Context Awareness in Information Logistics

vorgelegt von
Diplom-Betriebswirtin (FH)
Sandra Haseloff

Von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktorin der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Erhard Konrad
Berichter:	Prof. Dr. Herbert Weber
Berichter:	Prof. Dr. Oliver Günther (HUB)

Tag der wissenschaftlichen Aussprache: 12.04.2005

Berlin 2005
D 83

Abstract

Information logistics aims to optimize information supply with regard to various dimensions according to users' demands. One of the dimensions which significantly affect information demands and their optimized fulfillment is the situation of entities or, in other words, the dimension of context. The pieces of information that are relevant to a user and the optimal way of supplying them are frequently determined by contextual factors. In addition, mobile and ubiquitous computing environments more and more complement or even replace traditional desktop computing. These environments are characterized by rapid context changes, in particular concerning the available communication media. Information logistic applications thus have to flexibly adapt to context in order to ensure an optimized information supply. A consideration of context also enables information logistic applications to become increasingly unobtrusive by reducing the amount of explicit user input they require. As a consequence, users are allowed to focus on their actual tasks rather than having to concern themselves with issues of how to interact with software applications.

Therefore, the quality of information supply and the usability of information logistic applications can significantly be improved by considering the dimension of context. The ability of software systems to adapt their behaviour and the provision of information and services to context is called context awareness. Contextual information, however, has not been taken into account in information logistics so far. The goal of this thesis is to make information logistic applications context-aware. For this purpose they are added a new component called the Context Component. This component deals with all aspects related to the representation, gathering, management, and supply of context. By this means information logistic applications are enabled to perform optimizations with regard to the dimension of context and thus to improve the quality of information supply and the degree to which users' demands are met.

This thesis presents concepts, models, and a reference architecture for the Context Component of information logistic applications. It incorporates a context model that serves as a basis for the consistent and efficient processing of context. Furthermore, techniques for a sophisticated gathering of context data from heterogeneous sensors, the augmentation, filtering, management, and supply of these data as well as their association with quality characteristics are presented. The reference architecture for the Context Component provides the various stakeholders of information logistic applications with a comprehensive guideline covering all aspects of the component's life cycle. It supports the fulfillment of both the functional and the non-functional requirements made onto the Context Component and as a result ensures that the component software is of high quality.

German Abstract

Ziel der Informationslogistik ist eine optimale Informationsversorgung von Personen, die deren individuellen Bedürfnissen entspricht. Zu diesem Zweck berücksichtigt die Informationslogistik verschiedene Dimensionen. Eine Dimension, die einen großen Einfluss auf Informationsbedarfe und ihre optimale Befriedigung hat, ist die Situation von Entitäten oder, mit anderen Worten, die Dimension Kontext. Welche Informationen für einen Benutzer relevant sind und wie sie ihm optimal zugestellt werden können, ist oftmals von kontextuellen Faktoren abhängig. Überdies wird die traditionelle Nutzung von Computern immer stärker durch mobile und ubiquitäre IT-Umgebungen ergänzt oder von diesen sogar abgelöst. In solchen Umgebungen finden Kontextwechsel, insbesondere hinsichtlich der verfügbaren Kommunikationsmedien, sehr häufig statt. Informationslogistische Anwendungen müssen sich daher flexibel an unterschiedliche Kontexte anpassen, um eine optimale Informationsversorgung zu gewährleisten. Die Berücksichtigung der Dimension Kontext ermöglicht es informationslogistischen Anwendungen weiterhin, nutzerfreundlicher und leichter bedienbar zu sein, da weniger explizite Eingaben vom Benutzer benötigt werden. Dadurch können sich Benutzer stärker auf ihre eigentlichen Aufgaben konzentrieren, anstatt sich um die Bedienung von Softwareanwendungen kümmern zu müssen.

Die Qualität der Informationsversorgung und die Bedienbarkeit informationslogistischer Anwendungen werden daher durch die Berücksichtigung der Dimension Kontext deutlich erhöht. Die Fähigkeit von Softwaresystemen, ihr Verhalten und die Versorgung ihrer Benutzer mit Informationen und Diensten an den jeweiligen Kontext anzupassen, wird als Context Awareness bezeichnet. Kontextinformationen wurden jedoch bisher in der Informationslogistik nicht berücksichtigt. Das Ziel dieser Dissertation ist es, informationslogistische Anwendungen kontextsensitiv zu machen. Zu diesem Zweck werden sie um eine neue Komponente, die Context Component, erweitert. Diese Komponente behandelt alle Aspekte der Abbildung, Erkennung, Verwaltung und Bereitstellung von Kontextinformationen. Durch sie sind informationslogistische Anwendungen in der Lage, Optimierungen in Bezug auf die Dimension Kontext durchzuführen, sodass die Qualität der Informationsversorgung und der Grad der Erfüllung von Nutzerbedürfnissen erheblich erhöht werden.

In der vorliegenden Dissertation werden Konzepte, Modelle und eine Referenzarchitektur für die Context Component informationslogistischer Anwendungen vorgestellt. Sie beinhaltet ein Kontextmodell, das die Grundlage für die konsistente und effiziente Verarbeitung von Kontext bildet. Darüberhinaus werden Techniken für die umfassende Kontexterkennung auf Basis heterogener Kontextsensoren, die Anreicherung, Filterung, Verwaltung und Bereitstellung dieser Daten sowie ihre Verknüpfung mit Qualitätsmerkmalen dargestellt. Die Referenzarchitektur der Context Component stellt den verschiedenen mit informationslogistischen Anwendungen befassten Interessengruppen eine Richtlinie zur Verfügung, die sich auf alle Aspekte des Lebenszyklus der Komponente erstreckt. Sie unterstützt die Erfüllung sowohl der funktionalen als auch der nichtfunktionalen Anforderungen an die Context Component und gewährleistet auf diese Weise, dass die Komponentensoftware qualitativ hochwertig ist.

Acknowledgements

I would like to express my warm gratitude to all the people who have supported me in completing this dissertation.

First of all, I am obliged to Prof. Dr. Herbert Weber, head of the Fraunhofer Institute for Software and Systems Engineering ISST, for creating a creative and stimulating environment which has made this dissertation possible. The establishment of information logistics as one of the institute's master topics provided me with an exciting research opportunity. In addition, I would like to thank Prof. Dr. Weber for his constant readiness to discuss my research and for his beneficial comments.

I am grateful to Prof. Oliver Günther, Ph.D. for immediately declaring himself willing to take over the part of the second referee and for his responsiveness.

Earlier drafts of this dissertation were reviewed by Dr. Wolfgang Deiters, Dr. Ralf-Detlef Kutsche, and Dr. Agnès Voisard whom I would like to thank for their valuable comments and advice. Dr. Wolfgang Deiters also deserves special thanks for his continuous support and constructive criticism. In the initial phase his ideas and encouragement greatly helped me find my dissertation topic.

I appreciate my colleagues at the Fraunhofer ISST and the CIS group at TU Berlin for inspiring discussions and suggestions and for organizational aid which have contributed to the success of this thesis. Furthermore, I would like to thank all the students who participated in the prototype implementation.

I am deeply grateful to my parents, Vera and Hans-Dieter Haseloff, for bestowing on me ambition and a love for learning, providing me with the opportunity to study, and – perhaps most importantly – exemplifying through their own lives how not to give up.

Making this dream come true has required many sacrifices to be made, in particular by you as well, Christian. I am tremendously thankful to you for the numerous discussions, your inspiration and ideas, your belief in me, for bearing my thin-skinnedness, and for broadening my horizon in so many ways. Thank-you for countless starry things.

While you and i have lips and voices which
are for kissing and to sing with
who cares if some oneeyed son of a bitch
invents an instrument to measure Spring with?

e. e. cummings

Table of Contents

List of Figures xi

List of Tables xiii

1 Introduction 1
- 1.1 Motivation and Objectives 1
- 1.2 Thesis Contributions 3
- 1.3 Thesis Outline 4

2 Background and Requirements 7
- 2.1 Information Logistics 7
- 2.2 Context Awareness 10
  - 2.2.1 General introduction 10
  - 2.2.2 The significance of context in information logistics 12
- 2.3 Other Related Research Areas 18
  - 2.3.1 Spatial database systems 19
  - 2.3.2 Sensor networks 20
  - 2.3.3 Software architecture 21
- 2.4 Requirements Onto Context Awareness in Information Logistics 22
  - 2.4.1 Context modelling 22
  - 2.4.2 Context gathering 25
  - 2.4.3 Architecture of the Context Component 28

3 Related Approaches 31
- 3.1 Context-Aware Infrastructures and Applications 31
  - 3.1.1 Sentient Computing 31
  - 3.1.2 Technology for Enabling Awareness (TEA) and follow-up research 32
  - 3.1.3 The Context Toolkit 34
  - 3.1.4 Coordinated Adaptation Platform 35
  - 3.1.5 Affective Computing 37
  - 3.1.6 Framework for context-aware pervasive computing applications 38
  - 3.1.7 SOLAR 40
  - 3.1.8 Context Service 41
- 3.2 Existing and Proposed Standards 42
- 3.3 Summary and Assessment 45

4 An Object Model for Context 47
4.1 An Object Model for Location 47
4.1.1 Location containers 48
4.1.2 Location structure 51
4.1.3 Location coordinates 53
4.1.4 Prepositions 58
4.1.5 Operations on locations 61
4.1.6 Interactions with the location model 65
4.1.7 Summary 67
4.2 An Object Model for State 70
4.2.1 Motion 70
4.2.2 Activity 75
4.2.3 Physical condition 78
4.2.4 Emotional condition 79
4.2.5 Summary 81
4.3 An Object Model for Reachability 84
4.3.1 Generic reachability model 85
4.3.2 Instance of the reachability model for current use 88
4.3.3 Summary 97
4.4 An Object Model for Surroundings 99
4.5 Summary and Assessment 101
5 Context Gathering Techniques 105
5.1 Sensor Adaptors 105
5.1.1 Overview of possible context sensors 106
5.1.2 Criteria for context sensor assessment 109
5.1.3 Transformation of sensor data into context data 113
5.1.4 An object model for sensor adaptor integration 117
5.2 Virtual Sensors 126
5.2.1 Context data combination 127
5.2.2 Context data pre-aggregation 127
5.2.3 Context data derivation 128
5.2.4 Definition of rules for context data combination and derivation 131
5.2.5 An object model for virtual sensor integration 138
5.3 Context Builders 148
5.3.1 Context data filtering 149

5.3.2 Context data aggregation 150
5.3.3 An object model for context builder integration 150
5.4 Summary and Assessment 152

6 Architecture of the Context Component 157
6.1 Integration of the Context Component into the Information Logistics Framework 158
6.2 Context Component Viewpoints 163
6.2.1 External viewpoint 164
6.2.2 Logical viewpoint 170
6.2.3 Dynamic viewpoint 173
6.2.4 Structural viewpoint 182
6.2.5 Physical viewpoint 187
6.2.6 Interrelation of the viewpoints 191
6.3 Summary and Assessment 192

7 Implementing Context-Aware Information Logistic Applications 195
7.1 Implementation of the Context Component 195
7.2 A Context-Aware Information Logistic Portal 196
7.2.1 Application description 197
7.2.2 Application architecture 199
7.2.3 Application design and implementation 201
7.3 Summary and Assessment 213

8 Conclusions 215

Appendix: Context SRML DTD 219

References 225

Index 245

x

List of Figures

Figure 1: Key dimensions covered by information logistics 8
Figure 2: Constituent elements of context 18
Figure 3: TEA architecture [Laer99] 33
Figure 4: Components of the Context Toolkit [Dey00] 35
Figure 5: Architecture of the Coordinated Adaptation Platform [EfCh01] 36
Figure 6: Architecture of the framework [HeIn04b] 39
Figure 7: Architecture of the Context Service [LeSo02] 41
Figure 8: Sketch of an itinerary 48
Figure 9: Location containers 49
Figure 10: Location spans and repeated locations 51
Figure 11: Location structure 52
Figure 12: Location coordinates 55
Figure 13: Classes for validity checks 56
Figure 14: Location prepositions 60
Figure 15: Operations on locations and operation execution services 65
Figure 16: Interactions with the location model 66
Figure 17: Overall location model 68
Figure 18: Subdivision of motion 71
Figure 19: Motion model 73
Figure 20: Attribute model 74
Figure 21: View of human activity after Engeström [Enge87] 76
Figure 22: Activity model 77
Figure 23: Physical condition model 79
Figure 24: Emotion model 81
Figure 25: Overall state model 82
Figure 26: Generic reachability model 86
Figure 27: User-related attributes of the reachability model 88
Figure 28: Current input facilities of devices 89
Figure 29: Exemplary representation of a display's characteristics 91
Figure 30: Overall reachability model 97
Figure 31: Object model for surroundings 100
Figure 32: Top-level structure of the context model 102
Figure 33: Context constraints 103
Figure 34: Sensor adaptors 118
Figure 35: Context element specification 120
Figure 36: Context element formats 121
Figure 37: Complete context element specification 122
Figure 38: Sensor adaptors' results 126
Figure 39: Virtual sensors 139
Figure 40: Context specification 140
Figure 41: Virtual sensor creation 141
Figure 42: Virtual sensors' results 143
Figure 43: Basic implementation of virtual sensors 144
Figure 44: Condition part of the virtual sensor implementation 145

Figure 45: Action part of the virtual sensor implementation 147
Figure 46: Context builder and context gatherer 151
Figure 47: Hierarchy of context gathering components 152
Figure 48: Overall model for context gathering structures 154
Figure 49: Elements of information logistic applications 158
Figure 50: Subsystems and components of the information logistics framework 159
Figure 51: Context Component architecture's viewpoints 164
Figure 52: Interface `ContextComponent` 166
Figure 53: Context Component's results 167
Figure 54: Representation of contexts' and context elements' validity 168
Figure 55: Interface `ContextMetadata` 169
Figure 56: Communication protocol-specific interfaces of the Context Component 170
Figure 57: Context Component front-end 172
Figure 58: Dependency interface for the Context Component 172
Figure 59: Context Component back-end 173
Figure 60: Life cycle of the Context Component 174
Figure 61: Message exchange within the Context Component 176
Figure 62: Interactions with the participation of sensor adaptors 178
Figure 63: Interactions with the participation of virtual sensors 180
Figure 64: Interactions with the participation of context builders 181
Figure 65: Message exchange for the provision of metadata 182
Figure 66: Context Component packages and their relationships 184
Figure 67: Context Component modules and their relationships 185
Figure 68: Clients' relationships with the Context Component 186
Figure 69: Physical configuration for medium-sized applications 189
Figure 70: Physical configuration for large-scale applications 190
Figure 71: Modified user interface of the KXS 198
Figure 72: Architecture of the context-aware portal 200
Figure 73: Definition of contexts in the Context Editor 203
Figure 74: Components for the integration of the RFID location sensor 207
Figure 75: Components for the integration of the schedules-based sensor 209
Figure 76: Components for the integration of the KXS-based sensor 211

List of Tables

Table 1: Combinations of basic and temporary motion 72
Table 2: Sketch of an exemplary mapping table for an indoor location sensor 115
Table 3: Characteristics of the external viewpoint 165
Table 4: Characteristics of the logical viewpoint 170
Table 5: Characteristics of the dynamic viewpoint 173
Table 6: Characteristics of the structural viewpoint 183
Table 7: Characteristics of the physical viewpoint 187

1 Introduction

1.1 Motivation and Objectives

Modern information and communication technology has opened up the opportunity of having unlimited access to information at any time and place. However, information overload threatens to vitiate the benefits of a ubiquitous availability of information. In view of this situation the research area of information logistics has been established. Information logistics aims at optimizing information supply to individuals by ensuring that people receive exactly those pieces of information they require in the manner that best suits their needs. As a result, a just-in-time information supply oriented towards people's demands is achieved. The main key to putting this concept into practice is software technology. Software applications that follow the paradigm of information logistics are called information logistic applications. Information logistic applications optimize information supply with regard to various dimensions. They are modular software systems consisting of several components. Each component is responsible for the execution of specific functional tasks related to the optimization of information supply with regard to a particular dimension. In addition, specialized components that coordinate the applications' overall execution and the optimization of information supply across all dimensions as well as basic services required by every component exist.

Up to now information logistics has taken into account the dimensions of content, time, communication, and, to some extent, location. This includes the assessment, selection, and retrieval of content according to users' demands, its transformation into suitable formats, and its timely delivery to the place it is required at. Accordingly, various research topics have been addressed such as content rating and filtering, document transformation, management and representation of information demand, time and prognosis models, etc. The dimension of location, too, has significant impact on both the content that is supplied to users and the way information supply is carried out. Information demands may arise at particular locations only, for instance when a person wishes to be notified about traffic jams she is heading towards when driving. Location information may furthermore affect the selection of the pieces of information a person is supplied with. This is, for example, the case when people are provided with the weather forecast for the area they are currently in. However, although the importance of the dimension of location to information logistics is obvious, so far no mature concepts for the consideration of this dimension exist. In addition, we have perceived that a truly optimized information supply has to take further factors into account that up to now have not been dealt with at all. Similar to the way the location of persons or objects determines the what and how of information supply other information describing their situation and environment also have a significant influence on people's information demands. Not until aspects such as the current activity of a person, her objectives, surroundings, the communication media she has at her disposal, or even her mood, and so on are considered a comprehensive, user-oriented optimization of information supply can be carried out. Therefore, there is a need to take situational information – or, in other words, context – into account as a key dimension of information logistics. The significance of context for information supply mainly results from the fact that information demands are frequently determined by context. Consider, for example, the following cases:

- Administrative officials working on application forms want to have access to laws, guidelines, or precedents that are related to the case in front of them.
- Travelling allergic sufferers wish to be notified whenever they are close to an area with a high concentration of pollen.
- Salespersons want to be informed about the latest sales figures and negotiations concerning the respective customer they are driving to.
- During exercise people wish to be notified whenever certain bodily functions reach critical values.
- Generally, people want information to be delivered to a communication device they have immediate access to and that is most suitable for displaying and processing the information.

Numerous other scenarios can be thought of which show how a consideration of context improves the provision of people with information according to their needs or even how it makes an accurate information supply only just possible. The importance of the dimension of context is further increased by the growing trend towards mobile and ubiquitous computing. In mobile environments the context of persons requiring information, in particular the communication media available to them and these media's characteristics, may be subject to rapid changes. Optimizing information supply to mobile users involves a quick adaptation of the way information is provided to changes in the environment. In addition to this, context also contributes to a large extent to facilitating the interaction of persons with computing devices and applications. Automatically detecting and adapting to context allows applications to require significantly less user input to be fed in, thus enabling people to focus on their actual objectives rather than forcing them to concern themselves with the details of operating a computer.

These benefits a consideration of context entails show that making information logistic applications context-aware substantially improves the quality of information supply and the extent to which people's demands can be fulfilled. Consequently, this thesis is based on the need to enable information logistic applications to adapt to context and proposes to regard context as a key dimension of information logistics. As mentioned above, information logistic applications are composed of several components that each provide a coherent functionality. Thus, in order to integrate features of context awareness into information logistic applications a new component dealing with the various aspects related to the dimension of context is required. This new component is called Context Component. The principal objective of this thesis is to provide concepts, models, and a reference architecture for the Context Component of information logistic applications. This includes the modelling of context as a basis for the consistent processing of this type of information throughout information logistic applications. In addition, the component is to be responsible for the gathering and management of context as well as for providing other components of information logistic applications with context data. The reference architecture for the Context Component is intended to serve as a guideline to developers, testers, integrators, and various other stakeholders of information logistic applications. It incorporates all mechanisms related to the representation, gathering, management, and supply of context. Furthermore, the architecture is aimed at ensuring that the Context Component fulfills the requirements made onto it in a high-quality manner. As a result, a facilitated implementation, customization, testing, deployment, operation, and maintenance of the component is enabled.

1.2 Thesis Contributions

This thesis claims to provide a novel and usable approach towards integrating features of context awareness into information logistics. It identifies and describes the requirements that have to be fulfilled in order to comprehensively consider the dimension of context. Based on these requirements the thesis provides sophisticated solutions by means of which information logistic applications are enabled to optimize information supply with regard to context. In detail the contributions of the thesis are:

- An elaborate context model to enable a consistent representation and efficient processing of context throughout information logistic applications
- Techniques and programming abstractions allowing for a comprehensive gathering of context data from various heterogeneous sensors and for their management
- A reference architecture for the Context Component of information logistic applications that facilitates the component's development, operation, and maintenance and ensures a high quality of the component software
- An implementation of the Context Component and of a context-aware application using it.

The first contribution, the model for context we provide, ensures that all components of information logistic applications are enabled to make use of a uniform representation of context. The context model covers a variety of contextual information which can be captured at different degrees of complexity and precision, thus allowing to build powerful information logistic applications. Due to its formal and structured representation of context the model facilitates the processing of context within other application components, in particular the matching of contexts defined in conjunction with information demands with those determined by the Context Component. The context model incorporates a number of novel design approaches such as the representation of locations by means of different coordinate systems, the explicit consideration of prepositions, or the comprehensive modelling of attributes describing entities' state, surroundings, and reachability. Since the model is generic and extensible, it is universally applicable to any application domain, and it is able to meet different and changing requirements. The context model furthermore abstracts from the data context sensors provide and can flexibly be adapted to different environments. Therefore, the model for context we propose allows for a comprehensive representation of context and enables the development of sophisticated context-aware information logistic applications.

The second contribution of our research is the provision of techniques and programming abstractions for a powerful and efficient gathering and management of context. We identify the various tasks that have to be carried out in conjunction with context gathering and management and consider each of them in a comprehensive manner. Our context gathering techniques are based on novel programming abstractions (sensor adaptors, virtual sensors, context builders). They allow to integrate heterogeneous context sensors into the Context Component and to obtain data from them. Criteria for the assessment of context sensors are provided which facilitate the selection of context sensors that are appropriate for a specific application and which ease their integration. Furthermore, our approach allows to increase the value of the data acquired from context sensors by means of configurable mechanisms for context data transformation, combination, aggregation, derivation, and filtering. In addition, context

can be associated with information about its quality, thus taking into account the varying technical capabilities of context sensors and the different demands of the Context Component's clients. Our mechanisms for context gathering ensure that clients of the Context Component are provided with exactly those data they need in the manner they require. Due to the generality and flexibility of our techniques again universal applicability and adaptability to heterogeneous, dynamic environments are achieved.

This thesis' third contribution is a reference architecture for the Context Component. It incorporates both the context model and the context gathering techniques mentioned above. The Context Component's architecture ensures that the functional as well as the non-functional requirements onto the component are met in a high-quality manner. By addressing the concerns of various stakeholders the architecture provides a comprehensive guideline covering all aspects related to the Context Component's life cycle. It facilitates the development of context-aware information logistic applications, and at the same time it enables more sophisticated applications to be developed than those which have been available up to now. Furthermore, the architecture of the Context Component ensures that the component can smoothly be integrated both into information logistic applications and into different target environments.

Finally, in proof of the usability and validity of our solutions, the fourth contribution of our research is an implementation of the Context Component and of a context-aware information logistic application using it. This application is a context-aware portal system that depending on the current context of users offers them corresponding relevant information. Due to the complexity and scale of our concepts and the architecture of the Context Component the application makes use of only a subset of all possible features presented in this thesis. It is, however, sufficiently complex and mature to illustrate the benefits of a context-aware information supply and to prove the soundness and validity of our approach.

