the reselection of stateful components. In the near future different adaptation strategies will be developed and evaluated using our system.

10. Experiences: Minimalism and Extensibility in BASE

In the vision of Ubiquitous Computing everyday objects become smart. Technically, this requires some sort of processing and communication technology. We have designed and implemented a middleware for spontaneous networking in Ubiquitous Computing environments. The major objectives were minimalism and extensibility in order to deploy the middleware on a variety of devices ranging from sensor nodes to classical general purpose computers. In this chapter we will assess the taken approach based on two follow-up projects: the port of BASE to a small embedded system and the design and implementation of a component system on top of BASE. While the fundamental concepts and design principles of BASE have proven to be solid, both projects provided insights that led to minor conceptual and major technical changes.

10.1. Introduction

Ubiquitous Computing (UC) [Wei91] envisions spontaneous interaction of computerized devices in order to achieve complex goals and support people's tasks. As in ordinary distributed system settings, interaction is achieved through the exchange of data and therefore is based on mechanisms that enable communication of computer systems. Support for communication in UC environments faces challenges that go beyond those of systems in static environments. Apart from the heterogeneity of devices which, to some degree, can also be found in ordinary distributed systems, UC is based on networks that form spontaneously and change dynamically. The mobility of devices makes it inevitable, that devices integrate in their ever-changing surrounding networks in order to utilize the functionality provided by them.

Resulting from the need to enable communication between heterogeneous computer systems in dynamic environments, a number of infrastructures have been proposed. These infrastructures are designed to provide an easy and efficient way of building and executing applications for ubiquitous computer systems. Depending on the degree of device mobility anticipated, they can be classified into two categories. The first category of infrastructures is based on the concept of smart environments. Prominent examples are Gaia [RC00], Aura [GSS+02] and iROS [JFW02]. They provide means to integrate small, mobile devices into relatively heavy weight environments with the immense processing power and storage capacities of today's desktop systems. The second category of infrastructures is targeted at supporting mobile devices with limited resources without relying on the processing power or storage capacity of the environment. Two representatives of this category are RCSM [YKW+02] and BASE [BSG+03], a middleware that supports spontaneous communication between devices. BASE has been designed to support a wide range of devices from sensor platforms to general purpose computers. Its micro-broker architecture allows the creation of a portable system with minimal hardware requirements, but it makes extension mechanisms inevitable in order to optimally utilize the capabilities of different devices.

In this chapter we present our experiences with porting BASE to a JStamp processor [Systronix], a Java-based embedded system supporting only the Java 2 Micro Edition [SunJ2ME] in the Connected Limited Device Configuration (CLDC). Further experiences where gained when we designed and implemented a component system for UC on top of BASE. Our experiences so far are promising. Both projects together enabled a first evaluation of minimalism and extensibility of BASE and led to optimizations regarding the internal mechanisms and external abstractions provided by this middleware.

The remainder of this chapter is structured as follows. Next, we will present an overview of BASE's architecture. Section three briefly describes the projects that led to the experiences described in this paper. In the forth section we will discuss the problems that we have encountered, their solutions and lessons learned. Section five summarizes and concludes the chapter.

10.2. BASE – A Micro-broker Based Middleware

In order to understand the approach taken during the design of BASE's architecture, it is necessary to explain the underlying requirements and design rationales. As a complete description would go beyond the scope of this chapter, and can be found in Chapter 7. We only present a brief overview before presenting the architecture. A more detailed presentation of BASE can be found in [BS03a] and Chapter 8.

10.2.1. Design Rationales

BASE was designed to fulfill three major requirements. First, application programmers should be provided with a *uniform programming interface* for accessing device capabilities, like a GPS receiver, and application objects, both, local and remote ones. This allows transparently switching functionality at runtime or more general, adapting to changes in the availability of functionality in a uniform way, e.g., by switching to a remote location service once the GPS receiver stops operating indoors. Therefore, in BASE, a service abstraction is provided to the application programmer to access device capabilities and application objects.

Second, the variety of different devices will likely lead to a number of different interoperability protocols with different communication models, e.g., events, remote procedure calls (RPC), etc. These should be *decoupled* by the middleware *from the application communication model*. This allows for example using an event-based interoperability protocol to deliver request/response messages of an RPC.

Last but not least, the middleware should be *minimal* and *tailorable*. This allows the installation on resource restricted devices, e.g., sensors, as well as using resources on more powerful devices, such as presentation systems or desktop computers.

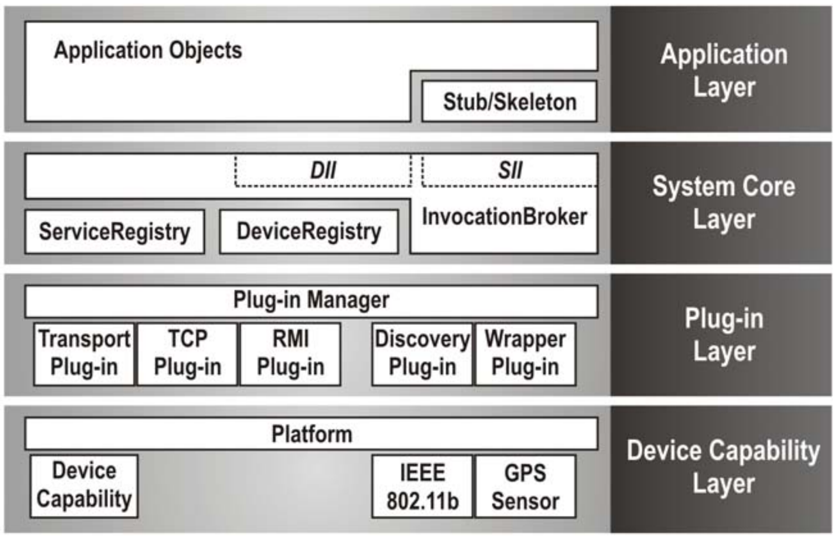


Figure 10.1. BASE Architecture

10.2.2. Architectural Overview

The architecture of BASE is depicted in figure 10.1. BASE offers application programmers a static (SII) and a dynamic invocation interface (DII). For the SII, stubs and skeletons are generated by a compiler and are used to map a method call to/from a so-called invocation object. If the DII is used, the application composes invocation objects directly. Invocation objects are Java objects, containing the *unmarshalled* invocation parts, like method name and parameters as well as further information on how to thread the invocation, e.g., which synchronization pattern should be used. While marshalling typically is a stub/skeleton responsibility, it was omitted on this layer and pushed down to the transport plug-ins to give the middleware maximum flexibility in choosing a suitable interoperability protocol at runtime.