1.3 Thesis Outline

The remainder of this thesis is structured as follows. In Chapter 2 an introduction into the concepts of information logistics, context awareness, and other related research areas as the fundamentals this thesis is based upon is given. Furthermore, the chapter discusses the requirements emerging from the desire to make information logistic applications context-aware. These requirements are identified on the basis of this thesis' objectives as well as of an analysis of the current scope of information logistics and its present limitations with regard to the dimension of context.

Subsequently, Chapter 3 is dedicated to a review of approaches related to this thesis. The chapter both examines recent research in the area of context-aware computing as well as relevant standardization efforts. In addition, based on the abovementioned requirements the existing approaches are evaluated, and their strengths and shortcomings are pointed out.

In Chapter 4 we present a comprehensive context model that constitutes the basis for a consistent representation and processing of context in information logistic applications. The model covers the relevant context elements location, state, reachability, and surroundings and provides a generic and formal means of describing each of them. Some fundamental aspects of

our context model have already been presented by us at the Informatik 2001 conference [Hase01c] and in a book dealing with information logistics and its concepts and benefits [Hase01b].

Chapter 5 describes techniques for an efficient gathering and management of context. It is based on our presentation at the Multi-Conference on Business Information Systems 2004 [Hase04]. In this chapter the characteristics of context sensors are identified and examined. A transformation process to convert data gathered from context sensors into objects according to the context model and vice versa is described. In addition, the chapter also provides mechanisms by means of which the data obtained from context sensors can be augmented in several ways.

The reference architecture for the Context Component of information logistic applications is presented in Chapter 6. This chapter first of all describes how the Context Component fits into the overall structure of information logistic applications. After that the component's architecture is presented by means of several viewpoints each of which addresses particular concerns the stakeholders of information logistic applications have.

In Chapter 7 the implementation of the Context Component and of a context-aware information logistic application based on it is described in order to prove the soundness of our solution. The chapter explains the application's functionality and its benefits and provides details concerning its implementation and the way it makes use of the Context Component's functionality.

Finally, Chapter 8 contains a summary and conclusion of this thesis and its main contributions and suggests directions for further research.

2 Background and Requirements

In this chapter the fundamentals of this thesis' work are presented. The following sections provide an introduction into the terms and concepts of information logistics, context awareness, and some other related research areas. In addition, the requirements that have to be fulfilled to treat context as a key dimension of information logistics are identified and described.

2.1 Information Logistics

Information has become one of the key factors of production. Huge numbers of information and communication systems exist which make information available in a quick and easy manner. The way information is accessed and used has been affected immensely by various technological developments made in recent years. Innovations in the area of telecommunication and network technology such as wireless or broadband networking have enabled high-speed data transfer which also spans mobile communication. The Internet as an integration platform for various kinds of information from different origins has become widespread and constitutes an enormous source of information. In addition, new computing devices, in particular small mobile appliances, have gained increasing acceptance. The technical capabilities of these devices allow for information to be retrieved from or delivered to nearly everywhere.

As a consequence, various means of obtaining information are at people's disposal. They open up the opportunity of an unlimited access to information at any time and place. This vision, however, also entails some problems: The variety of available information and information sources often makes it difficult for people to find what they need and burdens them with the laborious task of separating the relevant from the irrelevant. In addition, people are frequently provided with information at a time it has no value to them. Therefore, information overflow threatens to foil the benefits of ubiquitous information availability. At the same time, people searching for information often do not know where the information they need can be found, receive information that does not meet their requirements or is outdated, and cannot dispose of particular pieces of information at a given point in time. This results in an information undersupply occurring simultaneously with the abovementioned information overflow.

Since in many cases information supply is hence neither purpose-oriented nor timely, simply making information systems ubiquitously accessible does not suffice. Thus, telecommunication and hardware technology alone cannot counter the existing problems. The challenge of an accurate information supply has rather to be met by solutions that provide information in line with people's demands and enable a purposeful access to information. The research area of information logistics established by the Fraunhofer ISST is targeted on meeting this challenge. Information logistics represents a new paradigm for information supply and highly increases its quality by incorporating sophisticated concepts of personalization and adaptation.

Definition 1: Information logistics

Information logistics focuses on information supply to individuals and aims to optimize this process by means of a purpose-oriented provision of information adjusted to people's demands. It ensures that only the right and effectively needed information is made available to

individuals at the right time and in the right context. This includes the transformation of information in accordance with people's preferences and the available communication media in order to make sure that the supplied information can be processed by the recipient.

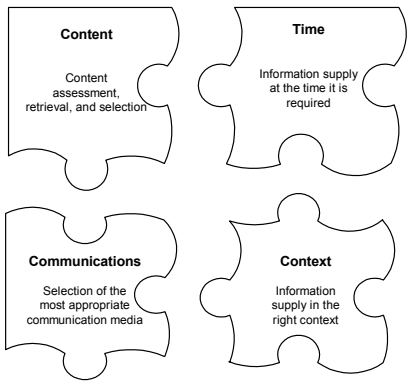

Figure 1: Key dimensions covered by information logistics

As can be seen from the definition given above – which is adopted and translated from [BuDe99] –, information logistics is targeted on optimizing information supply across various dimensions. Thus, the benefit of information logistics to individuals is always multidimensional. The following key dimensions covered by the research area of information logistics have been identified (see Figure 1):

- Content management

  Each individual has information demands that differ from those of other persons. Information logistics ensures that each person is supplied with only that content she really needs.

- Time management

  In the information society the just-in-time delivery of information is of utmost importance. Information is only useful if provided at the required time and if it is not outdated. Information logistics enables a timely supply of information that fulfills these requirements.

- Communication management

  An information logistic information supply involves various methods of providing individuals with information via several communication channels. In addition, the supplied information is adjusted to the present technical environment of the recipient, for instance the devices she has at her disposal.

- Context management

  As mentioned in the previous chapter, the context of entities had not been considered in the earlier stages of research in the field of information logistics. At first, only the dimension of location was taken into consideration. This dimension had been identified with the purpose of ensuring that people receive information at the place they need it and are not provided with information that is of no value at their current location. We have pointed out

that instead of taking into account entities' locations alone a consideration of their entire context is necessary in order to realize the full potential for optimization. Therefore, we consider context management as the fourth main dimension of information logistics.

The key to the implementation of the aims expressed in these four dimensions is software technology. By means of modern software engineering concepts an information logistic information supply can be put into practice. Software systems that follow the paradigm of information logistics are called information logistic applications.

Definition 2: Information logistic application

An information logistic application is a software system that equally takes into account the dimensions of content, time, communication, and context in order to ensure the optimization of information supply to individuals.

Several software systems already exist which consider some of the abovementioned dimensions. However, as pointed out in [BuDe99], the concept of information logistics is so far the only approach that pursues an overall optimization of information supply by comprehensively considering the entirety of these dimensions.

At the Fraunhofer ISST various information logistic applications targeted on different domains have been developed up to now, including

- Smart-Wear®, a wearable information broker that combines the principles of information logistic information supply with wearable computing technology. In [HeDe03] detailed information concerning this platform is given.
- The traffic information system W@keUp, developed in cooperation with the Gerhard-Mercator-University of Duisburg. On the basis of the past volume of traffic measured on motorways in North Rhine-Westphalia W@keUp forecasts the advisable departure times for journeys in order to ensure that drivers arrive at their destination on time.
- DONDE®, a mobile document management client allowing users to receive documents when away from the office without the need to have access to a computer [Hase01a].
- A service for content providers such as news agencies or advertisers called @ptus® news which enables them to supply customers with personalized compilations of documents.
- @ptus® weather, an application designed to promptly and directly transmit information concerning potentially dangerous weather conditions to affected persons.

Additionally, in order to facilitate and quicken the development of information logistic applications and to guarantee their unvarying high quality as regards characteristics such as flexibility, interoperability, or scalability a reference architecture for information logistic applications has been developed. This architecture – called the information logistics framework –, its scope and content are dealt with in detail in Section 6.1. The various contributions in [DeLi01] provide further details regarding the research area of information logistics as a whole as well as regarding specific aspects of information logistics, including some of the existing applications.

2.2 Context Awareness

We have identified context management as one of the principal dimensions information logistics has to be concerned with. In this section we therefore provide an introduction into the research area of context awareness and define the relevant terms related to it.

2.2.1 General introduction

In contrast to communication among humans the interaction between humans and computers is subject to various limitations. They result from the inability of computers to understand human language, to grasp the background and current situation of the human interacting with it, to conceive the world the human is part of, and so on. As a consequence, information must be provided to computers in an explicit manner and must previously be transformed. Since this procedure is error-prone, time-consuming, and reduces user acceptance, the vision of a transparent and unobtrusive interaction with computers has been developed. It is being pursued in several research areas of computer science such as ubiquitous computing [Weis91] or pervasive computing [HaMe01].

Furthermore, as mobile and ubiquitous computing environments more and more complement and even replace traditional desktop computing the variety of situations both the computer and the human using it may be in increases significantly. Information about the current environment such as the available resources and their characteristics (bandwidth, printers, information sources, etc.) or the surroundings and locations of computing devices and their users becomes essential as computers have to adapt to often rapidly changing conditions.

These problems have motivated the emergence of the research area of context-aware computing. The basic aim of context awareness is to provide computers with situational or, in other words, contextual information as a result of which they accordingly adapt the execution of tasks and the way of interacting with users. Since most of the contextual information made available to computers is to be captured automatically, the amount of information people have to input explicitly is reduced. As a result, the interaction with computer systems is facilitated and becomes more user-friendly. Humans are enabled to focus on their actual tasks and intentions rather than having to concern themselves with how to use a computer. Computer systems themselves thus become less noticeable. In addition, an adaptation to context opens up opportunities for the creation of new services, for example in the area of mobile computer use, and enhances the provision of information and services to users. Similar to information logistics context awareness, too, is targeted on improving the quality of information supply by adjusting it to entities' contexts. This adaptation also contributes to increasing the quality of the services context-aware computer systems provide. As can be seen from the motivation and objective of context-aware computing, context awareness is an essential prerequisite for putting major visions regarding future computing applications as expressed in ubiquitous or pervasive computing, for example, into practice. However, in contrast to information logistics it only takes one single dimension into account and does not consider the entirety of the dimensions relevant to information supply. Thus, as already explained in Chapter 1, context awareness is an essential feature contained in the more comprehensive approach of information logistics.

The term context-aware computing was first introduced by Schilit and Theimer who described context as location, nearby people and objects, and changes to those objects over time [ScTh94]. Since then a large number of further definitions of the terms context and context awareness has been proposed in the area of computer science. Yet, none of them has been able to become generally accepted. The vast majority of existing definitions of the term context can be grouped into two categories. Those belonging to the first category define context by example. The context definitions given by Brown et al. [BrBo97], Ryan et al. [RyPa98], and others consist of a discrete enumeration of elements that constitute context. These enumerations include location, identity, time, season, temperature, emotional state, orientation, etc. In the second category of definitions context is defined by synonyms such as environment [SaAb98], situation [HuNe97], or state [ScAi99]. Both types of context definitions suffer from considerable drawbacks. While a context definition by example is too specific and may quickly turn out to be incomplete as soon as contextual information that has not been considered in the definition becomes relevant, a definition that merely provides a synonym for the term context is too general to be applied in practice.

In the awareness of these shortcomings Dey has proposed general and widely applicable definitions of the terms context and context awareness [Dey00], [Dey01] that have been gaining a lot of acceptance. According to Dey context is defined as »*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.*« In addition to this, Dey defines context awareness as follows: »*A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*« As explained in the following section, we take these definitions as the basis for our own research.

Context awareness has been recognized as an important technology basic to modern computing applications. Therefore, much research has been done in this area in recent years, and various context-aware prototype applications exist. In Chapter 3 we examine several of these approaches in detail. At this point we restrict ourselves to giving an overview of the most prevalent application areas for context-aware systems and to referring to some representative applications of each area. Context-aware systems have been developed for domains such as:

- Tourism

  Users are provided with relevant information concerning towns, museums, or sights they visit. Context awareness is achieved by considering the current location of users, the sight they currently look at, their interests, previously visited places, and so on and by accordingly adapting the supplied information. Example applications in this area are the GUIDE [DaMi98] and the Cyberguide [LoKo96] systems or Stick-e documents [Brow96].

- Fieldwork

  Context is used in conjunction with the recording, filtering, and presentation of data concerning observations made in fieldwork environments, for instance in the areas of archaeology or animal protection, as described by [RyPa98] or [PaRy98].

- Augmented devices

  Devices, especially handheld computing devices, are equipped with one or more sensors to make them react to changes in the physical environment. These devices are used to control equipment [HaFi98]; they adapt their display to the light conditions [ScBe99] or to their tilt [Reki96], or change the appearance of applications [HiPi00] by means of the collected data.

- Resource management

  Computing devices, rooms, or buildings are made aware of their context and as a result automatically control the objects that are present. They may, for instance, detect and operate available devices such as printers or digital whiteboards. Examples of this type of context-aware applications include the Reactive Room system [CoTa95] or Satchel [FlPe00].

- Communication, collaboration, and coordination

  Context is used to automate and facilitate the communication, collaboration, and coordination among people, for example by forwarding incoming phone calls to a person's current location [WaHo92a], by determining the communication media by means of which a person can be contacted at a given point in time [TaYa01] or automatically choosing the most suitable communication channel in a given situation [DaWa97], or by providing people with awareness about group members and their current context [GrSp01].

- Information storage and retrieval

  The retrieval of information such as names and addresses or documents is facilitated by recording contextual information and attaching it to documents or events. People are enabled to query these records and can thus more easily find information, for example documents they have used in a context similar to their current one. Exemplary context-aware applications for information storage and retrieval – also referred to as reminder tools – are the Forget-Me-Not system [LaFl94], the Remembrance Agent [RhSt96], or the MemoClip [Beig00].

2.2.2 The significance of context in information logistics

In this section we clarify the meaning and scope of context in the area of information logistics. For this purpose we first develop own definitions of the terms context, situation, and context element. After that we identify and define those elements of context that are of crucial importance to information logistics and in particular to the work of this thesis.

2.2.2.1 Definition of basic terms

We have already mentioned that Dey's definitions of the terms context and context awareness have become widely accepted and have been adopted by various other researchers (e.g. [BeHa02] or [FeBe01]). Due to their universal validity we likewise use these definitions as the basis for our research. Yet, we believe that the term context still needs to be put in more concrete terms and that furthermore a definition which is mathematically underpinned is advisable. Therefore, we have developed the following definition of the term context:

Definition 3: Context

Context is any information that can be used to characterize the situation of an entity [Dey01]. It denotes a set of typed attributes which possess specific values in concrete situations. The set $C := \{a_1:T_1, a_2:T_2, ..., a_n:T_n\}$ *thus represents a context, with* a_i *being context attribute names and* T_i *being context attribute types. A context is constructed with reference to a given objective, i.e. there is a set of context attributes which are relevant to the respective objective.*

This definition adds two aspects to Dey's concept of context. Formalizing context as a set of typed attributes facilitates the handling of context in concrete applications, while specifying that context is constructed with reference to a certain objective eases the management of the possibly infinite amount of situational information context may comprise. Both Dey's and our definition of context use the term situation as an important related concept. Since Dey does not provide a definition of situation, we remedy this flaw by defining this term as follows:

Definition 4: Situation

A situation is a part of the world state at a specific point in time or within a specific time interval.

Thus, a situation is, in other words, an instantiation of a suitable choice of attributes which can be regarded as a subset of the world state. As can be seen from this definition, time plays an outstanding role in conjunction with the concept of situation as attributes – which may also be employed to represent context – are instantiated with an unambiguous reference to time.

In order to embed features of context awareness into information logistics it is necessary to identify those aspects of context that are particularly important in this field of application. We refer to such specific aspects of context as context elements.

Definition 5: Context element

A context element is a subset E of C which is constructed according to a functional logical criterion.

In the following section we consequently identify and define the context elements that are important to the research area of information logistics and to the subject of this thesis.

2.2.2.2 Identification and definition of context elements

In order to optimize information supply information logistics has to consider several context elements. In this section these elements are identified, defined, and illustrated by some examples. In addition, a distinction is made between context elements belonging to the scope of the Context Component and those dealt with in other parts of information logistic applications. The context elements discussed below represent those data that cover the demand for context awareness in information logistics in general. Depending on the scope and purpose of a specific information logistic application only a subset of these elements may be required or some additions may be necessary. For the application scenarios and prototype implementations that have been developed so far, however, these elements are sufficient and are assumed to be so in most information logistic applications.

Location

As already pointed out, the location of an entity is an important aspect of its context. In conjunction with information supply it is particularly relevant for location-dependent information demands [SaSc01]. These demands may only just occur with the presence or absence of an entity at one or more locations. This is, for example, the case when an alert is to be set off when a hired car enters a foreign country without permission or when an employee wishes to be sent her schedule for the day as soon as she arrives at the office. In addition to this, the requested information itself may depend on location. An example of this is the request for information about traffic jams ahead of a person driving in a car.

Definition 6: Location

A location is a coherent construct with or without expansion in space described by coordinates. We distinguish between atomic and non-atomic locations. Non-atomic locations contain one or more other locations; in atomic locations no other locations are contained.

In contrast to other conceptions of location – for example in geometry defining location as the whole of all points possessing a required property [Koec97] – the definition of location in the context of this work enables information logistics to regard any type of information that answers the question of where an entity is as a location. Thus, there is no restriction to the type of location information that is processable by information logistic applications.

Examples of locations according to our definition are »Berlin«, »Aunt Emmy's«, »Room 1.29«, »car«, »way to work«, »W - 137o45'.23.649; S +25o12'.47.099«, or even »car tyre«. Among these »Room 1.29«, for example, is a non-atomic location if it contains other locations that can be detected by the available context sensors and that are relevant to the application such as the furniture in the room. It is atomic if there are no other entities that can or need to be located within it. Atomicity therefore is not an unchangeable property of a location, but it varies according to the available sensors and the presence of mobile people or objects.

Definition 7: Coordinates

A location's coordinates are an element of a coordinate system. Each singular coordinate is a component of this element.

To give an example the coordinates specifying the position of the location »Room 1.29« consist of the room number only. There may be additional coordinates such as a room description, a room type, the measurements of the room, and so on. These coordinates identify the room uniquely within the associate coordinate system for rooms.

Definition 8: Coordinate system

A coordinate system is a vector space spanned over a domain. It has an origin which is the superordinate non-atomic location or the system boundary.

Accordingly, there are various types of coordinate systems in an information logistic application. There may also be several instances of any type of coordinate system. The domain a coordinate system is spanned over consists of continuous or discrete values from various

dimensions. These dimensions correspond to coordinate types; the values are valid coordinate values. They may be specified by an enumeration in case of discrete values or by setting a valid value range with suitable constraints.

An example of a coordinate system is a room coordinate system with the dimensions number, type, description, and measurements and with valid values for these dimensions. Its origin may be the non-atomic location »Fraunhofer ISST building« containing the rooms of the coordinate system. It may also be the system boundary – that is, void – if superordinate locations are irrelevant to the application and there can be no ambiguities.

State

The state of an entity represents that part of context that is directly related to the mind and body of the entity itself. The relevance of information to an individual is highly determined by what the person is doing and/or what physical, motional, and emotional state she is in.

Definition 9: State

An entity's state denotes information about its mental and/or physical situation.

Since the mental and/or physical situation of persons or objects is a highly complex area related to several other disciplines such as psychology, it is beyond the objective of this thesis to provide a comprehensive model for the context element of state. Instead, we will concentrate on those of its aspects that are of major relevance to information logistics. These aspects are motion, activity, physical condition, and – with regard to humans – emotional condition. Some simple examples of states are »conferring«, »on business trip«, »angry«, or »sick«.

Reachability

Reachability is a context element that plays a major role in conjunction with the execution of information supply. In order to provide users with information an information logistic application needs to know which communication media are at the users' disposal and can be used for communication.

Definition 10: Reachability

Reachability denotes the sum of all communication media a person has at her disposal and is able to use at a given point in time.

Since information logistics aims at information supply to individuals, the term reachability only refers to the communication media available to a person. Although objects may as well contain communication media, the reachability of an object, for example a car, only becomes relevant to information logistic applications when there are persons in or near the object that may use its communication facilities.

Surroundings

Another aspect of context that influences information supply is the surroundings of an entity. The surroundings may both affect the way information is presented to users as well as the selection of information that is to be supplied.

Definition 11: Surroundings

The surroundings of an entity are one or more pieces of information about the environment of the entity. Each of these pieces of information is a single datum that refers to a particular environmental condition.

Examples of surroundings are data about the light conditions, temperature, noise level, or concentration of carbon dioxide in an entity's environment. Applications utilized in environmental studies, for example, may trigger an alarm when some indicator data exceed a certain threshold. Or consider an electrician who is presented technical instructions in a different way – say, via speech output – when working in a humid and dark tube than when working in an ordinary building where a display is more convenient to use.

Other elements of context

This subsection gives an overview of further context elements. Since for reasons which are explained below these elements do not belong to the scope of the Context Component, we illustrate them briefly without providing a definition of each of them.

- Identity

  Knowledge of the identity of relevant entities as mentioned in [BrBo97] and [RyPa98] is essential to any context-aware application. This not only includes the identity of the entity a context refers to, but also the identities of people or objects contained in this context such as nearby persons or available equipment. It is obvious that without this information an information supply that is adjusted to entities' contexts is not feasible as no relation between contexts and the entities known to the application can be established.

- Nearby people and objects

  People and objects near an entity also belong to this entity's context. Brown has pointed out that this context element is not restricted to actually present entities; it may also include imaginary companions [Brow98]. We regard communication media as elements of reachability and thus do not count them among nearby objects. Yet, the presence of entities other than communication media certainly may affect information supply. Consider, for example, a confidential message that is not displayed on a user's screen when other persons are near.

- Time and history

  Another prominent context element is date and time as, for example, mentioned in [Dey98] or [RyPa98]. This element provides information that answers the question of when a particular situation occurs. It is available on every computer. In addition to this, historical data about interactions between entities such as a history of past information supplies or of an entity's system usage are elements of context as well. To give a simple example knowledge of historical data can prevent an application from supplying a person with the same piece of information more than once which contributes to stemming information overflow.

- Personal preferences

  The preferences, interests, beliefs, and other attitudes of an application's users also belong to these users' context and may have a great influence on how information supply is to be

carried out and what information is to be provided. These aspects which we have subsumed under the heading »personal preferences« may on the one hand serve as a means to select the content users are to be provided with. In addition, they may also affect the communication medium employed for information supply, the delivery time, the number of deliveries, and other parameters concerning the execution of information supply.

- Information demand

  Closely related to the personal preferences of users is their information demand which is a most vital factor for information logistics. A user's information demand defines what information she wishes to be supplied with and what conditions the information has to fulfill. These conditions may refer to various aspects of information such as its type, age, quality, subject, or contents. Consider, for example, a user who wishes to be notified whenever a new document referring to a particular project is available on her company's Intranet.

- Available content

  The context of an entity furthermore comprises the available content, its information value, age, and the like. This also includes the authors or submission dates of documents [Klem00], the available information bases or view schemas of databases [ThAn02], and similar characteristics. Evidently, the available content and metadata describing it not only affect which information can be provided to users at all, but also serve as a means to adapt the provision of information to people's demands.

- Further context elements

  Numerous other aspects that may constitute an entity's context have been identified. The social situation of an entity [ScAd94], the knowledge it possesses, its objectives or focus of attention, or things like communication costs, application data such as the current text selection [PaMi97] as well as changes to all of these elements are some examples of further context elements we have not mentioned so far. A classification of contexts, aimed at the application area of mobile distributed databases, can be found in [Wate96]. Since a situation is defined as a part of the world state, the information characterizing it is virtually infinite, and no complete enumeration of all possible context elements can be given. Therefore, all other elements of context that have not been explicitly dealt with before are subsumed in this passage. As a result, our examination of context takes the abundance of possible context elements as well as aspects of entities' situations that may prove to be of particular importance in the future into account.