In the system core layer, the invocation broker is responsible for delivering the invocation to either a local device capability or a remote service by choosing an appropriate plug-in. The invocation broker relies on information from the service registry (local services) and the device registry (currently reachable devices and the corresponding transport plug-ins) in order to dispatch an invocation. Since plug-ins can realize arbitrary protocols the invocation broker has to synchronize the invocation according to the application programming model and the underlying plug-in.

Plug-ins can be dynamically loaded and thus allow the extensibility of the middleware. The invocation broker follows the micro-kernel philosophy by only offering minimal functionality, i.e., how to find a service responsible for the invocation, dispatch it, and synchronize the invocation according to the application communication model. Thus, we call it a micro-broker.

Since all plug-ins, i.e., for device discovery, device capability, and transports rely on the same interface, i.e., handle invocations, applications can use the same programming interface (SII, DII) to access them. As stubs and skeletons do not provide any marshalling functionality, transport plug-ins have to ensure the marshalling of parameters and construction of interoperability protocol messages.

10.3. First Experiences

The first prototype of BASE was developed using an IBM J9 implementation of the Java 2 Micro Edition with the Connected Device Configuration (CDC). The CDC omits a variety of features from the Standard Edition, e.g., reflection, while others, such as object serialization, are present. Initial measurements [BSG+03] showed a reasonable small memory footprint of about 130 Kbytes but also that the initial marshalling resulted in two to three times overhead compared to Java RMI. This overhead mostly resulted from the naïve approach taken, i.e., serializing an invocation object with Java's object serialization.

10.4. Porting and using BASE

After the initial prototypical implementation that built upon the J2ME CDC platform, we started two projects related to BASE. One project ported BASE from its original platform the JStamp. The other project aimed at the development of a component system on top of BASE. The combined experiences created a picture that allowed an initial evaluation of both, the internal structure and the external abstractions.

10.4.1. Porting BASE

Although BASE is targeted at systems of all sizes we decided not to deal with all complexities that arise from the application of extremely restricted platforms during the development of the first prototype. Therefore, we did not build upon the most restricted platform defined by the J2ME specification. Instead we used the CDC, since it has a range of advanced features that allowed us to speed up the initial development. These features included for instance, JVM support for object serialization and dynamic class loading. The typical hardware that provides CDC sized runtime environments are high-end personal digital assistants or TV set-top boxes. Clearly, UC aims at devices that are even smaller. Therefore, we began to port BASE to the CLDC shortly after the first prototype was built successfully. The CLDC is targeted at devices including low-end personal digital assistants and embedded processors. Porting BASE required two tasks. First, we had to remove or reconstruct all convenient features that were solely available on CDC enabled systems. Second, we had to build platform specific transport and discovery plug-ins, since the JStamp processor did not support our existing IP-based transport and discovery plug-ins. Both tasks together gave us a chance to evaluate the internal structures when porting BASE to other platforms.

10.4.2. BASE as a Platform for Components

BASE aims at abstracting from platform specifics, but it leaves application programmers with only basic support, when dealing with fluctuating availability of local and remote services. As these fluctuations are inherent in mobile ad hoc networks, code of stable applications is necessarily tangled with code that manages dependencies on functionality provided by services. Since this kind of tangled code raises the complexity of application development, we decided to automate dependency management by the middleware using a component abstraction. The resulting component system used BASE as means of communication. Since we did not want to change the main mechanism and abstractions provided by BASE during its development, the component system can be seen as an application built on top of BASE. Therefore, this project enabled us to evaluate BASE's external structures that are used during application development.

10.5. Experiences

Before we present the lessons learned from conducting the port and the development of a component system, we will describe the resulting modifications to BASE. The modifications can be divided into two classes depending on their effects. The first class has been foreseeable and did not have conceptual impact. The second class is more interesting as it affects the fundamental concepts of BASE.

10.5.1. Technical Modifications

The additional restrictions imposed by the CLDC led to technical issues that could be resolved in a straight forward manner. Most noteworthy we were facing the following difficulties:

Class loading: the initial version of BASE made use of dynamic class loading in order to locate and execute plug-ins and services at runtime. As dynamic class loading is very restricted by the CLDC, we had to reduce this flexibility. Instead of dynamic class loading we modified BASE to use linked classes. We simplified the resulting more complex configuration process by providing a graphical configuration tool that generates desired configurations.

Object serialization: the CLDC does not provide means for serialization of objects. Since plug-ins are responsible for the marshalling, the first prototype of BASE simply serialized the invocation object. As mentioned before, this resulted in an unnecessary overhead and additionally, it was not possible on the CLDC. Our solution to this problem is straight forward. Via a serialization interface the marshalling code can access the object's state and write/read it to/from an output/input stream. We will later describe a solution for a more flexible and performance oriented plug-in structure.

10.5.2. Conceptual Modifications

BASE's plug-in concept offers a rather coarse grained structure currently including marshalling, interoperability, discovery, and transport layer abstractions. As the JStamp did not support our existing transport and discovery plug-ins, we had to develop new plug-ins. Although developing plug-ins is a fairly simple undertaking, due to their coarse grained structure, we were not able to reuse much of the existing code. Along with the marshalling performance mentioned earlier and current activities for

QoS management, we have to conclude that the plug-in concept so far provides suitable abstractions to interface to the micro-broker but requires additional structuring into an interoperability framework. Optimized marshalling code for distinct interfaces, service discovery, as well as transport layer related issues, e.g., SSL encryption, can be integrated via interceptors offering a simple configuration and re-use of these elements in other plug-ins.

Apart from the technical modification described earlier, the inability to load classes dynamically also led to conceptual changes. Just like JINI [Edw99] services, BASE services were designed to provide stubs for their clients. The automated delivery of stubs allows service-instance specific stubs and skeletons, but it relies on the ability to load classes dynamically. Porting BASE led to the conclusion that, due to its overall architecture, service-instance specific stubs and skeletons are an unnecessary feature. With respect to JINI services, loadable stubs are the only way to support flexible communication mechanisms. While BASE decouples stubs from the specifics of the transport and interoperability layers, JINI's stubs cut right through all communication layers. Therefore, JINI clients have to use the stub provided by the service. Otherwise they will not be able to create valid requests. The only functionality provided by BASE's stubs is the creation of Invocations. Encoding and transmission of data is handled by plug-ins. As a result, clients are able to include stubs for all services that they might use. The fact that BASE does not need service-instance specific stubs and skeletons results in a leaner ServiceRegistry.

10.5.3. Lessons Learned

From conducting both projects we learned a lot about the design decisions made during the initial development of BASE. A very obvious lesson that can be learned is that porting a Java-based system is not always as simple as some people claim. Although Java is usually considered to be a platform independent language, switching to a more restricted J2ME configuration can lead to costs that are comparable to the costs of porting platform dependent programs. Both, the lack of object serialization and dynamic class loading required the design of new mechanisms to achieve a similar level of convenience.