In information logistic applications the last-mentioned context elements are taken into consideration in specific application components [Sand01] other than the Context Component which is the subject of this thesis. The reference architecture for information logistic applications commits these applications to a specific structure. This structure consists of components each of which reflects a particular functional task an information logistic application has to perform. It defines that identity and other attributes of entities such as their preferences and attitudes are managed by a Profile Manager. A Subscription Manager deals with all aspects of information demand. The points of time and time intervals related to information demand are managed by the Timer component [ScLe01], and a specific History service allows for the storage and retrieval of data about past incidents. For the assessment, selection, and retrieval of

content several content services are employed which are managed by a Content Broker. Section 6.1 gives further details about the structure of information logistic applications and the responsibilities of the components they consist of.

As can be seen from these explanations, the elements of context dealt with in this subsection are subject to intensive research in other areas of information logistics. The goal of this thesis, however, is to provide solutions for a component that manages those pieces of contextual information which have not been considered so far. As a result, the context elements of identity, nearby people and objects, time and history, personal preferences, information demand, available content, and others are outside the scope of this thesis and its objectives. Instead, we focus on those aspects of context not addressed by other research areas of information logistics, in particular those mentioned in the previous subsections.

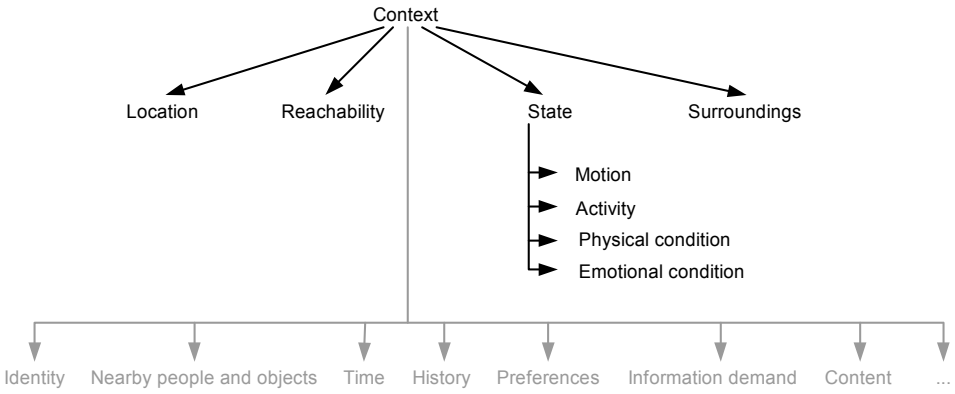

Figure 2: Constituent elements of context

To summarize and illustrate at a glance the explanations given in this section Figure 2 charts the context elements we have identified. Those aspects of context that are an essential topic of this thesis are depicted in black colour, whereas those that are being investigated in other research areas within information logistics are typed in gray. Although the figure may give the impression that the elements of context form a tree structure, this is not the case. Context elements may be interdependent in the sense that particular values of one context element's attribute determine the possible attribute values of another element. Consider, for instance, a mineworker who is located in a shaft far below the ground. This person is not reachable via her cellular phone while at this location. Thus, context elements are not fully orthogonal. We address the interdependence among context elements in Chapter 4 and Chapter 5 when presenting our context model and context gathering techniques, respectively.

2.3 Other Related Research Areas

This thesis is based upon and carries on ideas from several other domains of computer science. In this section spatial databases, sensor networks, and software architecture as particularly relevant research areas our work is related to are briefly introduced.

2.3.1 Spatial database systems

Since location is an important aspect of context, the dimension of space plays a major role in conjunction with context awareness. Many aspects related to the representation and management of spatial information have already been studied in the area of spatial databases. A spatial database system (SDBMS) is a database system that offers spatial data types in its data model and query language and supports these types in its implementation by providing at least spatial indexing and efficient algorithms for spatial join [Guet94]. SDBMSs are characterized by the ability to manage large collections of geographic objects. Modelling, querying, data structures and algorithms for the implementation of SDBMSs, and appropriate system architectures are the fundamental problems addressed by this research area. SDBMSs deal with geographic objects each of which consists of two components, a spatial object describing the location, shape etc. of the object in space and nonspatial or descriptive attributes [RiSc01].

The main approaches to geographic space modelling can be divided into entity-based and field-based models [RiSc01]. In entity-based models such as the realms introduced by Güting and Schneider [GuSc93] geographic objects or entities are the primary objects. They are composed of an identity, a spatial object, and a common description. The main abstractions used to represent spatial objects are points, polylines, and polygons. Field-based models, in contrast, view space as a continuous domain. Each point in space is assigned one or more attribute values by means of continuous functions. Concerning the modelling of collections of spatial objects various models such as the spaghetti model, the network model, or the topological model have been developed. The main problem in the implementation of geographic space models is the transformation of the infinite point sets in Euclidean space into finite representations that are processable by computers. For this purpose different representation modes such as tessellation, vector mode, and half-plane representation are made use of [RiSc01]. In addition, several algebras as, for example, the geo-relational algebra [Guet88], an algebra for manipulating maps [ScVo89], or the ROSE algebra [GuSc95] have been proposed to capture the representation of spatial data, their relationships, and the operations on them. The most important operations of spatial algebras are those referring to spatial relationships. Spatial relationships can be classified as topological [Egen89], direction, and metric relationships [Worb92]. To achieve a general extensibility of databases in terms of user-defined spatial data types the concept of abstract data types (ADTs) is used [StRu83].

Querying in spatial databases first of all involves to connect the operations defined in spatial algebras to the features of a database system's query language. This also includes extensions made to existing query languages to enable them to handle spatial data. In addition, since an interactive use of SDBMSs requires to present input and output graphically, a graphical representation of spatial data types is needed. Furthermore, data structures for spatial data types as well as algorithms for the operations of spatial algebras are prerequisite to implement SDBMSs. In addition to atomic operations, spatial indexing and the support of spatial join are of particular importance. At the system architecture level an integrated architecture using extensible database systems has recently been gaining the most acceptance and is made use of in several SDBMSs such as Postgres [Momj01], Monet [BoQu96], or GéoSabrina [LaPa93].

There are many further issues related to spatial databases such as spatial objects with imprecise boundaries [ErSc97], spatio-temporal [KoSe03] and moving objects databases [GuBo00], or spatial constraint databases [BeBe97], to name but a few. Due to the great diversity of this research area we restrict ourselves to this brief overview and refer the reader to the references for further reading.

2.3.2 Sensor networks

The vision of pervasive and ubiquitous computing is based on the assumption that small, embedded devices autonomously sense and process a variety of physical phenomena. In the novel area of sensor networks research is being done that aims at putting this aspect of the vision into practice. Sensor networks open up powerful opportunities to sense, analyze, and manipulate the physical world. The basic idea of this research area is to aggregate a large number of sensor nodes into computational infrastructures that automatically and in real-time collect, process, and supply information about the physical world [ChKu03]. Sensor networks promise to enable a new quality of applications and services in many domains such as environment monitoring, traffic surveillance, industrial automation, and so on.

While the functions of its individual nodes are relatively simple, a sensor network's complex functionality is achieved by combining a vast number of sensor nodes in a dynamic system. Concerning both individual sensor nodes and sensor networks as a whole various conceptual and technical challenges exist [EsCu02] that require to fundamentally re-examine traditional approaches to distributed systems and hardware technology. Tiny, energy-efficient, durable, and robust sensor nodes have to be developed. On the part of the networks again energy efficiency plays an important role and needs to be carefully considered by the protocols and algorithms used. Since nodes may be added to or removed from the network during operation and may even be mobile, the dynamics of sensor networks are extremely high. They have to be addressed by techniques for spontaneous, ad-hoc networking. In addition, fault-tolerance and reliability are required to ensure a continuous operation of the network in spite of errors or failures of individual nodes. Due to the small size of sensor nodes and their deployment in large and possibly inaccessible areas sensor networks have to autonomously configure and administer themselves. Many networks furthermore have to meet special demands concerning security or real-time operation.

Another important problem in the area of sensor networks is collaborative signal and information processing. Because of energy constraints sensed data are typically aggregated and fused locally on sensor nodes. These nodes produce and communicate relevant, semantically enriched data of a higher level of abstraction. With respect to other aspects such as naming, routing, or communication protocols, too, new solutions are required to handle the specific characteristics of sensor networks. The objective of a generic sensor network infrastructure in addition calls for concepts and architectures regarding the programming and operation of sensor networks. The networks have to be dynamically (re-)programmable and must suit different usage scenarios. They furthermore have to be integrated into exisiting infrastructures or the Internet. In connection with sensor networks the dimensions of space and time gain particular importance, for example concerning the localization of sensor nodes, the synchronization between them, or the sampling scale.

Many of these challenges are currently being addressed by various areas within computer science. The University of Berkeley's SmartDust prototype [WaLa01], for example, represents a first step towards extremely small, networked sensor nodes. The TinyOS [HiSz00] and TinyDB approaches [HeHo03] provide an operating system for sensor networks and an infrastructure based on database technology, respectively. In other areas such as routing [InGo03], sensor collaboration [ZhSh02], or the management of sensor data [Mani03] relevant results have also been gained recently. Sensor networks thus promise to revolutionize the way physical data are collected and processed. In Chapter 5 we address some of the core challenges of this area and provide solutions for them in conjunction with our approach to context gathering.

2.3.3 Software architecture

The architecture of software systems has proven to be a factor of critical significance to their success. Past experience in software engineering has shown that software systems which lack an architectural design usually suffer from a number of shortcomings that considerably affect their quality and cost [Webe92]. An explicit consideration of architectural issues therefore is an essential part of the software engineering process, leading to a practice called architecture-based development [BaKa99].

Nearly all approaches concerned with software architecture give their primary attention to the structural issues of a software system which, according to Garlan and Shaw, include »*gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives*« [GaSh93]. Although its roots go far back [Dijk68], up to now no standard, universally accepted definition of software architecture exists. Like in the research area of context awareness a variety of definitions differing from each other to a greater or lesser extent have been proposed (e.g. [BaCl03], [OMG03], [IEEE00], [ShGa96], or [PeWo92]). Nevertheless, it is widely agreed upon the fact that a software architecture describes the structure of an application, the behaviour of the application's components, their relationships, and the interfaces they provide.

A software architecture is based on the functional and non-functional requirements made onto an application and is designed with the aim to meet these requirements. If for particular application areas reference requirements can be identified, it is possible to design a reference architecture for these types of applications. Reference architectures represent patterns for the design of applications of a specific area. Their benefit is a reuse of entire architectural concepts on the design level. Information logistics is an application area the specific requirements of which have suggested the design of a reference architecture. Accordingly, the information logistics framework has been defined which is dealt with in greater detail in Section 6.1.

Generally, various different stakeholders of a software system exist each of which has specific concerns and looks at the system's architecture from a particular perspective. Moreover, an architecture is concerned with several different aspects of software. Communicating about a particular part of a software architecture thus is difficult as long as it is not clear to which of its aspects the discussion refers. Therefore, is has proven reasonable to describe software architecture by means of several different views. Each view addresses a specific set of stakeholders' concerns and thus provides a representation of a software system from a particular perspec-

tive. In doing so, it is possible to make use of different languages, modelling techniques, and notations for each view. In Chapter 6 we come back to the concept of architectural views and discuss it in greater detail in connection with the architecture of the Context Component.

2.4 Requirements Onto Context Awareness in Information Logistics

This section identifies and describes a set of requirements that have to be fulfilled in order to integrate features of context awareness into information logistics. These requirements are based on the objectives of this thesis as well as on an analysis of the current scope of information logistics and its limitations with regard to the dimension of context.

The requirements onto context awareness in information logistics can be grouped into three main categories. First of all, there is a need for a context model covering all relevant context elements identified above. This model has to be applied throughout information logistic applications in order to ensure the consistent processing of context data. Thus, we identify requirements related to context modelling as the first basic category of requirements our approach has to fulfill. Furthermore, an adaptation of information supply to context can only be carried out if the contexts of entities are known. It is therefore obvious that information logistic applications need means of determining context. We refer to this task as context gathering and classify the requirements related to it as belonging to the second category. The functional tasks related to context awareness have to be encapsulated in the Context Component of information logistic applications. In order to facilitate the development of this component, to increase the quality of the component software, and to ensure its smooth operation and evolution an elaborate software architecture for the Context Component is required. The third category of requirements therefore contains issues related to the Context Component's architecture.

2.4.1 Context modelling

As mentioned before, there is a fundamental need for a generally applicable context model which ensures the processability and consistency of context throughout information logistic applications. Furthermore, this model contributes to a common understanding of context among all stakeholders of information logistic applications. From the superordinate requirement for the existence of a universal context model several more fine-grained demands with respect to this model can be derived. This section considers these individual requirements.

Consideration of the variety of contextual information

Evidently, a context model for information logistics has to cover all of the relevant context elements identified in Section 2.2.2.2. The attributes these context elements consist of, their types and possible values as well as the relationships among them have to be identified and described. The context model needs to be extensible with respect to the set of context elements it covers and the attributes these elements possess. Both contexts provided by sensors and those specified in demand profiles may furthermore vary strongly in complexity. Therefore, the context model is required to support different degrees of complexity and precision.

Contextual information may in addition be incorrect, incomplete, or inconsistent. As a consequence, other approaches have emphasized that a context model needs to represent the quality characteristics of context data [Heln02]. However, although the quality of context clearly needs to be taken into account, the divergence of context quality results from the specific characteristics and the imperfectness of context sensors. Therefore, we do not consider quality attributes as an integral part of context itself that is within the scope of the context model. As a dependant of context sensors context quality is rather addressed in conjunction with the requirements onto context gathering described later on.

Abstraction

Context data are captured by means of various sensors. The formats of the data provided by context sensors may differ from each other to a great extent. In addition, each user of an information logistic application may also specify contexts in an individual way. Gearing a context model to the specific representation of one or more select sensors or users imposes severe restrictions on applications. The ability of such a model to represent the entire existing variety of contextual information, to keep up with the evolution of an application, or to flexibly adapt to different technical environments, to name but a few desired characteristics, is limited by the missing universal validity of such an approach. As a result, our context model must abstract from specific context representations employed by particular context sensors or persons.

Enabling structured and consistent processing

In order to determine whether an information supply is to be carried out information logistic applications have to compare the contexts contained in demand profiles with those reported by context sensors. Such a comparison can only be made if context is represented in a structured way. A uniform structure ensures that all parts of information logistic applications share a common understanding of the syntax and semantics of context, its elements, attributes, and values. This understanding is an essential prerequisite for the processing of context. In addition, it must be possible to query contexts in order to access those of their elements and attributes that are relevant in given circumstances. This again presupposes that contextual information is meaningfully structured. A structured representation furthermore is required for the augmentation of context data obtained from sensors as explained later on.

Universal applicability

As already pointed out, the concept of information logistics is not tied to a specific application domain, but rather aims at providing a universally applicable solution that can flexibly be adapted to specific requirements and conditions. Consequently, a context model for information logistics is required to provide a generic representation of context. Changes in the application domain and environment of an information logistic application and in the requirements of users should have minimal impact on the context model.

Support of a human-readable representation

The definition of information demands may be carried out by information logistic applications or by users themselves. In the latter case users interact with the context model to specify contextual conditions onto information supply. To ensure the processability of context throughout

information logistic applications suiting the model to the way users conceive and express context is not advisable. On the other hand, however, a context representation that is tailor-made for an optimized electronic processing of context is unsuited for users for the most part. As a consequence, a context model for information logistics has to support the representation of context in a human-readable format. Since the definition of information demands is usually carried out by means of graphical user interfaces, contexts should in particular be convertible into a graphic representation. Moreover, specific aspects of the context model's complexity need to be hidden from users in order to make the model convenient to use.

Support of context comparisons

We have already mentioned that in order to satisfy context-dependent information demands information logistic applications have to compare the contexts defined in users' demand profiles with those reported by context sensors. The required comparisons may vary strongly in complexity. Simple checks merely involve determining whether the values of a particular context attribute are equal or examining if a sensed value is within a certain range. More complex calculations have to be made when several context attributes and their values need to be compared or when simple comparisons of two values for equality do not suffice. This is, for example, the case when a transformation of locations' coordinates or a computation of the distance between two locations have to be performed before the actual comparison can be carried out. As a consequence, a context model for information logistics should support the straightforward and efficient comparison of contextual information. In addition, it has to make algorithms and operations available which facilitate context comparisons and aid those parts of information logistic applications using the model in the execution of tasks commonly needed in conjunction with context comparisons as, for example, transformations of locations.

Interoperability

Within the scope of other projects dealing with context awareness some models for context have already been proposed. Moreover, several standardization efforts, in particular concerning the representation of location, are being made. Chapter 3 presents a detailed examination of these approaches and points our why none of the existing models can be adopted in information logistics. However, a number of services has been developed on the basis of these context models which may also be required in or useful to information logistic applications. The Open GIS Consortium, Inc., for example, has specified a Gazetteer Service [AtFi02] which transforms symbolic into geometric locations. As mentioned above, such transformations of locations as well as other operations supporting the comparison of contexts have to be provided by the context model for information logistics, too. In order to make use of external services the context model for information logistics should be interoperable with other models for context and in particular with proposed standards. Thus, the context model should support a conversion of contextual information into a different model's representation and vice versa.

Support of privacy and security

In context-aware computing applications a high amount of contextual information about entities is gathered and maintained. Since much of this information may refer to very personal and confidential aspects of people's lives – such as their current activities or moods, for example –, most people do not want this information to be entirely revealed to any other person. As a

consequence, issues of privacy and security are of tremendous importance to context-aware computing. Dealing with these issues, however, is beyond the scope of this thesis, because privacy and security in context-aware computing is a research topic on its own. Yet, in view of the importance of privacy protection in context-aware applications, a context model for information logistics should provide a fundamental support for privacy and security. The context model should allow for a selective addressing of and access to dedicated pieces of contextual information in order to enable the definition and administration of security policies.

Performance and scalability

In context-aware information logistic applications a large amount and variety of contextual information is gathered and processed. Data have to be obtained from various sources and have to be converted into an abstract representation. Contextual information furthermore serves as a basis for different kinds of computations, including comparisons and transformations of context. In addition, the number of users an information logistic application has to serve may also vary from a few dozens to several thousands. Therefore, a context model for information logistics has to scale to unforeseeable, possibly very large numbers of users. Furthermore, it has to ensure that the chosen representation of context allows to input, access, and process context with adequate performance without imposing delay.

2.4.2 Context gathering

The second category of requirements onto context awareness in information logistics refers to the task of context gathering. Information logistic applications need to be aware of entities' contexts which as a result have to be captured by means of various context sensors. In this section the detailed requirements onto the task of context gathering are identified and described.

Integration of heterogeneous context sensors

The Context Component must acquire context data from appropriate sensors. In contrast to many other approaches that suffer from being restricted to the use of one or a few types of sensors only our goal is to enable an interaction with any possible type of context sensors. We have already pointed out that the heterogeneity of context sensors is immense. This includes aspects such as the format of the data they provide, their interfaces, mode of operation, coverage, and limitations as well as economic criteria such as costs, etc. An integration of arbitrary context sensors into the Context Component thus requires a diligent consideration of their heterogeneous characteristics and capabilities. The way a context sensor is accessed and data are obtained from it has to be adapted to the sensor's specific characteristics.

Providing easy access to sensor data

The data gathered from context sensors are made use of both within the Context Component itself and in other parts of information logistic applications. Due to the heterogeneity of context sensors a variety of rather complex tasks has to be carried out to access the data they provide. If each entity that needs to process context was responsible for obtaining the relevant data from context sensors itself, application software would become unmanageable and would be almost impossible to maintain. In addition, many application parts would be required to perform tasks that are beyond their actual purpose.

Therefore, it is obvious that the details of interacting with context sensors must be encapsulated within the Context Component and hidden from other application parts. The modules responsible for obtaining sensor data have to provide a uniform interface to other entities by means of which these data can easily be accessed. This interface must furthermore allow clients to make specifications concerning the required data, for example with regard to their quality, costs, or complexity. The requirement for an easy access to context data also includes to provide clients with context in a representation they are able to understand and process. This means that the data obtained from context sensors have to be transformed into a uniform representation – i.e. into the representation of context described by the context model for information logistics – before they are made available to clients.

Augmentation of sensor data

In most cases a single context sensor provides only a subset of the variety of contextual information that is relevant to a specific application. In addition, each sensor is usually subject to some limitations with respect to characteristics such as scope or accuracy. A significant increase in the expressiveness, complexity, and quality of context data can thus be achieved by augmenting the data obtained from context sensors in several ways.

First, more than one sensor may provide data concerning the context or individual context attributes of an entity. In addition, information demands may contain contextual conditions which cannot be detected by one single sensor. It is therefore necessary to aggregate the data provided by several context sensors. Furthermore, context sensors may provide data with reference to different entities or types of entities. The data one context sensor supplies may correspond to the entities a different sensor's data refer to. In such cases a combination of these data has to take place in order to gain contextual information that could not be detected if the context sensors were treated independently of one another. Combining the context data provided by an infra-red location sensor that tracks persons with the data obtained from temperature sensors installed at locations covered by infra-red beacons, for instance, allows to gather contextual information about the surroundings of the tracked persons. Another important mechanism to augment sensor data is the derivation of context. By means of context derivation additional pieces of contextual information can be gathered without the existence of a dedicated sensor providing this information. Deriving context means that the data obtained from context sensors serve as a basis for inferences about context attributes that differ from those the underlying data refer to. By means of derivation processes interdependences among context elements can be taken into account.

Since context data can be acquired from several context sensors and can in addition originate from aggregation, combination, and derivation processes, the individual pieces of information that describe the context of an entity may contradict themselves. Moreover, due to the variety of sources that provide context duplicate pieces of the same contextual information may be generated. Therefore, an augmentation of sensor data also requires to include filtering mechanisms by means of which duplicate context data are eliminated and contradictions in gathered contexts are resolved. The filtering of context data also prevents invalid combinations of context attribute values in a context. To meet application-specific requirements and conditions it should be possible to make use of different policies for context data filtering.

Another requirement related to the augmentation of sensor data is the desire to persistently store context. Storing context creates a data pool containing a history of all obtained contextual information. This history can serve to access past contexts of entities, to infer entities' habits and draw conclusions about their demands, and to predict future contexts. Moreover, since the interaction with context sensors may be subject to disconnections as context sensors may be unavailable for a certain period, a persistent storage of context enables the Context Component to serve requests for contextual information in spite of sensor unavailability.

Consideration of context data quality

As already mentioned, the sensors employed for the acquisition of context data may be limited with respect to their accuracy, precision, scope, reliability, or other characteristics. These limitations affect the quality of context data in several ways. Since not all pieces of contextual information that are required at a given point in time may be detectable due to restrictions in the scale of context sensors or due to temporary disconnections, gathered context data may be incomplete. Besides, the data a context sensor provides are frequently afflicted with a certain degree of uncertainty. Their accuracy, precision, and reliability may vary over time and may also differ from the quality characteristics of other sensors' data. In addition, context data are highly dynamic and may quickly become outdated. Therefore, there is a risk of the data obtained from context sensors to be incorrect. Another aspect reducing the quality of gathered context data is inconsistency as data provided by several sensors or acquired during augmentation processes may contradict themselves. This aspect, however, has already been addressed in the previous paragraph.