Apart from the platform related issues, the conceptual modifications provided two interesting insights. First, we learned that it is possible to use our plug-in concept to successfully build plug-ins for small devices. At the same time, we discovered that the granularity of the plug-in layer is not yet satisfactory. Therefore we have to conclude that the plug-in concept offers the required extensibility, but it needs a more sophisticated structure to increase reuse of existing code and to provide improved support for developers.

The second modification showed that the plug-in architecture allows removing service specific stubs and skeletons without loss of functionality. The extensibility provided by BASE's plug-in layer is sufficient to achieve at least the same degree of flexibility as systems like JINI.

The previously discussed lessons can be derived directly from modifications, but there are also lessons learned that result from keeping existing concepts. For example, one interesting feature of BASE that did not change during the projects is its reflection mechanism. In contrast to the Standard Edition, J2ME does not support reflection. However, in the presence of dynamic invocation creation and appropriate means to specify services and their interfaces via the service registry, a simple reflection mechanism is provided by BASE. It was an ongoing discussion in the team whether to aim for general reflection, i.e., storing signatures and class-relations in the service registry, or only providing interface names and the class-hierarchy information. So far, we have chosen the latter approach

without experiencing any restrictions. Our component system provides more powerful concepts for interface description and exploration including non-functional parameters and hence we decided to keep BASE minimal.
Another and probably the most important lesson that we have learned is also a result of not changing anything. During the development of the component system, there was no need to modify BASE. All necessary additions were implemented in the application layer. Only two extensions were integrated directly into BASE. First, components managed by the component system use stubs and skeletons that inherit from the original stubs and skeletons provided by BASE. This enables a faster dispatch of messages since there is no additional indirection in the dispatch chain. Second, some of the functionality provided by the Registries is accessed directly in order to remove indirections that might have negative impact on the performance of the system. Note, that these design decisions are performance optimizations. We could have done everything in the application layer (although this would have led to a much slower system). This brings us to the conclusion that BASE provides suitable abstractions for implementing applications as well as high-level infrastructures.
The successful development of the component system also raised questions. Our preliminary evaluation indicates that the overhead caused by a carefully designed component system is reasonably small compared with the initial cost of using BASE. The current version of BASE requires 90KB. Through the usage of the component system, these requirements are increased by 30KB. Considering target systems like the JStamp that have at least 1MB of memory, we are currently considering whether it makes sense to completely abandon the service abstraction and use components instead. But at the moment it is too early to fully assess all consequences of such a move.

10.6. Conclusions

In this chapter we have presented our experiences with conducting two projects that build upon BASE. While the internal structures have undergone technical and conceptual modifications, the external structures stayed remarkably stable. The conceptual modifications led to a follow up project, in which we began to design an improved plug-in layer to overcome the described deficiencies. Furthermore, we were able to successfully port BASE to a new set of target devices and to utilize it for a larger application. This success is encouraging and it shows that BASE is not only suited for smaller devices, but also that it can be used as infrastructure for applications as well as for further high-level abstractions. We are highly confident that the minimalism of our micro-broker approach together with the extensibility of its plug-in architecture will prove to be adequate for UC environments.

BASE and the component system are freely available to research institutions and can be downloaded at http://www.3pc.info.

11. Summary and Outlook

The proliferation of sensor technology and the miniaturization of computing devices already provide the foundations to capture the physical world's state. Integrating this state into applications allows presenting information and selecting services based on the physical world's state. Applications thus become context-aware. First examples of context-aware applications are already available on the market. Car navigation systems are based on road maps providing a model of the physical world. Integrating dynamic information, such as the current traffic information, allows the routing function to avoid traffic jams. Application scenarios for context-aware computing span all domains where human users interact with computer systems. Context-aware tourist guides, reminder services, home automation are examples for such application systems. Certain domains can clearly benefit from context-aware computing technology. Support for senior citizens could consists of body monitoring in order to ensure that help is called if some critical state is monitored. A smart pill dispenser can keep track of the correct medicine to be taken. Offering the latest health information plus a history to a physician in case of an emergency can help to provide the best-possible medical care. But not only human-centered computing can benefit from context-aware computing. A smart factory could track the position and state of tools and resources in order to integrate it into its resource management. This allows balancing stocking of resources and tools versus possible production downtimes due to their unavailability.

Context as a concept reflects all information that relates to the situation of entities relevant for applications and users. Applications can use context information by different means. If the context is stored in a context model and the application provides the precautions for adaptation to context changes we refer to this class of system support as *adaptation by application*.

Such context models are typically realized as context services which allow storing context information obtained by sensors, the application, or the user. Applications interface to such context services by using query languages in order to retrieve or modify context information. Based on such context models applications can select information and services depending on the context, change their presentation, issue some action, or allow to tag information to context. Context services can be designed for a single application, an application domain, or aim at generic context management. The underlying system models influence the context management. Infrastructure-based approaches can rely on one or a number of services offering the context information to applications. Applications thus have to access the infrastructure whenever context information is required. Another possible approach is based on ad hoc communication. Mobile devices are connected by wireless communication technology and form a spontaneous network. The unpredictable topology changes and the resulting network partitions prevent the management of context in a single service. Context information can be managed on a peer-to-peer-based fashion where mobile devices manage their context locally and exchange information with other devices. Location services for mobile ad hoc networks are examples of context information that is maintained collaboratively among mobile devices.

In contrast to context services which provide applications only with information about context but do not provide any support for application adaptation, support for *adaptation by system* exists as well. Applications are automatically configured depending on the available information and services. Spatial proximity as a major context information can be reflected either by a spatially restricted resource management or by using ad hoc communication. In the first case, which is common for smart environments, the management of a spatial area is provided by an infrastructure. The smart environments controls the integration and leaving of devices and mediates the interaction. Applications are mapped onto the available services. The relevance of information and services is reflected by being available through the smart environment which only manages a spatially restricted area, e.g., a meeting room or a smart home. Spatial relevance of information and services is naturally reflected in spontaneous networks based on mobile ad hoc networks. The communication between devices is based on wireless communication and thus devices in direct communication range are considered to be in proximity. In contrast to smart environments there is no central control provided by an infrastructure. This requires the system support to discover available services and information for each participating device and to adapt applications accordingly.

11.1. Contributions

This thesis provides a general discussion about system support for context-aware computing. A classification of system support along the dimensions of the underlying system model, i.e., ad hoc or infrastructure, and the system support for application adaptation, i.e., by system or by application, is given and structures this research area.