An accurate processing of context data requires their quality to be taken into explicit consideration. All entities which process contextual information have to be provided with its quality characteristics in order to accordingly adapt the way they handle context. A major requirement onto context gathering therefore is the determination, representation, and supply of data concerning the quality of sensed context. Since the relevant quality characteristics may be application-specific, context data quality has to be represented in a generic and extensible manner.

Dynamic discovery of context sensors

We have pointed out above that the specific characteristics of context sensors must be dealt with in dedicated modules and hidden from other application parts. In addition to this, the access to sensor data should be facilitated even further by means of a dynamic discovery mechanism. Such a mechanism allows entities that require gathered context data to specify which information they need and which attributes this information is to possess. It thereupon finds suitable context sensors – or, more precisely, modules encapsulating them – that are able to supply the requested data and provides its clients with access to them. A dynamic discovery mechanism furthermore ensures that clients are only supplied with modules that are available at the moment they are needed. This allows applications to dynamically switch the modules employed for context gathering.

In addition, a dynamic discovery mechanism can be used to obtain data concerning the capabilities of the context sensors that are available in an application. Since the mechanism manages all modules responsible for the interaction with sensors, it can be queried for all

detectable pieces of contextual information along with their attributes and possible values, including associated attributes such as quality characteristics. This information is essential to other application parts to prevent contexts which cannot be detected in a particular application from being specified. Therefore, the context gathering process has to include a mechanism for the dynamic discovery of the modules interacting with context sensors.

Universal applicability

In each information logistic application a different set of context sensors is likely to be used. The mechanisms employed to augment the data acquired from these sensors as well as the relevant quality characteristics of context data may also vary to a large extent. The context gathering process therefore needs to take the variability of different applications' requirements and of technical environments into consideration. It has to be possible to smoothly add new sensors to an application and to flexibly react to changes in existing sensors with minimal impact on the context gathering process as a whole. In the same manner, augmentation processes and quality characteristics of context data have to be flexible and extensible with regard to application-specific logic or attributes, respectively.

Performance and scalability

Various context sensors have to be integrated into the Context Component, data have to be obtained from them, and these data have to be augmented and supplied to clients in an efficient manner. Furthermore, the dynamic discovery mechanism responsible for finding context sensors and supplying clients with access to them needs to be implemented using efficient algorithms. Due to the dynamics of context data the context gathering process needs to be sufficiently performant to ensure that the data it makes available are up-to-date. Just like information logistic applications in general it has to scale to a potentially large number of users and additionally to a large number of possibly complex context sensors, augmentation processes, and quality attributes.

2.4.3 Architecture of the Context Component

The architecture of the Context Component is intended to enable a straightforward development, use, and evolution of this component. The way this architecture is designed thus affects the entire life cycle of information logistic applications. In detail the Context Component's architecture has to fulfill the individual requirements presented below.

Fulfillment of functional and non-functional requirements

A fundamental requirement onto the architecture of the Context Component is that is has to support the fulfillment of both the functional and the non-functional requirements made onto the component. The Context Component's architecture thus has to be designed to ensure the component's ability to represent, gather, manage, and supply context and context metadata. Accordingly, the Context Component needs to provide corresponding interfaces that make these functionalities available to clients. Moreover, the architecture of the Context Component has to enable queries for particular context elements or attributes in addition to requests for the contexts of entities as a whole. The Context Component's clients may also have special demands concerning the quality of context data, the points in time contexts are to refer to,

and the complexity of contextual information. It must therefore furthermore be possible for clients to make specifications regarding these attributes as well. In addition, the Context Component's clients in some cases need to know the context of an entity with respect to a given point in time and in other cases wish to be notified whenever an entity's context fulfills certain criteria concerning its attributes and values, quality characteristics, and the like. The architecture of the Context Component therefore needs to provide mechanisms that allow the component to be queried both synchronously and asynchronously.

The Context Component's architecture also has to enable the component to meet requirements concerning non-functional properties. In particular it is to ensure a flexible adaptation to a variety of contextual information and context sensors in arbitrary application domains. Since in many cases the Context Component has to provide application-specific functionality, its architecture must be extensible in order to meet changing demands and be employed in diverse environments. These requirements have to be fulfilled with simultaneous consideration of the component software's maintainability; furthermore, it must be ensured that the Context Component provides the desired functionality in a robust and reliable manner. Finally, the Context Component's architecture is required to support the efficient execution of the component's tasks and to make sure that it is able to scale to potentially large numbers of users.

Identification and structuring of computational elements

In order to use the Context Component the component's clients need to know which functionality it is able to supply and how this functionality can be accessed. As mentioned above, the clients of the Context Component therefore have to be provided with interfaces by means of which they are enabled to use the component's services. In addition to the interfaces themselves, a detailed description of them along with the methods they contain, these methods' purpose and parameters, and the associated data structures has to be made available.

Moreover, there are several stakeholders – such as developers and integrators, for example – who have to possess an in-depth understanding of how the Context Component's functionality is implemented. These stakeholders need a description of all computational elements the component is composed of along with the structure of these elements, their relationships to each other and to external elements, and their interfaces. For this reason the architecture of the Context Component has to identify the computational elements required to produce the component's functionality and has to arrange these elements in a clear structure. A well-ordered component structure is a prerequisite for a high-quality component software. It substantially facilitates development and maintenance and thus contributes to ensuring that the functional requirements onto the Context Component are met. The design of the Context Component's structure furthermore has to ensure that a separation of concerns between the individual tasks the component has to carry out is achieved in order to meet non-functional requirements and to enable software reuse. The architecture of the Context Component as a result is required to define and describe the entire component software's structure.

Identification and definition of processes

Apart from the software structure which deals with the static aspects of the component's computational elements the processes these elements are involved in are of equal importance. An examination of the processes executed within the Context Component provides informa-

tion about the component's dynamic aspects. The Context Component's dynamics have a significant impact on the fulfillment of the non-functional requirements for performance, fault-tolerance, scalability, and availability. This is due to the fact that they affect to what extent processes can be executed concurrently, how tasks are scheduled, and how the overall system load is shared among processes. The architecture of the Context Component therefore needs to identify, define, and describe the processes that are executed within the component. In conjunction with this it is necessary to define the behaviour of the component's individual computational elements, the exchange of messages between them, the sequence of this message exchange, and the elements' reaction to incoming messages in terms of actions and tasks they carry out. In doing so, the Context Component's architecture has to address issues of concurrency and synchronization as well.

Transparent distribution

A further requirement onto the architecture of the Context Component is the need to support a distribution of the component software over several different nodes. As a result, the performance of the Context Component as well as its availability and scalability can be increased significantly. The component's architecture has to ensure that communication with or within the Context Component is not affected by the physical location of the component's software modules. This means that the distribution of the Context Component software is required to be transparent to both context sensors and application software. In order to fulfill this requirement the Context Component's architecture should provide a set of configurations which describe how software modules can be mapped onto nodes in different usage scenarios as, for example, in large-scale applications with a great number of users and pieces of contextual information or in small or medium-sized applications. The Context Component's software modules should in addition be able to be dynamically relocated.

Conformance with the information logistics framework

In Section 2.1 we have already mentioned that a reference architecture for information logistic applications, the information logistics framework, exists. This framework specifies the universal structure of information logistic applications and defines various mechanisms all application components require, for example regarding the configuration of components or their life cycle. Since the Context Component is to become a new core component of information logistic applications, it is obvious that it has to conform to the information logistics framework. Consequently, the architecture of the Context Component needs to ensure the interoperability of the Context Component with other components and services of information logistic applications. Moreover, generally binding mechanisms specified by the information logistics framework have to be supported by the Context Component as well.

The requirements onto context awareness in information logistics that have been identified and described in this chapter serve as a basis for the evaluation of existing approaches in the area of context-aware computing which is given in Chapter 3. They furthermore constitute the foundation of our own concepts for context modelling and context gathering and our architecture of the Context Component presented in Chapters 4, 5, and 6, respectively.

3 Related Approaches

Context awareness has been recognized as a vital feature for modern computing applications. Consequently, various context-aware infrastructures and prototype applications have already been developed. This chapter examines existing approaches in the area of context-aware computing. Due to the large amount of ongoing work in this field of research a complete description and evaluation of all proposed context-aware systems cannot be given in this thesis and besides would lack focus. Therefore, we restrict our examination of related approaches to select infrastructures and applications that are particularly relevant to our own work because of their prominence and outstanding features. We also describe approaches which have been introduced just recently and thus have not been dealt with in detail in the literature so far. Further overviews of existing context-aware computing applications can be found in [Dey00], [Mitc02], and [ChKo00]. In addition, some standardization efforts related to context awareness are currently being made as well. In particular in the area of Geographic Information Systems (GIS) a standard representation of location and of sensor capabilities is striven for. These standardization efforts are also discussed in this chapter. After the examination of related approaches Section 3.3 provides a recapitulatory evaluation of them. It assesses the approaches with regard to the requirements identified in the previous chapter and points out the shortcomings current context-aware systems and proposed standards suffer from.

3.1 Context-Aware Infrastructures and Applications

This section provides an overview of some of the most relevant existing context-aware systems and their features. It furthermore describes how these systems address the requirements onto context awareness identified in Section 2.4.

3.1.1 Sentient Computing

The Sentient Computing project at AT&T Laboratories Cambridge [AdCu01] is based on the idea of enabling computer systems to share people's perception of the environment. Sentient computing applications maintain a model of the real world which is updated on the basis of data acquired by sensors. This model is used to supply contextual information to applications. Context data are gathered by means of an indoor location sensor called Bat which operates with ultrasound, by resource monitors providing information about the status of equipment such as bandwidth or memory, and by a wireless proximity system called PICOnet [Hopp99]. Although the Sentient Computing project is targeted on sensor independence and the ability to integrate a variety of different context sensors, the existing system so far makes use of the abovementioned sensors only. Current research aims at an integration of further context sensors. As can be seen from the selection of supported sensors, the Sentient Computing project takes into account a few pieces of information concerning devices, but mainly focuses on the context element of location. Gathered location data are filtered in order to eliminate errors and are then used to update the world model. In addition, some derivations of context are also carried out; the sentient computing system determines, for example, that a person is seated by monitoring the velocity of the person's motion over a certain period of time.

In sentient computing applications each real-world entity is represented by a CORBA object which possesses a type, a name, a location, a set of properties, and an interface. Thus, in the Sentient Computing project a quite formal approach to context modelling complying with the object-oriented modelling paradigm is pursued. The CORBA objects are persistently stored in a relational database [HaHo99]. Sentient computing applications contain a spatial monitoring service which is responsible for the transformation of location data. The spatial monitor generates information about the containment and overlapping of two-dimensional locations. Furthermore, it detects location-related events and provides applications with notifications about relevant location changes. In sentient computing thus an event-driven programming style is used. In addition, a method for automatically storing generated data along with the points in time these data were created at has been developed. As a result, a timeline is constructed which contains both the generated data and some information from the world model. This timeline can be queried by means of a temporal query language.

Several applications have been developed in the course of the Sentient Computing project. They include model browsers displaying the current state of the physical world, so-called Follow-me systems that move application interfaces or route incoming phone calls to users' current locations, novel user interfaces like virtual buttons or smart posters, or applications supporting the creation, storage, and retrieval of data.

A grave drawback of the approach pursued by the Sentient Computing project is its focus on location as the only context element that is seriously taken into account. The requirement for the consideration of a variety of contextual information and of quality characteristics thus is not fulfilled by this approach. In addition, a formalization of location data in terms of containment and overlapping alone is not sufficient as it only considers part of the potential complexity of this context element. Sentient computing applications, for example, cannot recognize whether an entity is located in front of or behind a computer's monitor. Currently only a small number of context sensors is supported which primarily provide location information. In addition, the augmentation processes carried out on the basis of sensor data restrict themselves to simple aggregations and derivations. A dynamic sensor discovery mechanism has not been described in project-related publications. Since sentient computing systems rely on an event-based communication with applications, a synchronous query of context data is either not possible at all or – in case of queries made to the timeline-based data storage – returns previously generated data such as photos or documents that are associated with some contextual information which, however, is of minor importance in this type of usage scenarios.

3.1.2 Technology for Enabling Awareness (TEA) and follow-up research

The TEA project was a joint research of Starlab, Belgium, Omega Generation, Italy, Nokia Mobile Phones, Finland, and the German University of Karlsruhe's Telecooperation Office (TecO). In September 2000 the project ended, but the research begun within its scope is still being continued, in particular by the TecO. The main objective of the TEA project and its successors has been to augment mobile devices and everyday artefacts with context awareness. The projects are targeted on resource constraint platforms and primarily consider context elements that cannot be inferred from location. For the purpose of context gathering various simple, low-cost, and widely available sensors are integrated into mobile devices. Thus, the

projects address so-called direct context [GeSc02], i.e. context that is gathered and processed by mobile devices themselves rather than by a specialized infrastructure. One major goal of the projects is to develop dedicated hardware and to implement features of context awareness within it, thereby creating an application-independent add-on component to existing devices and artefacts. Due to the integration of diverse sensors and the consideration of context data of a high degree of abstraction such as people's activities, for instance, issues concerning the aggregation of sensor data and the derivation of context are explicitly addressed.

Within the scope of the TEA project an architecture for context gathering and for adapting to context has been developed [Schm03]. This architecture which is shown in Figure 3 consists of four layers, each providing a specific functionality. On the lowest layer data are gathered from a set of heterogeneous sensors. Above this layer there is a cue layer consisting of several cues each of which abstracts from the output of a single sensor and represents a feature extracted from the sensor's data stream. The cue layer provides a uniform interface to upper layers and thus hides the details of the context sensors employed from them. Above this layer an aggregation of several sensors' data is carried out on the context layer. In addition, this layer transforms sensor data into context which is considered as a function of the available cues. Finally, the top layer makes use of context by adapting an application's or a device's behaviour accordingly.

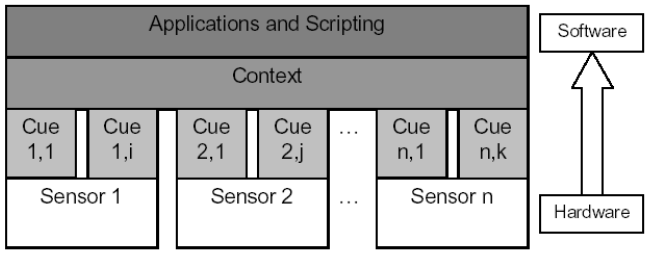

Figure 3: TEA architecture [Laer99]

During the course of the TEA project two prototype hardware boards have been developed. They are equipped with several sensors including accelerometers, light sensors, microphones, CO sensors, pressure and temperature sensors, etc. In addition, an application called the Context-Call system [ScTa00] has been implemented which demonstrates the usage of context in the area of telephony. In this application the context of callees is acquired by the abovementioned sensors and is reported to callers who thereupon decide whether a connection is to be established. Later research resulted in the development of the Mediacup [GeBe99], a coffee mug augmented with hardware and software for context gathering, processing, and dissemination, and of some applications based on this artefact as well as of Smart-Its, small embedded devices that are augmented in a similar way [HoMa01].

Since the TEA project and its successors are mainly concerned with tasks related to context gathering, the modelling of context plays a secondary role in these projects. Yet, Schmidt et al. describe a working model for context [ScBe99] which consists of an enumeration of hierarchically organized so-called features such as user, task, infrastructure, location, and so on. At the

top level context is subdivided into human factors and features related to the physical environment. Schmidt furthermore provides a specification language for context [Schm99] based on Brown's approach [Brow96], [BrBo97], a Standard Generalized Markup Language (SGML)-based description language. Schmidt's context description represents context attributes and their values, both referred to as contextual variables, by XML tags. Contextual variables can be grouped with matching attributes and are furthermore accompanied by actions that are triggered if the variables are true.

The focus of the TEA project and its successors on small mobile devices and everyday artefacts and their goal of implementing context gathering and context adaptivity in hardware require a restriction to simple computations. As a consequence, only a small amount of context can be determined and supplied in a particular device or application. There is no generic approach to representing the existing variety of contextual information in an application-independent manner. The proposed context model lacks formality and universal applicability and forces users to manually define context profiles for each type of application and each desired action anew. The context model furthermore mingles the representation of context with its use, thus failing to clearly separate concerns which makes the model insufficiently adaptable and reusable. In addition, the quality of context information is hardly considered as it is only possible to annotate contexts with an indication of their probability.

3.1.3 The Context Toolkit

The Context Toolkit, a programming toolkit for context-aware applications, has been developed by the Future Computing Environments Group at the Georgia Institute of Technology [SaDe99], [DeAb01], [Dey00]. The Context Toolkit's principal target is to provide a reusable solution for context handling that facilitates the implementation and deployment of interactive context-aware applications. The toolkit incorporates various services related to the gathering and supply of context, including an encapsulation of context sensors, access to context data, context storage, and a distributed infrastructure. These services are made available on the basis of three main abstractions, so-called widgets, interpreters, and aggregators. The concept of widgets is adopted from Graphical User Interfaces. As illustrated in Figure 4, widgets are responsible for the interaction with context sensors. They mediate between users and the environment by encapsulating information about a single piece of context. Widgets have state and behaviour and provide a uniform interface to other components or applications by means of which both synchronous and asynchronous queries can be made. Widgets furthermore incorporate services which are used to change the state of the environment using actuators. Another fundamental abstraction is that of aggregators. Aggregators can be considered as meta widgets which act as gateways between applications and widgets. They aggregate contextual information referring to real-world entities. Finally, interpreters abstract context data gathered from sensors into higher-level contextual information, thus carrying out context data derivation. In addition, the Context Toolkit contains a discoverer which is responsible for the dynamic discovery of appropriate components. These components are able to run independently and in a distributed manner and can be used by several applications. In order to support transparent distribution the Context Toolkit relies on a common communication mechanism based on HTTP and XML. The toolkit furthermore comprises a design process for building context-aware applications.

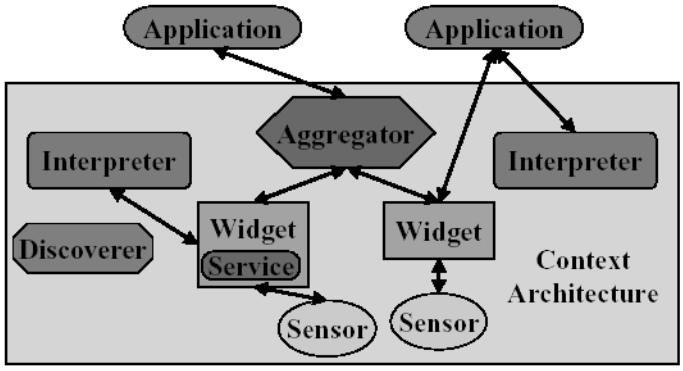

Figure 4: Components of the Context Toolkit [Dey00]

Another abstraction provided by the Context Toolkit is the so-called situation abstraction [Dey01]. It resides above the aforementioned components of the toolkit. Since a situation is understood as a collection of entities' states, the situation abstraction allows application designers to interact with the infrastructure as a whole instead of querying and subscribing to several widgets, interpreters, or aggregators individually. However, the support for this abstraction is still limited.

Exemplary applications implemented on the basis of the Context Toolkit include an In/Out Board which displays contextual information referring to the presence of people in a building and the DUMMBO Meeting Board, an existing system which has later been augmented with context awareness. The DUMMBO Meeting Board is a whiteboard supporting the capture of spontaneous meetings and the access to captured data. In addition, the Conference Assistant [DeSa99] assists people in performing various tasks when attending a conference as, for example, taking notes or deciding which activities to attend.

While the Context Toolkit addresses many requirements related to context gathering and supply, its main flaw is the absence of a formal model for context. In the toolkit context is represented by a set of context widgets' attributes. These attributes have been defined in an ad hoc manner without a formal structure specifying a common terminology for context. The fact that the representation of context relies upon the attributes of widgets furthermore limits the toolkit's ability to separate the description of context from the context gathering process. Besides, although the Context Toolkit does provide mechanisms for context derivation by means of interpreters, the functionality of interpreters is very limited as they are usually employed for simple data type conversions only. As a result, the support of comparisons between detected context and context required by an application is limited as well. In addition, the Context Toolkit does not address the quality characteristics of gathered context.

3.1.4 Coordinated Adaptation Platform

The Coordinated Adaptation Platform developed at Lancaster University is an infrastructure platform for adaptive context-aware applications. Its main objective is to enable the system-wide optimization of multiple and possibly conflicting adaptations to context at application

level. In order to achieve this goal the platform comprises an extensible set of attributes that may be used to initiate adaptation, mechanisms for the control and coordination of adaptive applications, system-wide adaptation policies, modules to make users aware of adaptation processes and to enable interactions between the platform and the user, and mechanisms for distributed operation. The architecture of the Coordinated Adaptation Platform [EfFr02b] provides two basic functionalities: The discovery and control of services that offer contextual information and the coordination of applications' adaptive behaviour which is triggered by context changes and carried out according to user-defined policies. The existing literature concerning the Coordinated Adaptation Platform contains varying explanations related to the components it consists of which suggests that the design of the platform has not been completed so far. In [EfCh01] the most detailed description of the platform's architecture is given, subdividing it into modules for context discovery and access, a context database, an application database, context agents, and an adaptation control module as shown in Figure 5.

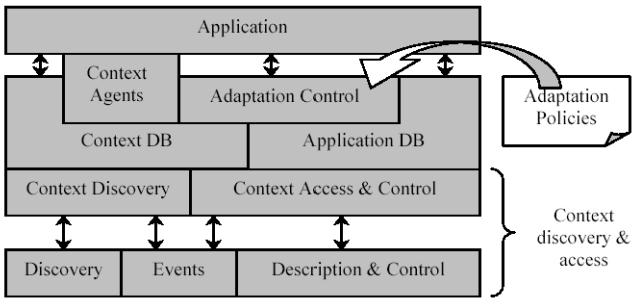

Figure 5: Architecture of the Coordinated Adaptation Platform [EfCh01]

The context discovery and access modules locate and interact with context sensors. They assume that sensors advertise themselves and provide an XML description of their capabilities and methods. The context database is a registry of all available context sensors which are classified according to the type of contextual information they provide. This part of the architecture serves to hide the mechanisms required to retrieve context from higher layers and enables a switch of context sensors. Similarly, the application database serves as a repository for the adaptation mechanisms of all applications running on the platform. Information concerning these mechanisms is also provided in XML and includes the type of context initiating an adaptation process. Context agents are pluggable pieces of programme code that carry out application-specific manipulations of contextual information such as context data aggregation. Finally, the adaptation control module monitors the status of adaptation mechanisms and decides upon the overall behaviour of the platform and the applications. The decisions made by this module are based on adaptation policies which are defined by users by means of a special policy language [EfFr02a]. Considerable parts of the Coordinated Adaptation Platform are based on the Universal Plug and Play Architecture (UPnP). The communication among the platform's components involves an exchange of XML messages by means of the HTTP protocol.

The main focus of the Coordinated Adaptation Platform clearly lies on providing support for the coordination of multiple adaptations across several applications. Although the platform is targeted on meeting the key requirements onto context-aware adaptive applications, issues of

context modelling and context gathering are addressed to a small extent only. Accordingly, a formal model for context has not been developed up to now. Context is rather represented by tags contained in the XML descriptions of sensors and adaptation mechanisms, thus being tightly coupled with these two types of entities. The individual elements of context are defined in an ad hoc manner and appear to include resource availability, time, and location only. The explanations concerning the context gathering process provided by the project are also scanty and basically restrict themselves to the discovery, description, and aggregation mechanisms illustrated above. In particular the augmentation of context data is dealt with only marginally in the Coordinated Adaptation Platform. It remains to be seen whether the further development of the platform addresses aspects of context awareness to a greater extent.