Specific contributions are made to the domains of support for adaptation by application. Location models as a basic structure of context models are presented and classified according to their suitability for supporting position, range, and nearest neighbour queries. The domain of context services in ad hoc systems is addressed by the Usenet-on-the-fly. This application allows users or applications to specify filters on information, which is exchanged between mobile devices in a mobile ad hoc network. The local relevance of information is reflected by the dissemination algorithm that propagates information between devices whenever they are in communication range. An improved version of this algorithm is provided which allows scheduling the advertisement of messages based on their popularity. Another contribution to the field of context services are the experiences gained from the integration of a local context server for Georgia Tech's Aware Home into the Nexus platform.

The research area of Peer-to-Peer Pervasive Computing – support for adaptation by system in an ad hoc setting – is covered in the remaining part of this thesis. First, a requirement analysis of this class of system support is given. Second, a flexible middleware platform that allows for spontaneous cooperation in Peer-to-Peer ad hoc systems is introduced. A micro-broker design allows minimal installation but also flexible extensibility. Third, the support for application adaptation by system is addressed by a component system. Based on the middleware a component-based application model is designed. The container managing the components can automatically adapt to resource changes because of the explicit dependencies modelled in the component's contracts. Fourth, the experiences of porting the middleware to resource restricted devices and building the component container close the thesis showing the feasibility of the introduced concepts.

11.2. Outlook

Context-aware computing already starts to become available in products, such as navigation systems. The integration of sensor and computing platforms as embedded systems into everyday objects as a trend can also be observed. The next challenges in context management are common context models allowing applications to share and reason about context. Standardized query languages along with the context models are required to share the costs of gathering and managing context.

Context-aware applications react to the changes in the physical world along with other context information, such as user preferences. As a result, these applications will change their behaviour over time with the context

information. The potentially high number of context information and the resulting combinations require applications to either neglect relevant context information or to deal with this information. Clearly, this complicates the task of developing and maintaining context-aware applications. In an ideal case, application programmers are provided with means to specify the variations in an application's behaviour and the system support automatically configures the application depending on the current context. A similar situation can be found in Peer-to-Peer Pervasive Computing. The fluctuation of services and resources in general will require constant adaptation of application to the ever-changing execution environment. If the application programmer is assisted by higher level support, such as a contract-based application model, e.g., the one provided by PCOM, the system performs constant self-configuration. Classical computer systems are also challenged by the number of involved components. The complexity of such systems leads to high effort for fault-isolation and configuration. The vision of Autonomic Computing aims at self-organizing, self-healing, and self-optimizing systems. The similarity of objectives of Autonomic and Pervasive Computing – as discussed in [WPT03] - leads to the questions how the concepts for adaptation and specification of applications can be transferred between these domains. Clearly, the system models differ. Systems considered by Autonomic Computing are of higher complexity and changes, such as errors or reconfigurations, are happening on a lower rate than in Pervasive Computing. The core problem, that a system has to adapt to a variety of potentially unknown changes, however, stays the same.
With basic technology being available, the vision of Pervasive Computing can become reality. Research challenges ahead do not only affect core computer science disciplines but also business cases for the deployment of

such systems as well as the social implications of a world populated with sensing, computing, and communication capabilities.

12. References

[3PC] Peer-to-Peer Pervasive Computing Project (3PC): http://www.3pc.info

[AAH+97] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, M. Pinkerton "Cyberguide: A mobile context-aware tour guide", Wireless Networks 3(5) (1997) 421-433

[ACH+01] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, A. Hopper, "Implementing a Sentient Computing System", IEEE Computer Magazine, vol. 34, no. 8, pp. 50-56, August 2001

[BA01] L. Bergmans and M. Aksit "Composing crosscutting concerns using composition filters", Communications of the ACM, 44(10), Oct. 2001.

[BBH02] C. Becker, M. Bauer, J. Hähner "Usenet on the fly - supporting the locality of information in spontaneous networking environments", Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments in conjunction with ACM Conference on CSCW 2002, New Orleans/USA, 2002

[BBR01] M. Bauer, C. Becker, K. Rothermel "Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks", Workshop on Location Modeling for Ubiquitous Computing, UBICOMP 2001, Atlanta, 2001.

[BBR02] M. Bauer, C. Becker, K. Rothermel "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)

[BCA+01] G.S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikoski, "The Design and Implementation of Open ORB version 2". IEEE Distributed Systems Online Journal, vol. 2, no. 6, 2001

[BCR+00] G. S. Blair, G. Coulson, P. Robin, M. Papathomas, "An Architecture for Next Generation Middleware", Proceedings of Middleware 2000, Lake District, UK, 2000

[BCS+98] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward "A distance routing effect algorithm for mobility (DREAM)" In Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom'98, Dallas, TX, 1998.

[BD04] C. Becker, F. Dürr "On Location Models for Ubiquitous Computing" accepted for publication in Personal and Ubiquitous Computing, Springer, 2004

[BG00] C. Becker and K. Geihs, "Generic QoS-Support for CORBA", Proceedings of the 5th IEEE Symposium on Computers and Communications (ISCC'2000) Antibes, France, 2000

[BHS+04] C. Becker, M. Handte, G. Schiele, K. Rothermel "PCOM - A Component System for Pervasive Computing", In Proceedings 2nd IEEE International Conference on Pervasive Computing and Communication (PerCom 04), Orlando, USA, 2004

[BJP+99] A. Beugnard, J.M. Jezequel, N. Plouzeau, and D. Watkins "Making components contract aware" IEEE Computer, 13(7), July 1999.

[BJR+03] M. Bauer, L. Jendoubi, K. Rothermel, E. Westkämper "Grundlagen ubiquitärer Systeme und deren Anwendung in der Smart Factory" Industrie Management – Zeitschrift für industrielle Geschäftsprozesse, 19(6), 2003

[BKW02] J. Baus, A. Krüger, W. Wahlster "A resource-adaptive mobile navigation system" In Proccedings of International Conference on Intelligent User Interfaces, San Francisco, 2002

[BMJ+98] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, J. Jetcheva "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98), Dallas, Texas, 1998.

[BMK+00] B. Brumitt, B. Meyers, J. Krumm, A. Kern, S. Shafer "EasyLiving: Technologies for Intelligent Environments" Handheld and Ubiquitous Computing (HUC), Bristol, UK, 2000

[BMR04] S. Bürklen, P. Marrón, K. Rothermel "An Enhanced Hoarding Approach Based on Graph Analysis" In Proceedings of the 5th IEEE International Conference on Mobile Data Management (MDM 2004); Berkeley, California, USA, 2004

[BR04] Bauer, Martin; Rothermel, Kurt: "How to Observe Real-World Events through a Distributed World Model" In: to appear in: Proceedings of the Tenth International Conference on Parallel and Distributed Systems 2004 (ICPADS 2004);Newport Beach, California, July 7-9, 2004

[BS01] B. Brumitt, S. Shafer "Topological World Modeling Using Semantic Spaces", In Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, Atlanta, Georgia, USA, Sep 2001

[BS03a] C. Becker, G. Schiele "BASE: A Minimal yet Extensible Platform for Pervasive Computing", International Conference on Tales of the Disappearing Computer, Santorin, Greece, 2003

[BS03b] C. Becker, G. Schiele "Middleware and Application Adaptation Requirements and their Support in Pervasive Computing", 3rd International Workshop on Distributed Auto-adaptive and Reconfigurable Systems (DARES) at ICDCS, pp. 98-103, May 19-22, Providence, USA, 2003

[BSG+03] C. Becker, G. Schiele, H. Gubbels, K. Rothermel "BASE - A Micro-broker-based Middleware For Pervasive Computing", Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communication, pp. 443-451, Fort Worth, USA, March 2003

[CDM+00a] K. Cheverst, N. Davies, K. Mitchell, A. Friday, C. Efstratiou "Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences", Proceedings of CHI 2000, Netherlands (2000)

[CDM+00b] K. Cheverst, N. Davies, K. Mitchell, and A. Friday "Experiences of developing and deploying a context-aware tourist guide: the GUIDE project", In Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, Boston, Massachusetts, 2000, 20-31

[CGS+02] S.W. Cheng, D. Garlan, B. Schmerl, J. P. Sousa, B. Spitznagel, P. Steenkiste, and N. Hu "Software architecturebased adaptation for pervasive systems", In International Conference on Architecture of Computing Systems (ARCS'02): Trends in Network and Pervasive Computing, Apr. 2002.

[CK00] G. Chen, D. Kotz "A Survey of Context-Aware Mobile Computing Research", Dartmouth Computer Science Technical Report TR2000-381, Dartmouth College (2000)

[CKW01] W. S. Conner, L. Krishnamurthy, R. Want "Making Everyday Life Easier Using Dense Sensor Networks", In Proceedings of UBICOMP 2001, Atlanta, USA (2001)

[CLC+02] N. H. Cohen, H. Lei, P. Castro, J. S. Davis II, A. Purakayastha "Composing Pervasive Data Using iQL" 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), Callicoon, New York, 2002

[Cooltown] Cooltown, http://www.cooltown.com/cooltownhome/index.asp

[CPW+02] N. H. Cohen, A. Purakayastha, L. Wong, D. L. Yeh "iQueue: a pervasive data-composition framework" 3rd International Conference on Mobile Data Management, Singapore, 2002

[CPW+99] M.H. Coen, B. Phillips, N. Warshawsky, L. Wiesman, S. Peters, P. Finin, "Meeting the Computational Needs of Intelligent Environments: The Metaglue System", Proceedings of the 1st International Workshop Managing Interactions in Smart Environments (MANSE'99), Dublin, Ireland, pp. 201-212, December 1999

[DA99] A. Dey, G. Abowd "Towards a better understanding of context and context-awareness", Georgia Tech GVU Technical Report, GIT-GVU-99-22 (1999)

[DALLAS] Accuracy of RTC: http://dbserv.maxim-ic.com/appnotes.cfm/appnote_number/632

[DGH+87] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, D. Terry "Epidemic Algorithms for Replicated Database Maintenace", In Proceedings of the 6th ACM Symposium on Principles of Distributed Computing, pp. 1-12, 1987.

[DR03] F. Dürr, K. Rothermel "On a Location Model for Fine-Grained Geocast", In Proceedings of the Fifth International Conference on Ubiquitous Computing (UbiComp 2003), Seattle, WA, Oct 2003, 18-35

[Dro03] T. Drosdol: Unterstützung symbolischer Koordinaten im Lokationsmanagement; Diploma thesis, University of Stuttgart, 2003

[Edw99] W. K. Edwards "Core JINI" The SUN Microsystems Press Java Series, Prentice Hall, 1999

[EE98] G. Eddon, H. Eddon "Inside Distributed Com", Microsoft Press, February 1998

[EPS+01] F. Espinoza, P. Person, A. Sandin, H. Nyström, E. Cacciatore, M. Bylund "GeoNotes: Social and Navigational Aspects", In Proceedings of UBICOMP 2001, Atlanta, USA (2001)

[ETH] ETH World, http://www.ethworld.ethz.ch

[FHM+04] A. Ferscha, M. Hechinger, R. Mayrhofer, R. Oberhauser "A Light-Weight Component Model for Peer-to-Peer Application" In the 24th IEEE International Conference on Distributed Computing Systems Workshops, Tokyo, Japan, 2004

[GAB+00] R. Grimm, T. Anderson, B. Bershad, and D. Wetherall. "A system architecture for pervasive computing", In Proceedings of the 9th ACM SIGOPS European Workshop, pp. 177-182, Denmark, September 2000.

[GPS] Information GPS: http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html

[GSF+01] C. H. Ganoe, W. A. Schafer, U. Farooq, J. M. Carroll "An Analysis of Location Models for MOOsburg", In Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, Atlanta, Georgia, USA, Sep 2001

[GSM] GSM World. Location-based Services.
http://www.gsmworld.com/technology/applications/location.shtml

[GSS+02] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste "Project Aura: Towards Distraction-Free Pervasive Computing", IEEE Pervasive Computing, special issue on "Integrated Pervasive Computing Environments", Volume 1, Number 2, (Apr-Jun 2002) 22-31

[HB01] J. Hightower and G. Borriello "Location systems for ubiquitous computing", IEEE Computer 34(8), 2001, 57-66

[HBB02] J. Hightower, B. Brumitt, and G. Borriello "The Location Stack: A Layered Model for Location in Ubiquitous Computing", In Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002), Callicoon, NY, 2002, 22-28

[HBR03] J. Hähner, C. Becker, and K. Rothermel "A Protocol for Data Dissemination in Frequently Partitioned Mobile Ad Hoc Networks", In Proceedings of the IEEE Symposium on Computers and Communications (ISCC 2003), Antalya/Turkey, 2003

[HHS+99] A. Harter, A. Hopper, P. Steggles, A. Ward, P. Webster "The anatomy of a context-aware application", In Proceedings fifth annual International Conference on Mobile Computing and Networking (MobiCom'99), Seattle, WA

[HKB99] W.R. Heinzelman, J. Kulik, H. Balakrishnan "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks",In Proceedings fifth annual International Conference on Mobile Computing and Networking (MobiCom'99), Seattle, WA

[HKL+99] F. Hohl, U. Kubach, A. Leonhardi, K. Rothermel, M. Schwehm "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, WA

[HL04] H. Hu and D. L. Lee "Semantic Location Modeling for Location Navigation in Mobile Environments", In Proceeding of the IEEE International Conference on Mobile Data Management, Berkeley, California, USA, Jan 2004

[HOT+99] C. Ho, K. Obraczka, G. Tsudik, K. Viswanath "Flooding for Reliable Multicast in Multi-Hop Ad Hoc Networks", Proceedings of MobiCom Workshop on Discrete Algorithms and Methods for Mobility (DialM'99), 1999.