3.1.5 Affective Computing

The Affective Computing Research Group of the MIT Media Laboratory aims at enabling computers to sense, recognize, and adapt to human emotions. This ability is to improve the interaction of humans with computers as well as the interaction among humans themselves. Within the scope of the Affective Computing project several issues related to this field of research are being addressed, including the sensing, recognition, understanding, and synthesis of human emotion. In addition, applications, interfaces, communication mechanisms, and wearable computers that are aware of and respond to emotions are being investigated and developed [Pica97], [PiKl02].

Since human emotion cannot be sensed directly – except in the case of explicit self-reports provided by persons –, mainly information concerning the physical state and behaviour of people is gathered from a number of biomechanical sensors such as galvanic skin response or respiration sensors. These sensors are usually either attached to people or are incorporated into objects such as computer mice, telephones, or clothes. The recognition of emotion involves a mapping of sensor signals to emotional states. Human emotions are classified into a set of discrete states which possess specific characteristics [PiVy01]. In addition, possible transitions between these states along with the probabilities of each transition are identified. An Affective Understanding module is responsible for the usage, processing, and storage of people's emotional states. It constructs and maintains a model of persons' emotional lives at different levels of granularity. This model is used to predict emotional states and to build and maintain a taxonomy of people's preferences. Eventually, the Affective Understanding module is also to model and process the entire context of people. The Affective Computing project furthermore comprises research in the area of synthesizing emotions in machines.

The emotional states of people serve as an input to computing applications and interfaces that adapt their behaviour, appearance, and the way they communicate to the emotions of users. In the SmartCar, for example, the stress of drivers is detected [HePi00], [Heal00]. The Affective Learning Companion is an agent that senses emotional states like boredom or engagement in learning situations and accordingly adjusts its response to users [KoRe01]. Furthermore, Affective Tangibles, physical objects that can be grasped, squeezed, or otherwise manipulated, and Affective Touchables which sense affective parameters while being held or touched and communicate the emotions by means of sight, sound, or haptic changes, have been developed.

Although the usefulness of incorporating a wider variety of contextual information is stressed within the context of Affective Computing, current research is focused on sensing the physical state and behaviour of people and deriving their emotional condition. So far thus, the Affective Computing research area takes into account the context elements of emotional condition and, to a lesser extent, of physical condition and activity with reference to persons only. As a consequence, various solutions developed with the project's primary concern in mind are not suitable for approaches aimed at achieving comprehensive context awareness. The model for context employed is restricted to human emotion which is represented by a set of discrete states. This approach is not reasonable for other elements of context that require a more sophisticated consideration of the variety of their attributes and values, complexity, and quality characteristics and may possess continuous attribute values. In addition, in Affective Computing context gathering is restricted to the use of physiological and behavioural sensors. Since only a limited number of such sensors is available, no approach to integrating arbitrary context sensors is being pursued. Thus, with respect to both context modelling and context gathering, the Affective Computing approach is not sufficiently generic and extensible to be applied to a comprehensive solution that meets the requirements onto context awareness altogether.

3.1.6 Framework for context-aware pervasive computing applications

A recent research activity is the approach towards an infrastructure for context-aware pervasive computing environments pursued by Henricksen et al. at the University of Queensland and the Distributed Systems Technology Centre. To facilitate the development of context-aware applications this infrastructure aims at providing generally required functionalities such as context gathering, context management, and context dissemination [HeIn02].

Within the scope of this research a context model has been developed that addresses the diversity of contextual information and its quality as well as complex relationships among context data and temporal aspects. The context modelling concept used is based on the Object-Role Modeling (ORM) approach and on extensions to it Henricksen et al. have made [HeIn03]. Physical or conceptual objects such as persons or devices are represented by entity types. Entity types play certain roles in fact types which can be regarded as associations between entities. Henricksen et al. categorize fact types according to their persistence and source. Fact types are classified as either static or dynamic, and dynamic fact types are again subdivided into the categories sensed, derived, and profiled types. In addition, a special temporal fact type is defined to represent start times and end times of facts. Dependencies between facts as well as quality characteristics are also taken into account in the model. Fact dependencies are represented by a special type of relationship between facts, while quality is captured by allowing facts to be associated with quality indicators [HeIn04a]. The model is accompanied by a mapping process that transforms the context model into data processable by the context management infrastructure. This mapping process results in the storage of context in a relational database and includes different procedures for the mapping of the individual fact types. Derived fact types are, for instance, mapped to database views. Furthermore, a so-called situation abstraction that is based on a novel form of predicate logic is intended to facilitate the implementation of context-aware applications [HeIn04b].

The framework's architecture is organized into a layered hierarchy of interacting components as shown in Figure 6. On the context gathering layer context data are acquired from sensors, interpreted, and fused. These data are translated by the context reception layer which is also responsible for routing queries from the management layer to underlying components. On the context management layer a set of context models as well as relational representations of their instantiations are maintained. Above this layer the query layer provides an interface for both synchronous and asynchronous queries to applications and the adaptation layer. The latter manages repositories of situations, preferences, and triggers and is responsible for evaluating them. Finally, the application layer provides toolkit support for branching and triggering, two programming models used to define context-based flows of application logic.

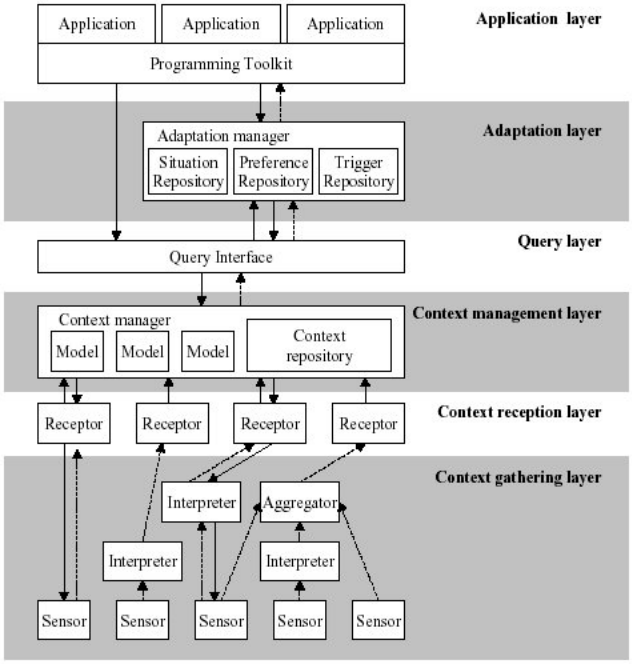

Figure 6: Architecture of the framework [HeIn04b]

Henricksen et al.'s ORM-based context model contains some simplifications that make a comprehensive representation of complex contextual information difficult. While the relationships between entities have been taken into account in detail, the question remains how entities' attributes and their values are modelled. Besides, the classification of fact types is partly based on assumptions that do not do justice to the nature of context. A person is, for example, assumed to carry out only a single activity at any given point in time. In conjunction with context gathering no elaborate augmentation processes have been described in detail so far. The proposed mechanisms are not adequate as context data filtering can only be carried out if the data that are to be filtered refer to the same context element, and derivation processes are based on database views which are not sufficiently powerful for this purpose.

3.1.7 SOLAR

A research group at Dartmouth College has been designing and developing the SOLAR system, a software infrastructure that supports the collection, processing, and dissemination of context [ChKo02a], [ChKo02b]. The approach also addresses issues of scalability as well as of security and privacy of contextual information [MiKo02]. SOLAR's central contribution is an abstraction for context gathering and supply called the operator graph [ChKo01]. This abstraction regards context sensors as information sources which generate events. A sequence of events is referred to as an event stream. Operators are objects that subscribe to and process one or more event streams received from sources or other operators and publish another event stream as an outcome. Thus, operators can be connected recursively to form a directed acyclic graph, the operator graph. The nodes contained in an operator graph are subdivided into three types: Sources, operators, and applications. Sources are wrappers for context sensors, while operators are functions of input events they receive. The sinks of operator graphs are applications which subscribe to one or more event streams and receive and adapt to incoming events. SOLAR furthermore distinguishes between four categories of operators according to the type of functionality they provide. Filters output subsets of the events they receive, transformers convert context data, mergers simply output every incoming event, and aggregators are responsible for supplying events related to a particular type of context data. The modular structure of operators thus allows for a flexible operator composition as well as for distribution and reuse.

As mentioned above, contextual information is represented as events in the SOLAR system. Each event is structured as a set of tag-value pairs. To receive contextual information applications either directly subscribe to context sensors or instantiate and subscribe to an operator graph that gathers and augments context data. Since SOLAR is targeted on supporting a variety of different applications in ubiquitous computing environments, applications are given the ability to define and interconnect operator objects themselves. SOLAR comprises a specification language by means of which applications specify operator graphs they wish to make use of. This language furthermore is employed for the advertisement and discovery of resources in SOLAR's context-sensitive resource discovery mechanism. In conjunction with resource discovery the issue of operator naming has also been addressed in detail [ChKo03].

The SOLAR system consists of several interconnected so-called Planets which run on separate nodes in a distributed network. Planets serve as an execution platform for SOLAR objects such as operators or proxies. Clients may connect and submit requests to any planet which thereupon parses these requests, creates appropriate operator instances according to the specification received from the client, and maintains a list of subscribers and a queue of input events. On the basis of SOLAR several applications have been developed such as the SmartReminder [Math01] or a meeting detector [WaCh04]. This application uses context to transfer incoming phone calls to voice-mail when a user is in a meeting. Meetings are detected by means of the pressure and motion effected on chairs.

The SOLAR system provides a promising approach towards context gathering that allows for a flexible and extensible combination of diverse augmentation processes. Yet, the proposed solutions concerning context gathering and supply are accompanied by a very simplified and informal representation of context. Like other approaches SOLAR lacks a formal model for context that allows to capture its variety and complexity. In addition, the quality of context data is

not addressed by this approach. Since applications are furthermore responsible for defining operator graphs themselves, a separation of concerns between context gathering and its use is not achieved. Applications are required to concern themselves with the context gathering process as they have to know which sensors and operators are available and have to make rather complex requests in order to be supplied with the contextual information they need.

3.1.8 Context Service

Within the scope of the Context Service project (formerly named Owl) of the IBM T. J. Watson Research Center a middleware infrastructure aimed at gathering and disseminating context in pervasive computing environments has been developed [EbHu01]. The project pursues a service-based approach to context awareness that provides applications with contextual information at a high level of abstraction. The Context Service gathers entities' context from various sensors, maintains context, and can be queried for it by clients both synchronously and asynchronously. Further issues such as context quality, context storage, extensibility, or privacy and security concerns are also addressed by this project. In the Context Service a form metaphor is used to model context [LeSo02]. A form describes a particular type of contextual information and is composed of a set of context attributes. A form may, for example, consist of the current user activity, contact means, and location. In addition, forms are associated with data concerning the quality characteristics of the context represented by them. Currently the quality metrics freshness and confidence are being used. Clients make queries to the Context Service by partially filling out form templates, i.e. by specifying the relevant fields and the requested quality of the data. The set of form types is extensible to take data heterogeneity into account.

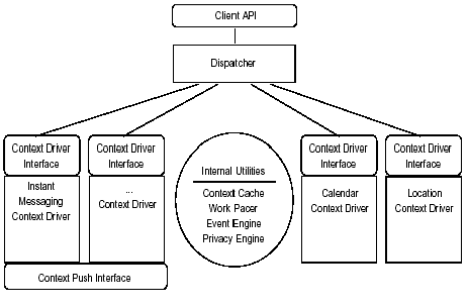

Figure 7: Architecture of the Context Service [LeSo02]

The architecture of the Context Service shown in Figure 7 consists of a dispatcher, a configurable collection of context drivers, and four utility components. Moreover, three programmatic interfaces are provided, a Client API, a Context Push Interface, and an internal Context Driver Interface. The dispatcher uses the Context Driver Interface to forward clients' requests to suitable context drivers. Context drivers are components that can be plugged into the infrastructure. They handle a particular type of contextual information by interacting with context sensors that provide this information. Context drivers may either pull data from sensors or receive updated information from them via the Context Push Interface. The utility components, a context cache, a work pacer that schedules the pulling of data from context sensors,

an event engine, and a privacy engine, can be used by context drivers. Furthermore, context drivers may obtain context from other drivers via the Client API which serves to allow context data or quality metrics to be filtered, combined, and aggregated. Other mechanisms to aggregate data from multiple sources have also been investigated and described within the scope of the Context Service project [CoPu02], [CoLe02]. Yet, it is still unclear whether and how they are to be integrated into the Context Service's architecture. In contrast to many other approaches, including our own, the Context Service project explicitly addresses privacy and security issues. The service includes a Privacy Engine responsible for controlling access to contextual information. It serves to specify, store, and retrieve privacy policies. The privacy protection mechanism employed by the Privacy Engine is based on Role Based Access Control (RBAC).

Two example applications have been developed on top of the Context Service. The first one, a Notification Dispatcher, uses context in the form of instant messaging online status and calendar events. It routes messages to the receiver's communication device that is considered most appropriate based on her current context and on the urgency of messages. The second application is called Pervasive Content Distribution System (PCD). The PCD uses predicted context or, more precisely, the predicted locations and tasks of users to forecast users' access to web content. It then preprocesses and predistributes content in order to reduce access latency.

Although the Context Service tackles several advanced issues related to context awareness, the concepts proposed so far lack maturity in some respects. The form abstraction used to represent context evidently does not distinguish between different context elements which makes it difficult to handle the potential complexity of context. Since details concerning the structure of forms' attributes have not been published so far, it is not yet clear to what extent a formal modelling of context is aspired. The existing examples, however, suggest that forms have been designed in an ad hoc manner geared to the context sensors employed, resulting in a decrease in extensibility. In addition, the mechanisms clients can make use of to specify the context they need are limited to a specification of types of contextual information and of quality metrics. The Context Service's architecture considers processes of context data augmentation only marginally. Neither context data derivation nor a filtering of context data are carried out. The latter is not desired as clients are intentionally supplied with the sum of all gathered context data along with aggregated quality metrics. To some extent this contradicts the service's goal of relieving applications from the need to concern themselves with the management of context.

3.2 Existing and Proposed Standards

At the early stages of mobile and context-aware computing location was the only context element that was taken into account in detail. Much work was aimed at the provision of location-based services which have become commonly available by now. In addition, location information has also been an essential topic in the area of geomatics and in Geographic Information Systems. Thus, a particularly great variety of location models has been developed, each expressing location information in an individual format. As a consequence, interoperability problems have arisen, and the need for open and commonly accepted standards has become evident. Several organizations and committees have addressed this problem by developing and proposing standards aimed at ensuring interoperability between location-aware services and applications. This section provides an overview of the most prominent standardization efforts.

First of all, the 3rd Generation Partnership Project (3GPP) has standardized the representation of location that is to be used within GSM and UMTS networks and by subscriber applications. In the corresponding standard called Geographical Area Description [3GPP03] location is expressed as latitude and longitude according to the WGS 84 (World Geodetic System 1984) [DrRi98] reference system. The standard also covers geometric shapes, namely ellipsoid points, ellipsoid arcs, and polygons. Ellipsoid points may be associated with information concerning their uncertainty and their altitude. 3GPP's location model additionally allows to represent entities' horizontal and vertical velocity. The standard furthermore defines the coding of location information and the message format by means of which this information is exchanged.

The Location Working Group of the Open Mobile Alliance (OMA) is targeted at providing an end-to-end architectural framework for mobile location-based services. It addresses issues such as interoperability, privacy and security, billing, and roaming. The Location Interoperability Forum (LIF) as well as the Location Drafting Committee of the Wap Forum have become absorbed in this group, and the specifications developed by them are to be adopted. In particular the LIF had developed a Mobile Location Protocol Specification [LIF02] defining how applications can access location information from a wireless network over the Internet. Like 3GPP's location model this specification allows location to be expressed as various shapes and additionally with reference to different geographic coordinate systems. The Mobile Location Protocol stipulates that any implementation must at least support the WGS 84 system. Furthermore, information concerning the horizontal speed of entities, their direction of movement, and the quality of location information is taken into consideration.

The International Organization for Standardization (ISO) has been developing a family of standards related to digital geographic information. This standard, named ISO/TC 211 Geographic Information/Geomatics, consists of more than 20 separate specifications which address the acquisition, management, analysis, access, presentation, and transfer of geographic information among different systems, users, and locations. Among these specifications both a model for location as well as standards related to the acquisition of location information can be found. According to the model developed by the ISO [ISO00] location is described by means of geometric or topological objects. Location information may refer to different coordinate systems and includes data concerning size, shape, and orientation. In addition, the standard defines a taxonomy of spatial operators which serve to use, query, create, delete, and manipulate location information. Several services for coordinate transformation and other types of conversions as well as positioning services are defined and described [ISO02]. The Positioning services standard [ISO03] specifies an interface that provides both general and technology-specific data structures used to access various location sensors.

A variety of specifications referring to the representation and processing of geographic information has been developed by the Open GIS Consortium, Inc. (OGC). These specifications are embedded in a reference model [Buch03] which aims at providing an architectural framework for the results of the OGC's ongoing work. The OGC provides both an abstract specification that serves as a conceptual foundation and reference model for implementations as well as several implementation specifications containing technical details relevant to software engineers. One of these implementation specifications refers to the Geography Markup Language (GML) [CoDa03], an XML encoding for geographic information. The specification regards locations as geographic features the state of which is defined by a set of properties. Properties

may be either simple data types or geometric types such as points, polygons, or collections of other geometries. Geometric types are in turn represented by either a tuple of geographic coordinates which may refer to different coordinate systems or by coordinates contained in a single character string. By allowing for the definition of application-specific schemas and geometric types GML is an extensible language. Although it is possible to make use of properties that do not refer to geographic values, the focus of GML lies upon geometric locations. In addition, several discussion papers exist that address issues of acquiring and processing location information. The Sensor Model Language (SensorML) defines an XML schema that serves to represent the general, geometric, and observational characteristics of sensors [Bott02]. While observational characteristics refer to the physical properties a sensor is able to measure along with these data's quality, geometric properties describe the geometric and temporal characteristics of the data supplied by sensors. Thus, although geometric location information plays an important role in SensorML as well, the language is to be applied to arbitrary sensors. Further discussion papers define services to parse and mark free text messages [Lans01] or to transform symbolic locations into a geometric representation [AtFi02], [Marg01].

As another effort to achieve a common representation of location the Spatial Location BOF of the Internet Engineering Task Force (IETF) has been initiated. This group has proposed a common data set and an extensible framework for expressing location information in the Internet [KoTa01], [KoTa02]. The data set consists of mandatory and optional elements. It stipulates that location is expressed as latitude and longitude with reference to the WGS 84 system and is attached a timestamp. Optional elements include orientation, speed, accuracy, course, direction, and unspecified attributes. In addition, the Spatial Location BOF proposes an XML encoding of the common data set. The group has also addressed the representation of location that is not specified by geographic coordinates by proposing a common syntax and coding for descriptive location [TaKo01]. Basically this approach subdivides locations into three types: Civil objects such as countries, towns, rooms, etc., geo objects like rivers or tunnels, and opaque objects which serve to represent any other locations not belonging to the former two types. For civil and geo objects an enumeration of possible values for each of these types is given.

Finally, another prominent standard that is not related to location is W3C's Composite Capabilities/Preference Profiles (CC/PP) framework [NiHj00]. CC/PP serves to specify device capabilities and user preferences by means of profiles. Profiles are transmitted from client devices to content servers which accordingly adapt the delivery of content to the capabilities and preferences described in the profiles. CC/PP is based on the Resource Description Framework (RDF) designed by the W3C. A CC/PP profile contains a structured set of attributes and values for these. The framework defines a standard set of CC/PP attributes, their permissible values, and associated meanings which constitute a CC/PP vocabulary [KlRe04]. Since CC/PP is designed with an emphasis on flexibility and extensibility, different vocabularies may be defined.

As mentioned above, almost all existing and proposed standards related to context awareness deal with the context element of location only. Besides, since these standardization efforts are rooted in mobile computing and geomatics, they focus on geometric locations, i.e. on describing locations by means of geographic coordinates. In information logistic applications, however, location frequently needs to be represented in a manner that abstracts from geometry. In many domains such as office environments, healthcare, or workforce management, for example, the relevant locations are usually rooms, buildings, customers, or itineraries, and the like.

In these cases it is often more natural and efficient to express these locations in a symbolic manner instead of using geographic coordinates. This furthermore eliminates the need to transform each sensor-determined or user-defined location into geographic coordinates which often results in a significant and unnecessary processing overhead, in particular when location sensors that do not provide geometric location information are employed. The only approach that explicitly addresses symbolic locations is the common syntax proposed by the IETF's Spatial Location BOF. However, their classification of locations into three groups containing a finite enumeration of location names or arbitrary text, respectively, is far too restricted and lacks formality. In addition, although many of the abovementioned models associate locations with additional attributes such as velocity or orientation, none of them provides a comprehensive and extensible approach to adequately expressing the potential complexity and quality of location information. Thus, a model for the context element of location that is to be used in information logistics cannot be based on the standards introduced above.

SensorML is a more general approach towards describing the capabilities of context sensors. Although at present this language is still in the development stage, it should be possible to make use of SensorML descriptions in information logistic applications as regards the integration of context sensors. However, this language, too, accentuates the geospatial locations covered by sensors, whereas it does not include a comprehensive consideration of all possible elements and attributes of context and their values. Thus, a usage of SensorML throughout the Context Component of information logistic applications, for example as a means to represent the characteristics of sensors in the dynamic discovery mechanism discussed in Section 2.4.2, would require extensions to be made to SensorML and data transformations to be carried out and as a result is not reasonable.

Both the results gained in the course of other projects [InRo03] as well as our own assessment suggest that CC/PP is not sufficiently expressive to be employed as a means to model context in information logistics. The CC/PP framework nevertheless provides valuable input concerning the representation of device characteristics that can be taken over into our own modelling of the context element of reachability.

3.3 Summary and Assessment

This section sums up the main characteristics of existing context models, mechanisms for context gathering, and context-aware architectures. It furthermore assesses the proposed solutions with regard to the requirements we have identified and to their suitability for being adopted in information logistics.

Many context models are based on ad hoc data structures. The absence of a formal model and thus of a universal structure for context makes it difficult to process contextual information in a consistent manner. In addition, informal models prove to be very inflexible as regards extensibility and adaptability to changing requirements and environments. Furthermore, the variety of potential context attributes and possible values for them is currently not considered in a comprehensive way. Most approaches are unable to represent complex contextual information and to describe context with varying levels of precision. In addition, the design of many

models has been geared to particular application domains or environments, thus failing to be universally applicable. As a consequence, none of the existing approaches towards context modelling completely meets the requirements onto a context model for information logistics.

Generally speaking, existing approaches either focus on context modelling or, in the majority of cases, on the provision of methods and an infrastructure for context gathering. Mature concepts concerning one of these issues are usually achieved at the expense of neglecting the other. While some context-aware systems support only a limited set of context sensors, the need to be able to gather data from a variety of different sensors has widely been recognized. Thus, many approaches address issues of context sensor integration and sensor data heterogeneity and aim to encapsulate sensor-specific functionality. As far as the augmentation of context data is concerned, however, very few elaborate concepts exist. Except from the SOLAR system and, to a lesser extent, the Context Toolkit context data augmentation is not considered at all or is restricted to very simple mechanisms. Furthermore, a generic and extensible set of quality characteristics is at present being considered by Henricksen et al. only. Other applications and infrastructures either support merely a few specific quality attributes or do not deal with context quality at all. Similarly, mechanisms for a dynamic discovery of available context sensors are not provided by all approaches. Since the requirements onto context gathering we have identified are therefore fulfilled only partly by existing systems, an implementation of context gathering processes in information logistics cannot be based upon the available concepts.