[JFW02] B. Johanson, A. Fox, T. Winograd, "The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms", IEEE Pervasive Computing, vol. 1, no. 2, pp. 67-74, April-June 2002

[Joh94] D. Johnson "Routing in Ad Hoc Networks of Mobile Hosts", In Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, USA, 1994.

[JS02] C. Jiang, P. Steenkiste "A hybrid location model with a computable location identifier for ubiquitous computing", In Proceedings of the Fourth International Conference on Ubiquitous Computing (UbiComp 2002), Goteborg, Sweden, 2002, 246-263

[JS03] G. Judd, P. Steenkiste "Providing Contextual Information to Pervasive Computing Applications", In Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom), 2003, 133-142

[JSprint] J-Sprint Homepage, http://www.j-sprint.com/

[JStamp] JStamp Homepage, http://jstamp.systronix.com/index.htm

[JTK97] A.D. Joseph, J.A. Tauber, and M.F. Kaashoek, "Mobile Computing with the Rover Toolkit", IEEE Transactions on Computers: Special issue on Mobile Computing, vol. 46, no. 3, pp. 337-352, March 1997

[Kan01] G. Kan "Gnutella" In Peer-to-Peer: Harnessing the Power of Disruptive Technologies, Andy Oram (ed.), O'Reilly, March 2001.

[KEG93] W. Kainz, M. J. Egenhofer, I. Greasley "Modeling spatial relations and operations with partially ordered sets" International Journal of Geographic Information Systems 7(3), 1993, 215-229

[KF02] T. Kindberg, A. Fox, "System Software for Ubiquitous Computing", IEEE pervasive computing, vol. 1, no. 1, pp. 70-81, January-March 2002

[KHB99] J. Kulik, W. Heinzelman, H. Balakrishnan "Negotiation-based protocols for disseminating information in wireless sensor networks", In Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking, Seattle, WA, USA, 1999.

[KL01] O. Kasten, M. Langheinrich "First Experience with Bluetooth in the Smart-Its Distributed Sensor Networks", Workshop on Ubiquitous Computing and

Communication. In Proceedings of 10th International Conference on Parallel Architectures and Compilation Techniques (PACT'01), Barcelona Spain, 2001.

[KMP99] G. Karumanchi, S. Muralidharan, R. Prakash "Information Dissemination in Partitionable Mobile Ad Hoc Networks", Proceedings of 18th IEEE Symposium on Reliable Distributed Systems, 1999.

[KOA+99] C. Kidd, R. Orr, G. Abowd, C. Atkeson, I. Essa, B. MacIntyre, E. Mynatt, T. Starner, W. Newstetter "The Aware Home: A Living Laboratory for Ubiquitous Computing Research", In Proceedings of the Second International Workshop on Cooperative Buildings - CoBuild'99, 1999

[KSS+99] G. Kortuem, J. Schneider, J. Suruda, S. Fickas, Z. Segall "When Cyborgs Meet: Building Communities of Cooperating Wearable Agents", In Proceedings Third International Symposium on Wearable Computers (ISWC'99), San Francisco, CA, USA, 1999.

[LBB+04] O. Lehman, M. Bauer, C. Becker, D. Nicklas "From Home to World: Supporting Context-Aware Applications through World-Models", In 2nd IEEE International Conference on Pervasive Computing and Communication (PerCom 04), Orlando, USA, 2004

[LBS] LBS Portal, http://www.lbsportal.com/

[Led99] T. Ledoux, "OpenCorba: A Reflective Open Broker", Proceedings of the 2nd International Conference on Reflection (Reflection'99), pp. 197-214, Saint-Malo, France, 1999

[Leo98] U. Leonhardt "Supporting location-awareness in open distributed systems", PhD thesis, Imperial College London, Department of Computing, 1998

[LJD+00] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, R. Morris "A Scalable Location Service for Geographic Ad hoc Routing" in Proceedings Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'00), Boston, USA

[LKA+96] S. Long, R. Kooper, G. D. Abowd, C. G. Atkeson "Rapid prototyping of mobile context-aware applications: the Cyberguide case study" In Proceedings of the second annual International Conference on Mobile Computing and Networking, White Plains, NY, 1996

[LKR99] A. Leonhardi, U. Kubach, K. Rothermel "Virtual Information Towers - A metaphor for intuitive, location-aware information access in a mobile environment" In Proceedings of third International Symposium on Wearable Computers, San Francisco, CA (1999) 15-20

[LSD+02] H. Lei, D. M. Sow, J. S. Davis II, G. Banavar, M. R. Ebling "The design and applications of a context service" Mobile Computing and Communications Review **6**, No. 4, October 2002

[Loo01] D. Loomis, The TINI(tm) Specification and Developer's Guide, Addison-Wesley, June 2001

[LR02] A. Leonhardi, K. Rothermel "Architecture of a Large-scale Location Service", In Proceedings of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS 2002), Vienna, Austria (2002) 465-466

[Mic00] Microsoft Corporation, Universal Plug and Play Device Architecture, Version 1.0, http://www.upnp.org/download/ UPnPDA10_20000613.htm, June, 2000

[Moz98] M. Mozer, "The Neural Network House: An Environment that Adapts to its Inhabitants", AAAI Spring Symposium, Stanford, pp. 110114, March 1998

[MS00] N. Marmasse, C. Schmandt "Location-aware information delivery with commotion" In Proceedings of the second International Symposium on Handheld and Ubiquitous Computing (HUC), Bristol, UK, 2000

[MS01] N. Marmasse, C. Schmandt "Location Modeling" In Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, Atlanta, Georgia, USA, Sep 2001

[NC01] H. Naguib, G. Coulouris "Location Information Management" In Proceedings of the 3rd International Conference on Ubiquitous Computing (UbiComp 2001), Atlanta, Georgia, USA, Sep 2001

[NGS+01] D. Nicklas, M. Großmann, T. Schwarz, and S. Volz "A Model-Based, Open Architecture for Mobile, Spatially Aware Applications", In Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD 2001), Redondo Beach, CA, USA, Jul 2001

[NKO+01] K. Nagel, C. D. Kidd, T. O'Connell, A. Dey, G. D. Abowd, "The Family Intercom: Developing a Context-Aware Audio Communication System" G. D. Abowd, B. Brumitt, S. A. N. Shafer (Eds): Ubicomp 2001, LNCS 2201, Springer-Verlag Berlin Heidelberg (2001), 176-183

[NM01] D. Nicklas, B. Mitschang "The Nexus Augmented World Model: An Extensible Approach for Mobile, Spatially-Aware Applications", In Proceedings of the 7th Int. Conf. on Object-Oriented Information Systems (2001)

[NTC+99] S.-Y. Ni, Y.-C. Tseng, Y.-S Chen, J.-P. Sheu "The Broadcast Storm Problem in a Mobile Ad Hoc Network", Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCom'99), 1999.

[ODA01] T. O'Connel, P. Jensen, A. Dey, and G. Abowd: Location in the Aware Home; In Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, Atlanta, Georgia, USA, Sep 2001

[OGT+99] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, A.L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software", IEEE Intelligent Systems, vol. 14, no. 3, pp. 54-62, May-June 1999

[OMG98] Object Management Group, CORBA Messaging, report orbos/98-05-06, 1998

[OMG02a] Object Management Group, Minimum CORBA Specification, Revision 1.0. August 2002

[OMG02b] Object Management Group, The Common Object Request Broker: Architecture and Specification, Revision 3.0. July 2002

[OMG02c] Object Management Group (OMG), "CORBA Component Model V3.0", formal/2002-06-65, 2002

[OpenGIS] Open GIS Consortium Inc.: OpenGIS Simple Features Specification for SQL, <http://www.opengis.org/techno/specs/99-049.pdf>

[OT98] K. Obraczka, G. Tsudik "Multicast Routing Issues in Ad Hoc Networks", Proceedings of the IEEE International Conference on Universal Personal Communication (ICUPC'98), 1998.

[Oxygen] Oxygen System Group Homepage. http://o2s.lcs.mit.edu/.

[Pas97] J. Pascoe "The Stick-e Note Architecture: Extending the Interface Beyond the User", In Proceedings of the International Conference on Intelligent User Interfaces. Editors, Moore, J., Edmonds, E., and Puerta, A., pp. 261-264, 1997

[PB94] C. E. Perkins, P. Bhagwat "Highly Dynamic Destination-Sequenced Distance Vector Routing (DSDV) for Mobile Computers", In Proceedings of ACM SIGCOMM'94, London, UK, pp. 234-244, 1994.

[PBB+02] D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, N. Treuhaft, "Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies", UC Berkeley Computer Science Technical Report UCB//CSD-02-1175, March 2002

[Pebbles] http://www.cag.lcs.mit.edu/~umar/publications/pebbles-abstract.pdf

[PostGIS] PostGIS, <http://postgis.refractions.net>

[PPL+03] G. Pingali, C. Pinhanez, A. Levas, R. Kjeldsen, M. Podlaseck, H. Chen, N. Sukaviriya "Steerable Interfaces for Pervasive Computing Spaces", In Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom), 2003, 315 322

[PR00] A. Puder, K. Roemer, MICO: An Open Source CORBA Implementation, 3rd edition, Morgan Kaufmann Publishers, March 2000

[PS01] M. Papadopuli, H. Schulzrinne "Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices", In Proceedings of MobiHoc 2001, Long Beach, USA, 2001.

[RBB03] K. Rothermel, M. Bauer, C. Becker "Digitale Weltmodelle – Grundlage kontextbezogener Systeme", in Total Vernetzt, Ed. F. Mattern, Springer, 2003

[RBB+03] A. Roy, S. K. D. Bhaumik, A. Bhattacharya, K. Basu, D. J. Cook, and S. K. Das "Location Aware Resource Management in Smart Homes", In Proceeding of the First

IEEE International Conference on Pervasive Computing and Communications (PerCom '03), Fort Worth, USA, Mar 2003

[RC00] M. Román and R.H. Campbell, "GAIA: Enabling Active Spaces", Proceedings of the 9th ACM SIGOPS European Workshop, pp. 229-234, Kolding, Denmark, September 2000

[RC01] M. Román and R.H. Campbell, "Unified Object Bus: Providing Support for Dynamic Management of Heterogeneous Components", Technical Report UIUCDCS-R-2001-2222 UILU-ENG-2001-1729, Universiy of Illinois at Urbana-Champaign, 2001

[RDD+03] K. Rothermel, D. Dudkowski, F. Dürr, M. Bauer, C. Becker "Ubiquitous Computing – More than Computing Anytime Anyplace", In Proceedings of the 49th Photogrammetric Week, Stuttgart, Germany, Sep 2003

[RJO+89] R. Rashid, D. Julin, D. Orr, R. Sanzi, R. Baron, A. Forin, D. Golub, M. Jones, "Mach: A System Software kernel", Proceedings of the 34th Computer Society International Conference (COMPCON 89), San Francisco, CA, February 1989

[RKC99] M. Román, F. Kon and R.H. Campbell, "Design and Implementation of Runtime Reflection in Communication Middleware: The DynamicTAO Case", Proceedings of the 1999 ICDCS Workshop on Electronic Commerce and Web-Based Applications, pp. 122-127, Los Alamitos, CA, 1999

[RKC01] M. Román, F. Kon, and R.H. Campbell, "Reflective Middleware: From Your Desk to Your Hand", IEEE Distributed Systems Online Journal, Special Issue on Reflective Middleware, July 2001

[RLU94] M. Rizzo, P. Linington, I. Utting "Integration of Location Services in the Open Distributed Office", Technical Report 12/94, University of Kent, Canterbury, UK, 1994

[RMK+00] M. Román, D. Mickunas, F. Kon, R.H. Campbell, "LegORB and Ubiquitous CORBA", IFIP/ACM Middleware'2000 Workshop on Reflective Middleware, NY, April 2000

[Roem01] Römer, K.: „Time Synchronization in Ad Hoc Networks", Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'01), 2001.

[RSC+99] M. Román, A. Singhai, D. Carvalho, C. Hess, R.H. Campbell, "Integrating PDAs into Distributed Systems: 2K and PalmORB", International Symposium on Handheld and Ubiquitous Computing (HUC'99), Karlsruhe, Germany, September 1999

[Sat96] M. Satyanarayanan "Fundamental Challenges in Mobile Computing" in Proceedings 15th ACM Symposium on Principles of Distributed Computing, Philadelphia, USA, 1996

[SAW94] B. N. Schilit, N. A., Roy Want "Context-Aware Computing Applications" IEEE Workshop on Mobile Computing Systems and Applications, 1994

[SBG99] A. Schmidt, M. Beigl, and H.-W. Gellersen "There is more to context than location", Computers and Graphics 23(6), 1999, 893-901

[Sch95] W. N. Schilit "A System Architecture for Context-Aware Mobile Computing" PhD thesis, Columbia University, 1995

[SDA99] D. Salber, A. Dey, G. Abowd, G.: The Context Toolkit: Aiding the Development of Context-Enabled Applications. Proceedings of CHI (1999)

[SensorML] Open GIS Consortium Inc.: SensorML http://www.opengis.org/docs/02-026r4.pdf

[SKJ+00] J. Schneider, G. Kortuem, I. Jager, S. Fickas, Z. Segall "Disseminating Trust Information in Wearable Communities", In Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K), Bristol, UK, 2000.