Architectures for context-aware applications may pursue either a centralized or a decentralized approach. In the former case a single context server is responsible for providing contextual information to clients. A decentralized architecture, in contrast, distributes the gathering, management, and supply of context over several processing resources. Although a centralized architecture has some disadvantages with respect to privacy and scalability, it is at present the more common approach. Decentralized processing as, for example, carried out in the TEA project and the SOLAR system, on the other hand, suffers from an increased amount of computation and communication and may require additional efforts to ensure data consistency. The information logistics framework stipulates a centralized Context Component – although its subcomponents and modules may of course be distributed as well – which makes fully decentralized approaches unsuitable for information logistics. Aspects of transparent distribution, however, are also addressed by some centralized architectures. Another important issue concerning architectures for context-aware applications is the separation of concerns between the gathering, management, and supply of context and its use. In some existing architectures such as those proposed by TEA, SOLAR, or the Coordinated Adaptation Platform there is no clear distinction between these concerns. As a result, applications are burdened with tasks that are beyond their actual purpose which significantly impacts the quality of these systems' software. Since the functional and non-functional requirements made onto architectures for context awareness are thus insufficiently met by existing approaches, they cannot be considered as a foundation for the Context Component of information logistic applications.

The examination of existing approaches has shown that so far no solutions exist that satisfactorily fulfill the requirements made onto context awareness in information logistics. As a consequence, none of these approaches can be adopted in our work. Instead, there is a need to develop new concepts and an innovative architecture for the Context Component in order to integrate features of context awareness into information logistics.

4 An Object Model for Context

Based upon the requirements onto context modelling this chapter presents an object model that serves as the foundation for representing and processing context. An object model describes the objects of an application and their relationships. It provides information about how an application is structured, i.e. which elements it consists of and which associations exist between them. The object model for context we introduce in this chapter ensures that context is processed in a uniform way throughout information logistic applications, thereby making the applications sensor-independent. We have chosen to represent context by means of an object-oriented model, because this technique satisfies the demand for a formal and structured representation of context. In addition, an object-oriented modelling technique fits best into the information logistics framework which, too, pursues an object-oriented approach. For this reason the context model presented in this chapter as well as the models developed in conjunction with context gathering and the architecture of the Context Component dealt with in the following chapters conform to the Unified Modeling Language (UML) [OMG03].

In Section 2.2.2.2 we have identified those elements of context that are particularly important to information logistics. We have furthermore differentiated between context elements belonging to the scope of the Context Component and those that are taken into account by other components of information logistic applications. Accordingly, our object model comprises the elements of location, state, reachability, and surroundings. Each of these elements has a complex structure that requires accurate modelling. Unlike other approaches, e.g. [GeJe01] and [BuPr01], we are convinced that a separation of concerns provides a more flexible and cohesive modelling of context with each element being treated on its own in an exhaustive manner. Therefore, the object model for context consists of separate models for the four context elements under consideration. This also accommodates the requirement for a separate query and processing of these elements. Some basic considerations concerning these models have already been presented by us in [Hase01c] and [Hase01b].

4.1 An Object Model for Location

The evaluation of proposed models and standards concerning the context element of location in the previous chapter has led us to the conclusion that existing approaches do not sufficiently meet the demands of information logistics. However, as the benefits of standardization with respect to interoperability are substantial, we have also pointed out the indispensability of a context model for information logistics to seamlessly integrate with proposed standards. In this section we present a location model for information logistics. This model fulfills the requirements identified in Section 2.4.1 and allows for an integration with data representations proposed for standardization. A German patent has been granted, and an application for an international patent has been filed for this location model[1]. Due to its complexity we elaborate the model gradually in the following by examining its various aspects step by step.

1. Title: »Method for supplying a program-aided information system with specific positional information«, German patent number 102 01 859, January 29th, 2004. International official file number PCT/EP03/00362, Date of international publication July 24th, 2003

4.1.1 Location containers

In this section we illustrate the grouping of locations into sets and lists. We refer to these sets and lists of locations as location containers. A grouping of locations into containers is necessary for several reasons. First of all, the location an entity is at may be an itinerary (or trajectory in the terminology of spatial databases). This type of location information is required by applications that provide information related to the itinerary a user is on and her current position on this itinerary, e.g. traffic information in the form of traffic jam warnings or estimated driving times. An itinerary is an ordered list of individual locations – the points on the itinerary – each of which is interconnected with one (for the start point and end point) or two (for all other points on the itinerary) other locations via a stretch of route. The stretches may possess additional attributes such as means of transport or distance, as illustrated in Figure 8. In order to represent itineraries in our location model we introduce location lists. In addition to the itinerary an entity is on, its current position on the itinerary, i.e. the singular element of the itinerary the entity is presently situated at, also needs to be known in many cases. Hence, in conjunction with itineraries the description of an entity's whereabouts is represented by an itinerary containing all its relevant points and the current position of the entity on the itinerary as well.

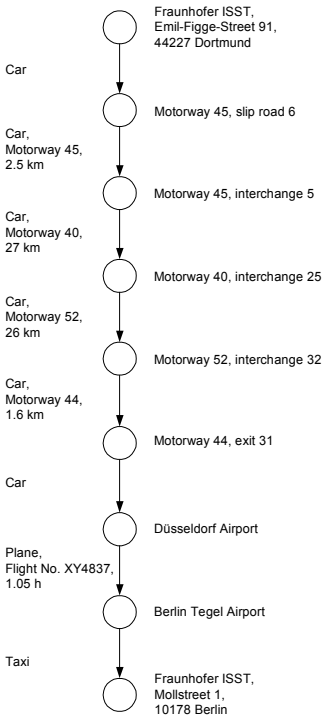

Figure 8: Sketch of an itinerary

Since the individual locations representing the points on an itinerary are ordered, the orientation of an entity that is on an itinerary, i.e. the direction the entity is oriented or moving towards, is determined by that point on the itinerary that follows the entity's current position. Apart from the orientation of an entity in the context of its movement on an itinerary the entity's orientation at its current location may be relevant as well and has to be treated sepa-

rately. For example, a salesperson who is on one of her sales routes is oriented towards the next customer she is visiting with regard to the sales route and in addition to this is also oriented towards a cash dispenser at her current location, a resting place. The representation of an entity's orientation at a single location is dealt with in the modelling of prepositions in Section 4.1.4 which allows us to define locations such as »in front of a cash dispenser« and in the modelling of state in Section 4.2 where the state elements of motion and activity may possess an attribute describing the direction a motion or activity is oriented towards.

Furthermore, information demands may refer to several locations or itineraries each of which causes or affects a user's requirement for information supply. An example of this is a salesperson wishing to receive the sales figures concerning her clients whenever she is on one of those routes that contain clients with a transaction volume greater than a certain amount. These cases of unordered enumerations of locations are represented in our model by location sets. Figure 9 shows a part of our location model describing the grouping of locations into containers and the relationships among the objects concerned. The `LocationSet` class implements a general `Set` interface providing the necessary operations to add, remove, or retrieve elements to or from the set. A `LocationSet` may contain one or more `Location` objects; it may also contain one or more `LocationList` objects.

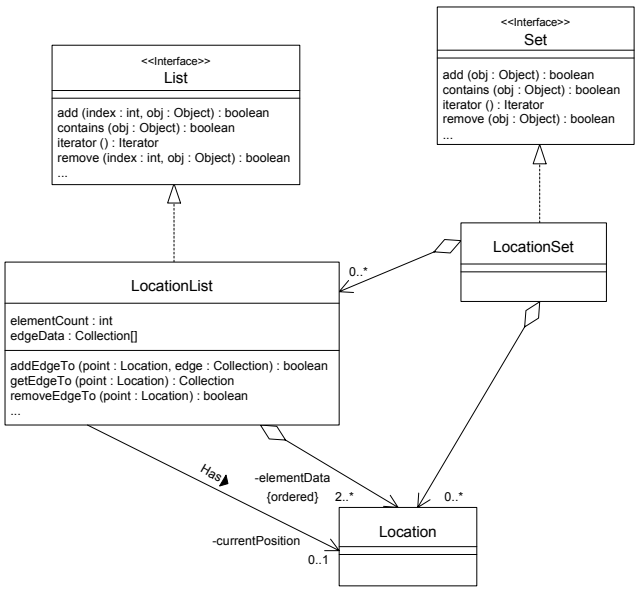

Figure 9: Location containers

The `LocationList` class also implements a `List` interface with standard operations for lists. In contrast to more general list objects, however, its nodes – depicted as an `elementData` attribute – are locations; as a result, a `LocationList` contains at least two `Location` objects. The edges connecting the nodes of a `LocationList` are the abovementioned stretches of routes. In our model these stretches are represented by the `edgeData` attribute of `LocationList` objects. In this attribute a vector of `Collection` objects containing relevant information about the stretches is stored. A `Collection` object's elements can be of

different types, e.g. `Distance` objects (see below), `Location` objects for means of transport or other places along the way, or character strings. The current position of an entity on an itinerary is indicated by a `Location` object. Therefore, there is a second association between the `LocationList` class and the `Location` class. This allows not only to cover the case that an entity currently is situated at one of the points on the itinerary, but also – via the construct of prepositions we explain later on in Section 4.1.4 – to indicate that an entity is situated between two of the itinerary's locations, i.e. on a stretch of route. In order to add, remove, and retrieve `edgeData` attributes to or from a location list the `LocationList` class defines additional methods not provided by the standard `List` interface.

In addition to location sets and location lists, we can identify two further location containers which we call location spans and repeated locations.

Definition 12: Location span

A location span is a location list denoting a spatial line segment that is bounded by two locations, the start point and the end point.

A location span is equivalent to an interval. Examples of location spans are »from here to Berlin« or »from the airport to the office«. Location spans are similar to location lists as they also contain ordered locations. Yet, they differ from location lists in that the number of locations they contain can be at most two. Besides, these locations are semantically interconnected with each other in a different way. Unlike an itinerary a location span does not provide any information about intermediate locations between start point and end point. The distinction between location lists and location spans facilitates the expression of information demands that occur at a particular segment of an itinerary, i.e. between two of the itinerary's locations that are not the start point and the end point, or refer to an entity's current position like in our first example. Therefore, we model location spans as a subtype of location lists.

Definition 13: Repeated location

A repeated location is a location list whose elements denote spatial recurrence.

In a repeated location spatial information occurs repeatedly as, for example, in »every 500 metres« or »every second kiosk«. Repeated locations are relevant for information demands that occur at a certain spatial frequency. Consider a car hire firm that wants to be notified that an inspection is due every 10,000 kilometres a car has been driven. The spatial information that occurs repeatedly can be either a location like the kiosk in the second example above or a length in space or time as in the first one. We model the latter kind of information as a distance. Distances are used to denote extents in space or time. They are employed in conjunction with repeated locations and prepositions which we describe in Section 4.1.4. A distance consists of a unit of measurement that can be a unit of length or time in repeated locations (or a location in connection with prepositions), a quantity of this unit, and a mathematic operator (=, < , >, etc.). For repeated locations this operator is mostly the equals sign. To continue our example of the car hire firm the unit of measurement in it is kilometres, the quantity is 10,000, and the operator is the equals sign. In addition to a distance, repeated locations also contain an ordinal number indicating which location or distance of a series is meant.

A repeated location can occur in conjunction with a single location, a location set, or a location list. Since repeated locations possess an inherent order – the sequence that they occur in has to be taken account of –, we also model them as a subtype of location lists. This also allows for them to be combined with a location set containing one or more locations and a repeated location as a conjunction to express information demands like »when I am driving the motorway 2, send me an updated traffic report at every junction«.

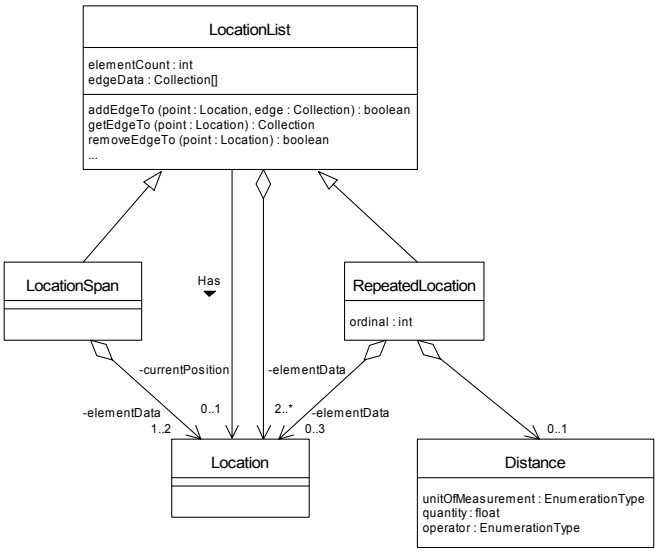

Figure 10: Location spans and repeated locations

Location spans and location lists are depicted in Figure 10. As already mentioned, both of them are subtypes of the `LocationList` class, but differ from it in the number of `Location` objects they contain. A location span contains one or two locations being the start point and the end point of the location span. A minimum cardinality of one means that one of the two points is not explicitly provided in an information demand profile and will be replaced by the current position of the respective entity as soon as the position has been detected. A repeated location contains zero to three `Location` objects. The maximum number of locations is given when a start point, an end point, and a frequency in the form of a location are specified. If there are no locations in a `RepeatedLocation` object, this means that no start and end points are provided and the frequency is indicated by a `Distance` object.

4.1.2 Location structure

The individual locations a location list or set consists of may be part of a hierarchical structure which we investigate in this section. The following concepts illustrate the distinction between atomic and non-atomic locations we have made when defining location in Section 2.2.2.2.

In many cases a location is known to be spatially included in some other location, for example a city in a country or an object in a room. Likewise, a location may contain one or more smaller locations, for example a region containing motorways or a building containing rooms. This

hierarchical structure of locations provides valuable information about the arrangement of locations and about containment relationships between locations. An entity that is situated at a particular location also is situated at any location containing it. This information becomes essential when locations have to be compared, in particular when an entity has been located at a certain place and the application has to determine whether this place matches given conditions in existing information demand profiles. These conditions may be fulfilled although the locations involved are not equal if the sensed location is contained in the location specified in a demand profile.

In order to represent containment relationships between locations our model associates locations with a structure as illustrated in Figure 11. A location is not required to possess a structure; the model allows for the use of it as needed and possible in a particular application. The structure itself manages a graph of locations that are arranged hierarchically. This graph consists of nodes and leaves, with nodes being elements of the graph that have other elements contained in them, whereas leaves do not contain any more elements.

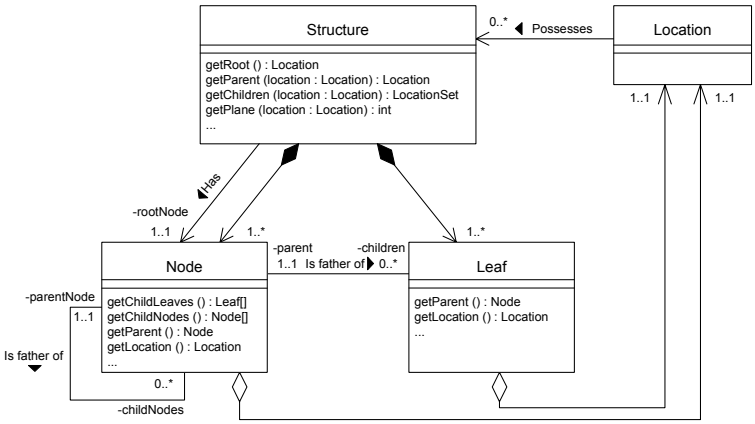

Figure 11: Location structure

A location is completely unaware of the nodes and leaves of the hierarchical structure it is part of. The `Structure` object a `Location` object may be associated with provides all the operations that are necessary to determine the graph of locations a particular location belongs to, its position in this graph, and the arrangement of all contained locations. For this purpose the `Structure` object aggregates both `Node` and `Leaf` objects; it also holds a reference to that `Node` object which is the root location in the graph. In order to facilitate the navigation in the graph `Node` objects know about both their parent nodes and their child elements which can be leaves as well as nodes. `Leaf` objects know about their parent node as well. In our model this mutual knowledge is achieved by bidirectionally navigable association relationships. `Node` and `Leaf` objects both aggregate one `Location` object each; this is the location the position of which in the graph is represented by the node or leaf. By this means the representation of locations and their characteristics is separated from the representation of their arrangement. As defined before, those `Location` objects that are aggregated in a `Node` object are denoted non-atomic locations, and those aggregated in a `Leaf` object are atomic ones.

There is a special case of locations that already possess some structural information in their coordinate representation (see Section 4.1.3): The coordinates of geographic locations, i.e. locations that represent addresses, are hierarchical by nature. The elements of an address that usually consists of a city, a street name and number and potentially some further information such as a zip code or country are already arranged in a hierarchical manner in that the city is contained in the country, the street in the city, and so forth. This intrinsic hierarchy of geographic locations is not represented by a location structure, but instead is contained in the coordinates of the location itself. When a geographic location is part of a structure, it may be the case that an address maps exactly to some other location, for instance a building, a site, or an enterprise. To give an example the geographic location »Otto-Hahn-Straße 19, 44227 Dortmund« maps exactly to the building location »Building of ISM International School of Management« and is therefore on the same hierarchical level. On the other hand, the geographic location »Mollstraße 1, 10178 Berlin« is on a higher hierarchical level than the building or enterprise location »Fraunhofer ISST Berlin«, because there are other enterprises that have their domicile in the building corresponding to this address, too. In our model a hierarchy can always be set up by reducing or expanding the level of detail of the geographic location. Reducing the geographic location in the example above to »Otto-Hahn-Straße, 44227 Dortmund« without a street number makes the geographic location contain the building location »Building of ISM International School of Management«. Nevertheless, our model also allows for considering the address as hierarchically higher than the building if a specific application benefits from this definition.

Theoretically there is little limit to the depth of the graph of locations, i.e. to the number of its planes, as virtually almost any location is contained in some larger location and may be split up into a number of smaller places contained in it. In applications employing this model, however, the quantity of utilized locations and the number of planes they constitute in a hierarchical structure are limited by various factors. These include the available location sensors and the data about the locations' properties associated with the coordinate systems (see Section 4.1.5). Therefore, our model leaves it up to the applications to build up a location structure as deep as required and feasible.

4.1.3 Location coordinates

This section investigates how locations themselves are represented in our model. We have already defined that each location is specified by a set of coordinates which belong to one particular coordinate system. As a result, an entity may be associated with more than one location at a time. This is the case when the entity's location is described by several `Location` objects whose sets of coordinates each refer to a different coordinate system or are related to the entity via different prepositions. For example, the location of a person may be »in the supermarket«, »at 33, Robert E. Lee Boulevard« as well as »in New Orleans« at the same time as these locations form a hierarchical structure and belong to different coordinate systems. A person may also be »on the chair« and »in front of the table« at the same time as these two locations are related to the person via different prepositions. We now describe in greater detail how the representation of locations themselves is accomplished.

From a functional point of view there are numerous ways of expressing spatial information. Apart from a geometric representation of locations in the form of terrestrian or nautical coordinates, expressed as latitude, longitude, and altitude, locations may also be described by names which is referred to as symbolic location representation. For both geometric and symbolic locations numerous different formats and ways of expressing an entity's whereabouts are possible. There is a variety of different coordinate systems for geometric locations such as Gauß Krüger, UTM (Universal Transverse Mercator) [HaBe89], WGS 84 (World Geodetic System 1984) [DrRi98], and many more. A geometric location's coordinates may be expressed as decimal degrees, degrees, minutes, and decimal seconds, degrees and decimal minutes, decimal gradients, or cartesian coordinates. Symbolic locations can be grouped into the following types:

- Buildings such as railway stations, prisons, hospitals, etc.
- Rooms
- Addresses as described in the previous section
- Topographical locations such as oceans, rivers, mountains, deserts, etc.
- Vehicles like cars, trains, aeroplanes, etc.
- Transport lines like flights, motorways, bus or railway lines
- Points on transport lines such as motorway exits or junctions or bus stops
- Objects like desks, machines, trees, etc.
- Events such as fairs, concerts, or sporting events
- Premises like forests, factory sites, car-parks, beaches, etc.
- Businesses such as hotels, supermarkets, bakeries, or restaurants
- Persons

This list of location types is neither complete nor mandatory. Depending on the scope and purpose of a particular application the list may be extended or more specific subtypes may be created. Likewise, location types that are not needed by an application do not have to be existent in it. For each type of symbolic locations different attributes may be used to further describe the locations of a type. In contrast to other location models [Leon98], [HaHo94] the list of location types given above does not explicitly include cellular locations, i.e. locations in the form of a cell constituted by the range of particular location sensors like RFID (Radio Frequency Identification) sensors or GSM cells [MoPa92]. This is due to the fact that in most applications this type of location is only relevant for positioning – and will be converted into a location of a different type immediately after the localization process – and does not occur in information demands. Yet, in specialized applications, for example applications to support telecommunications technicians, cellular locations may be referred to in information demands. In these cases they constitute a separate location type.

This variety of different types of locations and of attributes belonging to each type requires a sophisticated management of the location types, their attributes, and the valid values for them. We therefore do not regard a simple subclassing of `Location` objects into physical and symbolic locations and some subordinate types as it is done in other approaches, e.g. in [JoDa99] and [Kris00], as sufficient for our purpose. For this reason our model makes use of coordinate

systems which enable a flexible and powerful representation of locations. Figure 12 illustrates the representation of locations in our model. As already mentioned, each location is specified by coordinates that belong to a coordinate system. To represent this our model associates each `Location` object with one `Coordinates` object. A `Coordinates` object is in turn associated with one or more `Value` objects that contain the values of each coordinate. Any value refers to a location type's attribute, modelled as a dimension; therefore, every `Value` object is associated with one `Dimension` object.

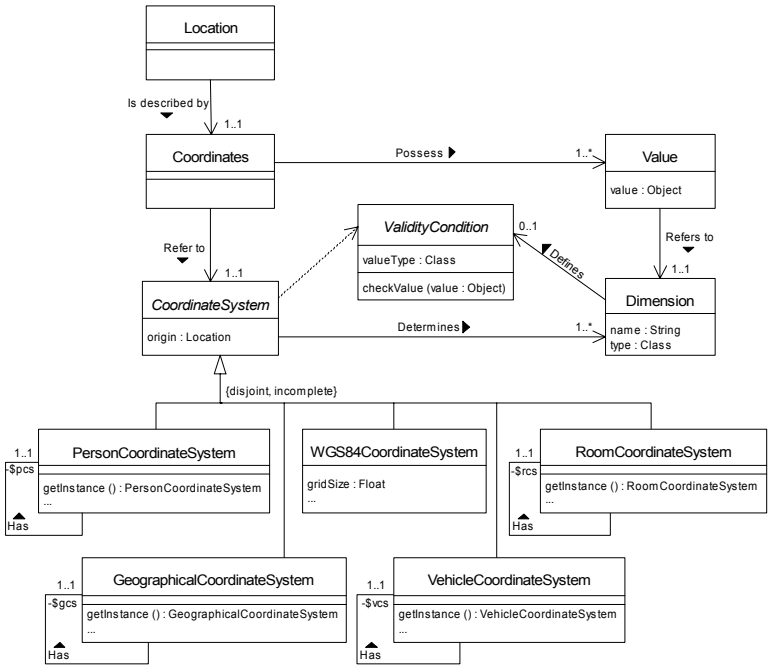

Figure 12: Location coordinates

There are various kinds of coordinate systems, corresponding to the different types or subtypes of locations an application needs to be able to process. Which coordinate systems finally exist in a particular application depends on the location sensors employed and their technical capabilities as well as on the available facilities for converting gathered location data into coordinate values of coordinate systems. A coordinate system is constructed for each location type that possesses distinctive attributes and therefore has to be treated separately. The coordinate systems shown in Figure 12 are not complete; they represent an example of possible types that may exist in an application. In the figure a `GeographicalCoordinateSystem` for addresses, a `RoomCoordinateSystem`, a `WGS84CoordinateSystem` for GPS data referring to the WGS 84 system, a `VehicleCoordinateSystem`, and a `PersonCoordinate-System` for so-called relative locations that refer to other people are exemplarily modelled. Dynamic locations, i.e. locations that are defined with reference to a person or object that is moving in space, certainly constitute a special type of location from a functional point of view and have to be treated in a specific way when localization is carried out. Yet, the representation of locations this model establishes does not necessitate any difference in the actual modelling of dynamic locations.