[Ste02] I. Stepanov "Integrating Realistic Mobility Models In Mobile Ad Hoc Network Simulation", Diploma Thesis No. 1989, in English, Department of Computer Science, University if Stuttgart, 2002. Available at ftp://ftp.informatik.uni-stuttgart.de/ pub/library/medoc.ustuttgart_fi/DIP-1989/DIP-1989.pdf.

[STM00] A. Schmidt, A. Takaluoma, J. Mäntyjärvi "Context-aware telephony over WAP", In Personal Technologies 4 (4) (2000) 225-229

[SunEJB] SUN Microsystems, "Enterprise Java Beans Specification", http://java.sun.com/products/ejb/docs.html, 2003

[SunJ2EE] Sun Microsystems. Java 2 platform, enterprise edition. http://java.sun.com/j2ee.

[SunJ2ME] Java Micro Edition Homepage, http://java.sun.com/j2me/

[SunRMI] Java Remote Method Invocation Specification. Revision 1.8, Sun Microsystems, available online: http://java.sun.com/j2se/1.4/docs/ guide/rmi/index.html, 2002

[Swiss] Swisscom. Go Mobile. http://www.gomobile.ch/static/en/community/community_web_public.htm

[Systronix] Systronix Inc home page, http://www.jstamp.com/

[Szy98] C. Szyperski "Component Software Beyond ObjectOriented Programming", AddisonWesley, 2nd edition, 1998.

[Tan96] A. Tanenbaum "Computer Networks - Third Edition", Prentice Hall, 1996.

[TAO] D.C. Schmidt, Minimum TAO. http://www.cs.wustl.edu/~schmidt/ACE_wrappers/docs/minimumTAO.html

[THB+02] J. Tian., J. Hähner, C. Becker, I. Stepanov, K. Rothermel "Graph based Mobility Model for Mobile ad-hoc Network Simulation", In Proceedings of the 35th Annual Simulation Symposium (ANNSS35), San Diego, USA, 2002.

[TKR+91] A.S. Tanenbaum, M.F. Kaashoek, R. van Renesse, and H. Bal, "The Amoeba Distributed Operating System-A Status Report", Computer Communications, vol. 14, no. 6, pp. 324-335, July/August 1991

[VB00] A. Vahdat, D. Becker "Epidemic Routing for Partially Connected Ad Hoc Networks", Technical Report CS-200006, Duke University, USA, 2000.

[VGH+02] S. Volz, M. Großmann, N. Hönle, D. Nicklas, T. Schwarz „Integration mehrfach repräsentierter Straßendaten für eine föderierte Navigation", In it+ti, Informationstechnik und Technische Informatik 5, 2002

[Wal99] J. Waldo, "The Jini Architecture for network-centric computing", Communications of the ACM, vol. 42, no. 7, pp. 76-82, July 1999

[WBG+01] T. Weis, C. Becker, K. Geihs, N. Plouzeau „An UML metamodel for contract aware components", In Proceedings of UML 2001, 2001.

[Wei91] M. Weiser, "The computer for the 21st century", Scientific American, vol. 265, no. 3, pp. 94-104, September 1991

[WHG92] R. Want, A. Hopper, and J.Gibbons "The Active Badge Location System", In ACM Transactions on Information Systems 10, 1992, 91-102

[WJH97] A. Ward, A. Jones, A. Hopper "A new location technique for the active office", IEEE Personal Communications 4(5), 1997, 42-47

[WK00] S. Wilson, J. Kesselman "Java Platform Performance: Strategies and Tactics", Addison-Wesley, May 2000

[Wor95] M. F. Worboys "GIS: A Computing Perspective", Taylor & Francis, London, UK, 1995.

[WPT03] R. Want, T. Pering, D. Tennenhouse, "Comparing Autonomic and Proactive Computing", IBM Systems Journal, vol. 42, no. 1, pp. 129-135, January 2003

[XWC02] B. Xu, O. Wolfson, S. Chamberlain "Spatially Distributed Databases on Sensors", In Proceedings of the 8th ACM Symposium on Advances in Geographic Information Systems, Washington DC, USA, pp. 153-160, 2000.

[YKW+02] S. S. Yau, F. Karim, Y. Wang, B. Wang, S. K. S. Gupta "Reconfigurable Context-Sensitive Middleware for Pervasive Computing", IEEE Pervasive Computing, vol.1, no.3, pp.33-40, 2002

13. Publications contained in this thesis

Chapter 3: C. Becker, F. Dürr "On Location Models for Ubiquitous Computing", accepted for Personal and Ubiquitous Computing, Springer

Chapter 4: C. Becker, M. Bauer, J. Hähner "Usenet on the fly - supporting the locality of information in spontaneous networking environments", Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments in conjunction with ACM Conference on CSCW 2002, New Orleans/USA, 2002

Chapter 5: J. Hähner, C. Becker, and K. Rothermel "A Protocol for Data Dissemination in Frequently Partitioned Mobile Ad Hoc Networks", In Proceedings of the IEEE Symposium on Computers and Communications (ISCC 2003), Antalya/Turkey, 2003

Chapter 6: O. Lehman, M. Bauer, C. Becker, D. Nicklas "From Home to World: Supporting Context-Aware Applications through World-Models", In Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communication (PerCom 04), Orlando, USA, 2004

Chapter 7: C. Becker, G. Schiele "Application Adaption Requirements and their Support in Pervasive Computing" In Proceedings of the 3rd International Workshop on Distributed Auto-adaptive and Reconfigurable Systems, ICDCS 2003, Providence/USA, 2003

Chapter 8: C. Becker, G. Schiele, H. Gubbels, K. Rothermel "BASE - a Micro-broker-based Middleware for Pervasive Computing" In Proceedings of the IEEE International Conference on Pervasive Computing and Communication (PerCom 2003), Fort Worth/USA, 2003

Chapter 9: C. Becker, M. Handte, G. Schiele, K. Rothermel "PCOM - A Component System for Pervasive Computing", In Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communication (PerCom 04), Orlando, USA, 2004

Chapter 10: M. Handte, C. Becker, G. Schiele "Experiences: Extensibility and Flexibility in BASE" Workshop System Support for Ubiquitous Computing (UbiSys) at UbiComp, Seattle/USA, 2003