There may be various other subtypes of the abstract `CoordinateSystem` class for the representation of other types of location data. According to our definition each coordinate system has an origin. This origin allows to distinguish between locations that belong to the same type but are contained in different superordinate locations and therefore may have different valid attribute values. An example of this is two sets of rooms in two different buildings of a company. In the first building there are offices, meeting rooms, and classrooms; the second building contains laboratories, server rooms, and assembly halls. The valid values for the location's attribute room type therefore are completely different in each building which requires to set different valid values for the dimension room type. A coordinate system's origin is a `Location` object that denotes the non-atomic location containing all locations the coordinates of which refer to this coordinate system. If there is no superordinate location for a coordinate system that is relevant to the application and processable in it, the system boundary is reached and the origin contains an empty value. The origin is stored as an attribute in the abstract `CoordinateSystem` class and is inherited by all subclasses. The coordinate systems' origins are also used to build the hierarchical levels of a location's structure, whereas information about locations on the same hierarchical level is created on the basis of information about the locations' arrangement associated with the coordinate systems (see Section 4.1.5).

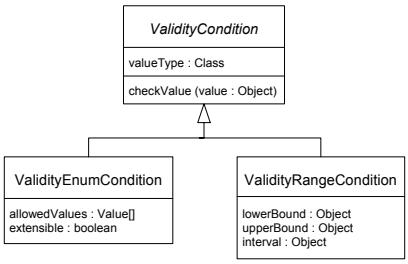

Figure 13: Classes for validity checks

Every coordinate system determines the attributes for the type of locations it represents and valid values for these attributes. For this purpose each `CoordinateSystem` instance is associated with one or more `Dimension` objects. The dimensions contain the names and the types of the locations' attributes. The values these attributes may take on may be restricted to a range or enumeration of valid values or may be subject to other conditions they have to fulfill, e.g. a certain format. In our model an abstract `ValidityCondition` class is associated with a dimension which is responsible for performing the necessary examinations to determine whether the value of a coordinate is valid. Objects of concrete subclasses of the `ValidityCondition` class are instantiated by the respective coordinate system. Such subclasses of the `ValidityCondition` class undertake the task of performing specialized validity checks. This can, for example, involve the verification of a value with respect to a particular value range the value has to be within or with respect to a set of discrete values the value has to be one of. This model provides the classes `ValidityRangeCondition` and `ValidityEnumCondition`, both subclasses of the `ValidityCondition` class, as a means to cover both of the mentioned validity checks, as depicted in Figure 13. Other subclasses of the abstract `ValidityCondition` class may be created to provide more specific verification methods.

There may be more than one instance of any particular coordinate system. As far as symbolic locations are concerned, this is the case when two coordinate systems of the same type have different origins, e.g. two room coordinate systems for rooms in two different buildings. If several instances exist, there may be information demands that refer to the type of the coordinate system or to one or more values of its dimensions in general without specifying a particular instance. To continue our example a user may define to not be delivered certain pieces of information whenever she is in a meeting room, no matter in which building this room is situated. In our model these cases are handled by an association to self the coordinate systems possess. Those coordinate systems that may have several instances contain a static attribute whose type is the same as the coordinate system itself. This attribute may be obtained by a call to a `getInstance()` method and can be used to specify information demands like the one described by setting the requested dimension values. In this regard we would like to mention that by definition the dimensions a particular coordinate system possesses are the same for all its instances, i.e. the attributes of all locations that form a type are the same.

The `WGS84CoordinateSystem` shown in Figure 12 does not posses an association to self. This is due to the fact that coordinate systems for geometric coordinates do not have an origin as the sum of the locations they represent is not contained in any other location. However, it is possible to have subclasses of such a geometric coordinate system in an application. This occurs when there are several location sensors that provide coordinates referring to this system, but that possess different degrees of accuracy.

Definition 14: Accuracy of a location

The accuracy of a location is its spatial exactness. It is determined by the size of the space specified by the coordinates of the location.

Geometric locations, although belonging to the same coordinate system, may have different accuracies if sensed using different technologies whose smallest detectable areas are distinct from each other. The accuracy of geometric locations is represented by the `gridSize` attribute of the respective coordinate systems. If sensors with different degrees of accuracy are employed in an application, an abstract geometric coordinate system is defined that presets the dimensions and some basic operations common to all subtypes, e.g. transformation operations (see Section 4.1.5). This abstract coordinate system is the parent of all subtypes who set a specific value for the `gridSize` attribute and may add or override the way validity checks or transformations are carried out.

Symbolic locations, on the other hand, do not possess an accuracy. Although on rare occasions the sensors employed to determine symbolic locations may be limited to a smallest area that is larger than a point formed by the coordinates of a symbolic location, the way of expressing this blur is different. For symbolic locations a location set can be constructed that represents an area containing all locations covered by the sensor. An example of this are sections of railway tracks that are relevant locations in an application supporting track maintenance. Each track section is a location that causes particular maintenance information to be supplied to the workers on the track. Yet, the location sensor employed only allows for the sensing of an area that covers two tracks. This is represented by a location set containing the two tracks covered by the sensors. The singular symbolic locations themselves are always considered precise.

Apart from setting dimensions and valid values for them coordinate systems may also be associated with information sources that provide additional information related to the dimensions and dimension values. We refer to these sources of information as operation execution services. They are especially relevant for symbolic locations that are ordered in space or time. The coordinate systems may have access to information regarding the order of the locations that refer to them such as maps or plans of buildings, topological locations, etc., or timetables of events, flights, trains, and so on. Operation execution services are employed to determine relationships between locations such as their distance or to carry out location transformations. They also provide additional information about the non-hierarchical arrangement of locations that is not covered by the location structure described in Section 4.1.2. Since operation execution services are only used when operations are invoked on locations and are not part of the representation of locations themselves, we describe them and their usage in detail in Section 4.1.5.

4.1.4 Prepositions

So far our explanations have been restricted to the modelling of locations alone, without any spatial relation to persons or objects. Yet, since the main purpose of the location model is to serve as a basis for the provision of individuals with information adapted to their own or another entity's whereabouts, these relations of persons or objects to locations are essential. In this section we introduce the concept of prepositions as a means of representing the spatial relation between locations and entities.

The term preposition stems from linguistics and denotes »*a word or phrase placed typically before a substantive and indicating the relation of that substantive to a verb, an adjective, or another substantive*« [Pick00]. Examples in the English language are *at*, *with*, *in*, *from*, or *above*. In our location model prepositions are used to indicate the spatial relation of an entity (a substantive) to a location (another substantive). This relation may either be existent in reality – when an entity has been located at a particular location – or it may be imagined or expected as in a location-dependent information demand. An explicit modelling of prepositions is not necessary in every information logistic application, because the location information processed in an application is often implicitly restricted to one particular preposition per location. An example of such an application is the Knowledge Worker prototype developed at the Fraunhofer ISST and exhibited in the Fraunhofer Office Innovation Center [DeLu00]. In this application the relevance of information to office workers is determined on the basis of their current location, and information supply is carried out or deferred accordingly. The locations being processed in the Knowledge Worker system are rooms, and the only relation of an entity to a room that is being processed is that of being in a room. In applications like this the usage of prepositions is implicit, and our model does not enforce to explicitly include the modelling of prepositions in order to keep an application's complexity to a minimum.

However, there are applications that require a more sophisticated expression of entities' locations. In these applications the different relations of an entity to a location are relevant to the application's behaviour. The Bat Teleporting system, for example [HaHo99], displays a person's screen on any monitor the person is close to whenever she is standing within a certain range in front of a monitor. As an effect, a person's screen virtually follows her around. In this applica-

tion it would be pointless to display a person's screen on a monitor behind her where she cannot see it. Another simple example is a salesperson who receives the latest sales figures for each customer she is driving to on her CarPC when she is no more than two kilometres away from the customer's. In these cases the relation of an entity to a location needs to be stated explicitly, and our model enables this explicit modelling by means of prepositions.

In modelling prepositions we start from the assumption that prepositions can only be modelled in connection with single locations or location sets, the latter meaning that the relation a preposition indicates refers to all locations contained in a set. A relation of an entity to a location list can only occur in the form of an entity's being on an itinerary; other prepositions make no sense in conjunction with a location list as a whole. For this reason we refrain from modelling relations to location lists. Different relations of an entity to any particular point on the itinerary are possible; however, they refer to a single location that is part of the location list and are therefore covered by a relation referring to this single location.

The part of our location model that covers prepositions is illustrated in Figure 14. The concept of a preposition being a relation between an entity and a location is reflected in the abstract `Relation` class which on the one hand serves as a superclass for the `LocationRelation` and `LocationSetRelation` classes we describe shortly and on the other hand is associated with the `Preposition` class representing the actual word or phrase that indicates the relation. A `Preposition` object may be associated with at most one `Distance` object. We have already mentioned this class in connection with repeated locations. Distances in the context of prepositions are again used to denote extents in space or time that occur in addition to the mere word or phrase of a preposition alone as, for example, »20 kilometres around Hamburg« or »less than 20 centimetres in front of a monitor«. A special feature of distances in the context of prepositions is that the unit of measurement may also be a location itself. An example of this is »one floor above Ogro Inc.'s offices«. Distances whose unit of measurement is a statement regarding time may need to be extended with additional attributes such as a means of transport which affects the amount of time stated in the distance. For distances measured in time specific subclasses of the `Distance` class may be created. As already mentioned, the aspect of time in information logistics is part of another research activity, so that we do not regard this field in detail in our work.

Not every preposition is suitable for every type of location. This has its causes in the topology of the locations of a type, in the scope and purpose of an application – for instance, »above flight no. 4711« will hardly be a meaningful preposition for the given location in most applications, but may be vital in a system for airspace control –, and in the location sensors and their capabilities available to an application. In addition, the significance of a preposition in combination with a particular location often requires a common understanding among all application's users and administrators. To take this into account each `Preposition` object is part of a pre-defined set of prepositions, represented by a `PrepositionCollection` class. Following the Singleton design pattern [GaHe94], there may be only one instance of this class in an application. The set of available prepositions can be configured for any specific application. They can only be instantiated by the `PrepositionCollection` class. The permissible prepositions for the locations of a type are defined by the coordinate systems as they know about the locations' topologies. In order to make sure that only prepositions that are available in an

application can be permitted by its coordinate systems the coordinate systems call methods like the exemplarily depicted `getPrepositionByName()` method on the `PrepositionCollection` class to be returned valid `Preposition` objects.

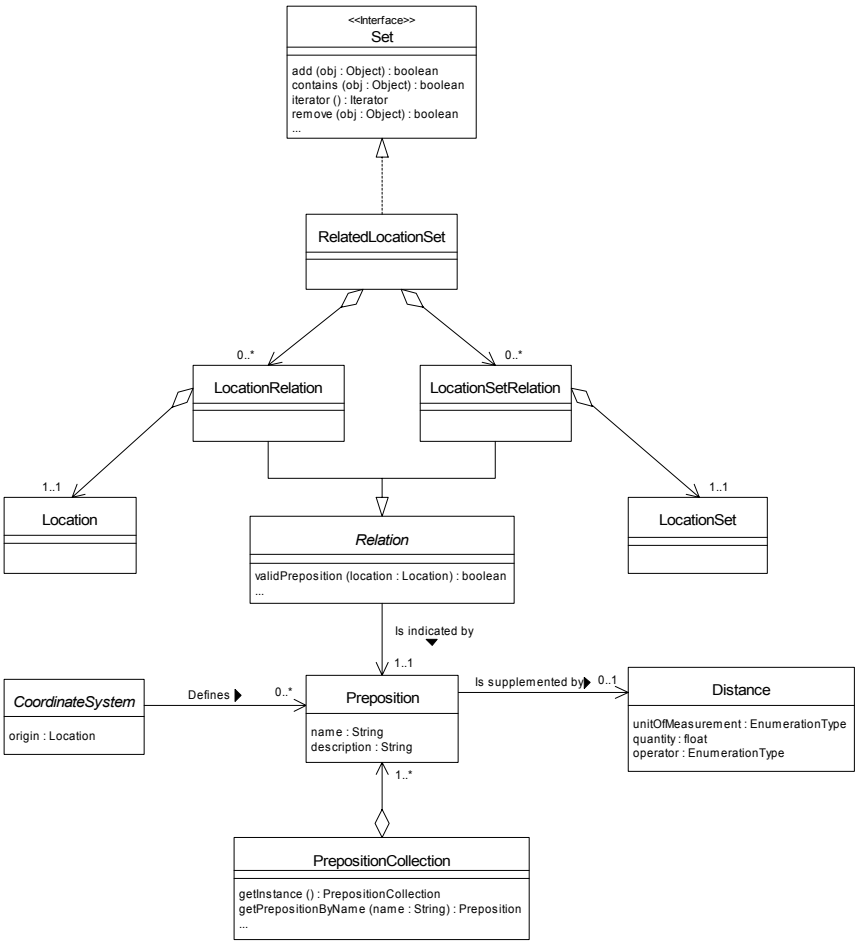

Figure 14: Location prepositions

In order to associate prepositions with locations the `Relation` class, apart from its association with the `Preposition` class, serves as the abstract superclass for the `LocationRelation` and `LocationSetRelation` classes. These two classes represent locations and location sets, respectively which possess a statement regarding their relation to an entity. The `LocationRelation` class aggregates a `Location` object and the `LocationSetRelation` class a `LocationSet` object. Due to their inheritance from the `Relation` class both classes are also associated with a preposition which is the preposition that indicates an entity's relation to the aggregated location or location set. Methods inherited from the `Relation` class allow to perform checks regarding the validity of a preposition that is to be associated with the location or location set during the allocation of the `Preposition` attribute. This is done by retrieving all prepositions permitted by the coordinate system the aggregated location belongs to and comparing these to the preposition that is to be allocated.

Both the `LocationRelation` and the `LocationSetRelation` class are again aggregated in a `RelatedLocationSet` class. This class – analogous to the `LocationSet` class – implements the general `Set` interface. The `RelatedLocationSet` class serves as a container for locations and location sets that are related to an entity via a preposition. It also enables a unification of the Context Component's interface by ensuring that the supplied location information is always a subtype of the same `Set` type.

4.1.5 Operations on locations

Outside the Context Component location information is used to determine whether corresponding information demands exist and have to be fulfilled as well as to optimize the supply of information to users. In order to do so the location information provided by the Context Component has to be processed in various ways. This involves the comparison of different locations, the determination of equality, proximity, containment, or overlapping of locations, and – due to the varying coordinate systems a location's coordinates can refer to – the transformation of a location into a location referring to a different coordinate system. Locations therefore need to provide several methods available for call both by other components of information logistic applications and by the Context Component itself. In this section we describe these operations the `Location` class makes available together with the objects and interfaces of the location model that serve to implement the execution of these operations.

The operations to process location information used by other components of an information logistic application or within the Context Component itself are:

- Determination of equality

  A location can be compared to another location for equality. Two locations are equal if the coordinates of both locations refer to the same coordinate system and all corresponding pairs of elements in the two coordinates have the same value. The determination of equality does not involve an examination whether two locations the coordinates of which refer to different coordinate systems describe the same place. In order to determine this the locations have to be transformed into a uniform coordinate system first. This implementation follows the behaviour of equality operations in the Java programming language which is the reference implementation language for information logistic applications, and it contributes to a separation of concerns.

  The corresponding method of the `Location` class to check for equality with another location is `equals (otherLoc : Location) : boolean`. The method returns `true` if the two locations are equal in the way described above; otherwise it returns `false`.

- Determination of containment

  Containment of locations is given when a location is spatially included in another location. These locations do not have to belong to the same coordinate system. We have already described the concept of spatial containment, and in connection with this we have introduced location structures. The determination of containment uses the structure of a location or – if at the given point in time no structure has been built – the origins of the coordinate systems to check for spatial inclusion.

To determine containment the `Location` class possesses a method `inIn (otherLoc : Location) : boolean` that returns `true` if the location on which the method is called is spatially included in the location passed to this method; otherwise it returns `false`.

- Determination of overlapping

  Two locations may overlap which means that each location covers a part of the other location, but is not fully contained in it. An example of overlapping locations is an assembly line installed across both the goods collection area and the packing area of a company's stockroom. As can be seen from this example, overlapping of locations is determined among different coordinate systems. To determine if and to what extent two locations overlap different sources of information which we call operation execution services can be used. Coordinate systems may have access to additional information about the arrangement of locations; in our example the coordinate system referenced by the stockroom building may be associated with a database, a map of the building, etc. containing information about the areas a building is subdivided into and installed objects and machinery. Other operation execution services may be geographical maps, GIS software or others. Depending on the available services and their capabilities a transformation may have to be carried out before the actual check for overlapping can be performed. We shortly describe how operation execution services are integrated into the location model and how they are used.

  To determine if two locations overlap there is the `overlapsWith (otherLoc : Location) : float` method. This method returns a floating point number in the range from zero to one with zero indicating that there is no overlapping, values from zero to less than one indicating the extent to which the given locations overlap, and a value equal to one indicating that there is an overlapping whose extent cannot be determined.

- Determination of proximity

  In many cases an optimization of information supply involves the determination of how far two locations are away from each other, for example to suggest an appropriate means of transport to reach a destination or to provide a person with information about monuments of industrial culture she is close to when travelling through the Ruhr district. To determine the proximity between two locations – which may belong to the same or to different coordinate systems – again operation execution services are employed; these can be the same as those used to check if locations overlap or others.

  The `Location` class provides the method `getDistanceTo (otherLoc : Location, measurement : String) : Distance` in order to determine how far two locations are away from each other. The second parameter of this method states which unit of measurement the returned `Distance` object is to contain.

- Transformation of a location

  Due to the various coordinate systems that may exist in an application a transformation of a location into a location with coordinates referring to a different coordinate system is often necessary. Such transformations are also required within the Context Component itself, for instance as a first step to determine overlapping or proximity. These transformations can again be carried out in different ways. They can be implemented as algorithms analogous to

the coordinate transformations employed in geometry (e.g. to convert a geometric location into another geometric location with a different reference system), as transformations based on geometric locations or on names that are entries in databases or other information systems, or as a combination of these methods (e.g. a database containing the geometric locations of buildings or topological locations).

The corresponding method of the `Location` class to transform a location is `transform (destination : CoordinateSystem) : Location`. Its parameter denotes the target coordinate system the location's coordinates are to be transformed into, and it returns the transformed location.

We have already mentioned that additional operation execution services are needed in order to carry out the operations locations make available. These services are of various types such as algorithms, databases, GIS software, maps, or web services, for example according to the Gazetteer Service specification [AtFi02], and the like. They may also be made up of a combined usage of several of these data sources, services, and operations. Some operation execution services are external systems or services, meaning that they do not belong to the information logistic application itself, but are made available by some third-party service provider, while others are developed in conjunction with the information logistic application. Hence, the data, their formats, and the interfaces operation execution services provide can differ from each other, and there is no universal way of describing the attributes and methods of them. Therefore, an additional layer of adaptation to the different interfaces of operation execution services and a service brokerage are required. Operation execution services are not part of locations or coordinate systems themselves, but are associated with coordinate systems which possess all information about the coordinates' dimensions and their permitted values as a means to support operation execution.

We distinguish between three different kinds of operation execution services. The first one, transformation services, is capable of transforming a location's coordinates into coordinates referring to a different coordinate system such as the abovementioned algorithms for geometric locations, for example. The second type of operation execution services is able to calculate a distance between two locations as, for instance, a geographical map and is called distance calculation services, and the third type provides additional information about the arrangement of locations and is therefore called arrangement services. Any existing operation execution service may belong to one or more of these types; it can be capable of providing only one up to all of the services.

Due to the variety of different coordinate systems the transformation of locations is of particular importance to our location model. Therefore, we address the issue of transformation services in greater detail here. The large number of different coordinate systems that may exist in an application leads to a vast amount of possible transformations that may have to be carried out. A provision of a complete set of transformation services for the transformation of locations from each into every coordinate system therefore would miss the point of this thesis; it would even violate the requirement for a generally applicable location model. This is due to the fact that application-specific coordinate systems may be defined which require special processing that cannot be implemented in advance and that the execution of transformations is often based upon application-specific instances of the data sources and services employed.

For this reason we consider it reasonable to restrict ourselves to defining the generally applicable method for transforming a location described above and leave the implementation of transformations up to the individual applications. On the basis of the W@keUp traffic information system [LiSc01] we give an example of how a transformation of locations may look like. The W@keUp system is an information logistic application that provides users with information about the current and forecast traffic situation on their personal routes before and while driving in a car. The locations processed in this applications are both itineraries that represent motorway sections and GPS coordinates. The GPS coordinates have to be transformed into junctions, exits, or intersections on the itinerary a person is driving on in order to determine at which position on the itinerary the person currently is. Based on this position the appropriate traffic information concerning the route ahead is selected and delivered to the user. The transformation service implemented uses both a database containing the names, numbers, and GPS coordinates of all junctions, exits, and intersections of the motorways in the Ruhr district of Germany as well as mathematical algorithms. First of all, a circle around the GPS coordinates indicating a user's position is calculated. All junctions, exits, and intersections the GPS coordinates of which are within this circle are retrieved from the database. After that the route the user effectively is on is calculated in several steps that take routes crossing the circle or those with a sharp bend into proper consideration, resulting in two sets of GPS coordinates for the points that make up the route the user is on. Finally, the names of these junctions, exits, or intersections are retrieved from the database, and with these names the Content Service providing the traffic information is queried.

The operations the `Location` class makes available and the brokerage of and adaptation to operation execution services are depicted in Figure 15. Since operation execution services are associated with coordinate systems, the operations on locations are also methods of the coordinate systems which in addition provide methods to obtain information about the arrangement of the locations belonging to them. For each of the three types of operation execution services a specific adaptor exists that provides the coordinate systems with a uniform interface for the respective operation and internally maps the methods of this interface to a service-specific access to the service. These three adaptors are the `TransformationAdaptor`, the `DistanceCalculationAdaptor`, and the `ArrangementAdaptor`, and the interfaces belonging to them are called `Transformation`, `DistanceCalculation`, and `Arrangement`. These interfaces possess methods to transform a location, calculate a distance between locations, and provide information about a location's arrangement, respectively.

Each adaptor to an operation execution service is associated with a `ServiceDescription` object containing information about the capabilities, properties, and interfaces it is able to supply on behalf of the operation execution service. It also implements a `RegisterableService` interface containing a method to obtain this description. This interface also serves to ensure type safety for all adaptors and is used for the registration of the adaptors with a `ServiceRegistry` object. All adaptors register themselves with this registry, and the coordinate systems use it in order to find appropriate operation execution services capable of carrying out a required operation. For this purpose the service registry possesses several `lookup()` methods and is associated with one or more so-called `ServiceMatchingService` objects responsible for matching a provided `ServiceSpecification` object which is similar to the service description and contains the required capabilities of an operation execution service against the available services. For reasons of clarity we do not depict the elements of this

matching process in Figure 15. This brokerage mechanism, however, has been prescribed as the standard means for service management and access in the information logistics framework and is used in other application parts as well. In connection with operation execution services the `ServiceSpecification` and `ServiceDescription` objects contain the coordinate systems of the coordinates the service accepts as input parameter and – with the exception of distance calculation services – those it is able to supply.

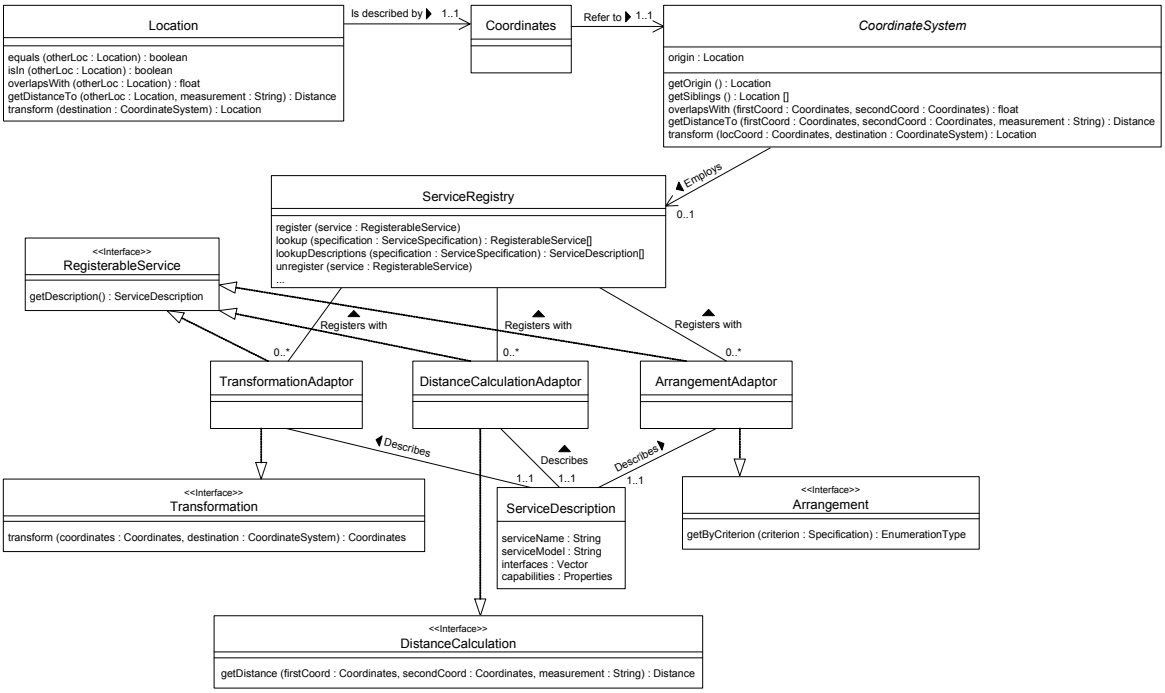

Figure 15: Operations on locations and operation execution services

4.1.6 Interactions with the location model

We round off our explanations concerning the location model in this section by describing which kinds of objects interact with the model and how this interaction takes place.

The location model is used to satisfy location-dependent information demands or, in other words, to ensure that information is delivered at and to the right place. For this purpose three main tasks have to be carried out: First, the location dependency of information demand has to be formulated and stored in an information demand profile. This can either be done by a user or – in case of an implicit determination of information demand – by a component of the information logistic application. A further task is to locate the entity an information demand refers to, either the user herself or another person or object, and to convert the data gathered from location sensors into locations according to the location model. Besides, information demands have to be processed outside of the Context Component to determine whether the

conditions of an information demand are met and an information supply is to be carried out which again has to be optimized with regard to various dimensions, one of which being the context element of location.

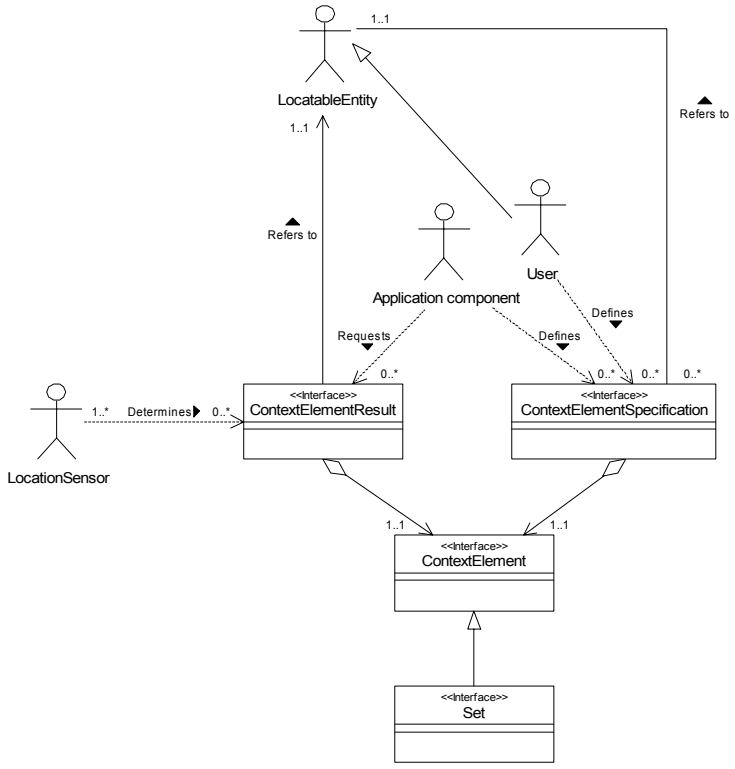

Figure 16: Interactions with the location model

We can therefore identify three main interactions with the location model: Location profile creation as a part of information demand specification, localization, and processing of location information, as illustrated in Figure 16. As actors, i.e. interacting objects external to the location model, we first identify location sensors responsible for locating an entity. Location sensors provide so-called `ContextElementResult` objects that each aggregate a `ContextElement` instance (see Section 5.1.4.3), in this case a `Set` which is either a `LocationSet` or a `RelatedLocationSet` object. A `ContextElementResult` refers to a locatable entity. Locatable entities are actors denoting entities that can be located. The actor `User` is a subtype of the locatable entity; users may additionally specify information demands the location-dependent part of which is depicted as the `ContextElementSpecification` interface (see Section 5.1.4.1) which also aggregates a `ContextElement` object. Furthermore, there are application components other than the Context Component which may also specify information demands and which in addition process location information to carry out an optimized information supply. The interactions between actors and the location model described in this section apply analogously to the way external entities interact with the models for the other elements of context as well as with the context model as a whole. For this reason we do not provide repeated explanations concerning these interactions in the following sections.

4.1.7 Summary

In this section we recapitulate the various aspects of our location model and provide a brief assessment of it. This summary is accompanied by an illustration of the overall model in Figure 17 on page 68.

At first, we have introduced location containers as a means to group individual locations. Our location model represents unordered enumerations of locations by `LocationSet` objects and itineraries by `LocationList` objects. Furthermore, location spans and repeated locations as subtypes of location lists serve as a representation of spatial intervals between two locations and of spatial frequencies, respectively. Moreover, the location model is able to describe hierarchical arrangements of locations by associating locations with a structure. A structure manages a graph containing nodes and leaves and provides methods needed to navigate in the graph of locations.

We have explained how locations themselves are represented in our model. For this purpose we have identified various types of locations each possessing different attributes and possible attribute values. The concept of coordinate systems has been introduced to cope with this variety of different location types. There are various types of coordinate systems each of which corresponds to a location type and defines its attributes and the valid values for these. Each location is associated with a set of coordinates which refer to a coordinate system. This representation of locations fulfills the requirement for generality by shifting the characteristics specific to different types of locations into coordinate systems. In order to distinguish between locations of a type that are contained in different superordinate locations and therefore may have different valid attribute values every coordinate system has an origin. This also makes it possible to create different instances of a particular coordinate system and to define information demands that relate to any location of a specific type in general. We have furthermore defined the accuracy of locations which is represented by an attribute coordinate systems for geometric locations possess, whereas symbolic locations are always considered precise.

Subsequently, the concept of prepositions has been introduced with the purpose of explicitly relating an entity to a location. This relation is achieved by subclasses of an abstract `Relation` class which is associated with a preposition representing a word or phrase. The model allows to define the set of available prepositions in an application and the valid prepositions for the locations of a type. In addition to this, prepositions may be complemented by a distance.

We have furthermore identified and described the operations that can be invoked on locations. These operations allow to determine equality of locations, containment, overlapping, and proximity as well as to transform a location into a location with coordinates referring to a different coordinate system. In addition, we have explained the usage of operation execution services as a means to carry out transformations and distance calculations and to obtain information about the arrangement of locations needed to determine if locations overlap. Due to the different interfaces and capabilities of these services our model includes mechanisms for adapting to them and for a brokerage of suitable services capable of carrying out an operation on given locations with given parameters.

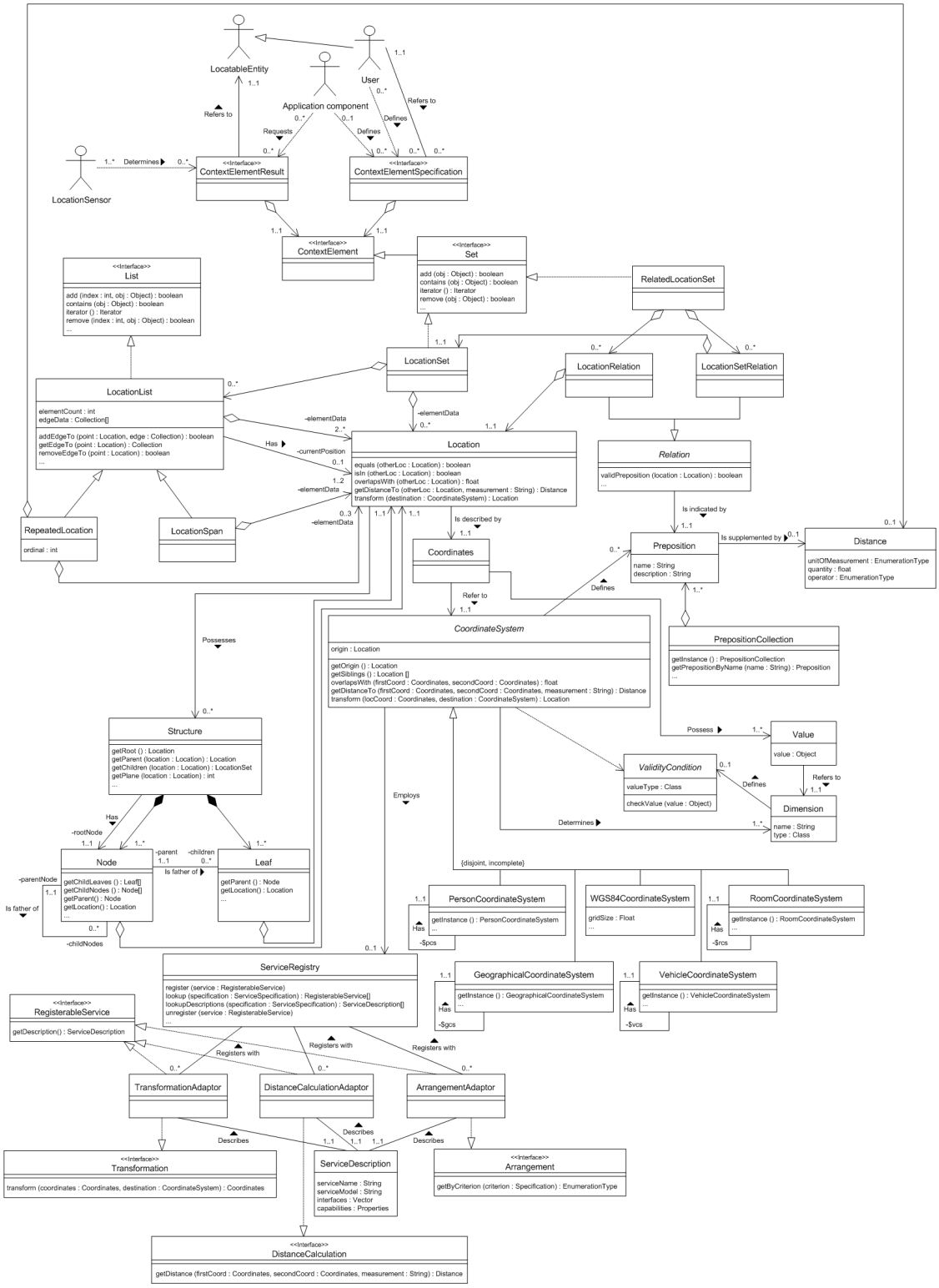

Figure 17: Overall location model

Finally, we have described the interactions with the location model taking place in information logistic applications. The three main usages of location information represented according to our model are the definition of location-dependent information demands, the localization of entities, and the processing of location information. Starting from these applications of the location model we have furthermore identified the actors interacting with it as being location sensors, locatable entities and users as a subtype of them as well as application components other than the Context Component itself. Location sensors provide location information referring to a locatable entity which is processed by the application components. Both the application components and the users furthermore define information demands.

Our object model for location distinguishes itself by the ability to represent locations both in geometric and in symbolic form. It therefore belongs to the category of so-called combined location models [Leon98]. The model allows to integrate various location types into an application, including itineraries and dynamic locations. Other main features include sensor independence, the ability to represent containment, distance, and explicit relations between entities and locations, and the possibility to determine relations between locations by using the operations locations make available. Our model supports locations with different degrees of complexity and precision. In addition, it ensures that location information is processed in a structured and consistent manner throughout information logistic applications.

The major advantages of this location model for information logistics are:

- Locations can be represented in both symbolic and geometric form, thus allowing to define locations as required and exploiting the advantages of both types of location representation.
- The model is generally applicable. Various kinds of locations, for example virtual locations, i.e. locations that do not have a physical presence in space, can be described.
- The location model can be easily extended to application-specific locations.
- The ability to define locations as needed and to refer to locations by their type and attributes makes the model very convenient for its users.
- The decoupling of sensor data from location representation makes the model sensor-independent and allows for the integration of various sensors.
- Mobility patterns or other secondary location information can be created easily based on the discrete and well-structured symbolic locations.
- The modelling of prepositions allows for very fine-grained and precise indications of place.
- Various relations between locations like containment, equality, proximity, or overlapping can be determined, thus enabling an easy detection of whether an information demand is to be satisfied and a facilitated optimization of information supply with regard to the dimension of place.
- Transformations of locations make different representations of the same location possible and thereby increase flexibility and usability.
- Locations with various levels of accuracy are possible.
- Access control is facilitated, because locations can be represented in symbolic form and referred to by their type or attributes.

We believe that these advantages of our location model fully compensate for the disadvantages listed below:

- The set of useful symbolic locations in an application depends on the application domain. This may result in a large number of symbolic locations that have to be managed and possibly created manually.
- The high functional capacity and flexibility of the location model can only be achieved at the cost of an increased model complexity resulting in a risen amount of data and computations.
- Transformations cannot be carried out without information loss or ambiguity in any case.
- External components have to understand and process more than one reference system locations refer to.
- A separate transformation of sensor data into locations according to the model is required.

4.2 An Object Model for State

As explained in Section 2.2.2.2, the state of an entity comprises information about the entity's mental and/or physical situation. It consists of motion, activity, physical condition, and emotional condition which are the relevant aspects of state in information logistics. Our investigations of the few existing activities in the area of modelling the elements of state have led us to the conclusion that so far no suitable models exist which in part is due to the fact that the context element of state is very extensive and may take diverse shape. In this section we therefore present a generic object model for state capable of representing the different attributes characterizing the mental and physical situation of entities. In doing so, we treat the individual elements of state successively.

4.2.1 Motion

Due to recent changes in society, technology, and culture people have become more and more mobile, and a growing demand for information access independent of location has been arising. In order to ensure that information is delivered to people in line with demand and adjusted to their context the ways in which people are mobile have to be examined in greater detail. This is done in this section by modelling the motion of an entity, i.e. the different types of entities' movements. The state element of motion is also relevant in conjunction with information supply, because the communication media that are available and, above all, convenient to use at a given point in time often depend on the way a person is moving in space. In addition to that, the relevance of information to a person is influenced by the person's motion pattern, among other factors. Consider, for example, a purchasing manager who wishes to be delivered only very urgent messages to her cellular phone when visiting and conferring with a supplier, whereas she would like to receive certain information of medium urgency on the phone or the CarPC during the journey to the supplier. The distinction between the state element of motion and that of activity (see Section 4.2.2) is in some respect arbitrary as performing a motion can be considered an activity as well. Yet, due to the special role mobility and its different patterns play for the optimization of information supply with regard to the dimension of

context we have chosen to separate the issues of motion and activity from each other. Furthermore, treating motion independently facilitates the enhancement of mobile IT use by providing information technology that is adjusted to the conditions of mobility.

The motion of an entity can basically be divided into two kinds: Mobile and stationary. A mobile entity is one that is moving in space in one of various ways, whereas a stationary entity is residing at a fixed place without moving. A distinction between mobility and immobility may be difficult to make, for instance when the usually stationary work of a lathe operator is interrupted by walks to the storeroom or the foreman. Kristoffersen and Ljungberg have proposed a subdivision into what they call modalities of mobility that takes these kinds of interruptions into account [KrLj98]. We adopt their division of mobility into the subtypes of travelling, wandering, and visiting (see Figure 18), but make some modifications and additions to the definitions of these terms given by Kristoffersen and Ljungberg where necessary.

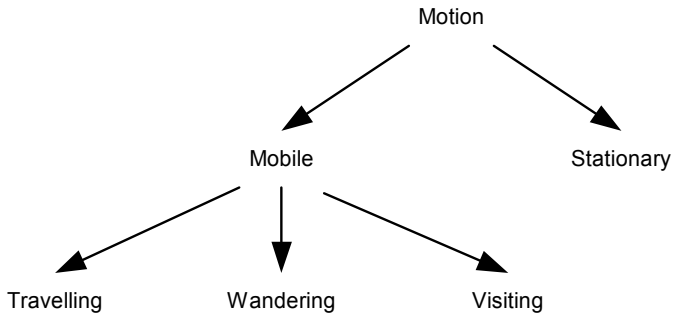

Figure 18: Subdivision of motion

The motion of travelling according to [KrLj99] is »*the process of going from one place to another in a vehicle*«. With this type of mobility Kristoffersen and Ljungberg aim »*to capture the mobility of people that go in vehicles*«. Examples of people performing the motion of travelling are salespersons driving to a customer's, commuters on their way to and from work, or people going on vacation. Travelling persons may be driving the vehicle they travel with themselves, or they may be passengers. This influences their available and useable communication media as well as the information that is desired by them, its amount and delivery channels. Although we agree with this definition in principle, we do not consider the postulate that travelling only occurs when an entity is situated in a vehicle as accurate. In our opinion this view is too restricted, and the motion of travelling should be extended to the whole process of beginning a journey from a starting point to reaching a destination without consideration of the means of transport employed, interruptions, or rests that occur. A person going on vacation, for example, is still travelling when being at the airport and waiting for her flight.

Wandering is defined as »*extensive local mobility in a building or local area*« (loc. cit.). Wandering is the type of motion with the highest degree of personal mobility. An example of a person performing the motion of wandering is a member of a company's IT management staff who walks around supporting other people.

The third type of motion, visiting, is executed when a person is temporarily spending time at a certain place and afterwards is moving on to another place. Although not explicitly mentioned by Kristoffersen and Ljungberg, the examples they give – a consultant visiting a client, a researcher giving a guest lecture in another university, etc. – make clear that the motion of visiting implicitly is based upon the assumption that there is a home location, i.e. a location a person has to be away from to be assigned the motion of visiting. A person can only be visiting when away from her home location, whereas the presence of the person at any place within her home location, in contrast, is not regarded as a visit. Both the place a person lives at and the place she works at can be considered the home location. These places are stationary locations where the person is situated for a coherent period of time on a regular basis. In an application providing support for either work or leisure time only, the home location is always set to one of these two locations and is unalterable. For applications that cover various spheres of life, however, the question of which of these location is to be the home location arises. Depending on various factors such as time of day, special events like vacations or special shifts, and the former whereabouts and activities of a person one of the two potential home locations could be considered the only valid home location for a given point in time. An alternative approach is to regard both places together as the home location and to not consider special cases such as an office worker paying a visit to her company for a talk with colleagues during her vacation or going home for a short time to let workmen in during working hours as visits. In the context of our work we define both the home location together with the work location of a person as the home location. Our main reason for this decision is that the gain in information achieved by distinguishing between the two home locations is extremely small and hardly ever needed and does not make up for the complexity of the determination which of the two potential locations is the home location at any given moment.

From our point of view and in contrast to Kristoffersen and Ljungberg the distinction between travelling, wandering, and visiting cannot be made on the assumption that at a given point in time an entity's motion is exclusive. When visiting, for example, a person may be sitting together with a customer, which means that the person is both visiting and at the same time stationary. During a journey, i.e. when travelling, a person may be on her seat in a train or wandering around an airport. In these cases two different motions are performed simultaneously. These two motions are characterized by the fact that one of it is executed during a period of time which covers the other motion as well. Hence, we distinguish between what we call basic and temporary motion, with the temporary motion being executed and finished while the basic motion is still going on.

	Stationary	Travelling	Wandering	Visiting
Stationary	n.a.	not possible	not possible	not possible
Travelling	possible	n.a.	possible	possible
Wandering	not possible	not possible	n.a.	not possible
Visiting	possible	possible	possible	n.a.

Table 1: Combinations of basic and temporary motion

Not every kind of motion allows for a simultaneous execution of another motion. It is quite obvious that when stationary an entity cannot be mobile in any way at the same time. Moreover, no motion can be performed temporarily during wandering; this would cause the basic motion to change. But, as we have already mentioned, a person can be stationary, wandering, or visiting when travelling, and it is also possible to be stationary, wandering, or travelling – for example when driving with a customer to a building site during a visit to the customer's – when visiting. The possible combinations of basic and temporary motion are listed in Table 1.

The subdivision of mobility into travelling, wandering, and visiting has been criticized for focusing too much on the main activities executed while being in motion and not sufficiently taking into account aspects related to the process of being mobile [Wibe99]. Surely the mere distinction between the described types of motion alone does not provide enough information to make extensive inferences about how an entity is moving. Therefore, our approach is to avoid these shortcomings of Kristoffersen's and Ljungberg's classification in two ways. On the one hand, our model is not restricted to assigning only a type of motion to an entity and to assuming this as the entity's only activity. Instead, it is capable of representing any additional activities an entity is carrying out and of building up a hierarchy of nested activities. The next section gives details about the modelling of the state element of activity. The other distinctive feature of our motion model is its ability to further characterize an entity's motion by a number of additional attributes as explained below.

The model for motion is illustrated in Figure 19. An entity's state, represented by an object of the `State` class, can be described by a `Motion` object if information about the entity's motion has been detected or provided in an information demand profile. This object represents the basic motion the entity is executing. The value of this object's `type` attribute is a character string containing one of the possible types of motion explained above. If the motion type is either »travelling« or »visiting«, the `Motion` object may be associated with another object of the type `Motion` which represents the temporary motion performed during the basic motion. If there are attributes that further characterize the motion, they are grouped in an `AttributeSet` object which is associated with the `Motion` object.

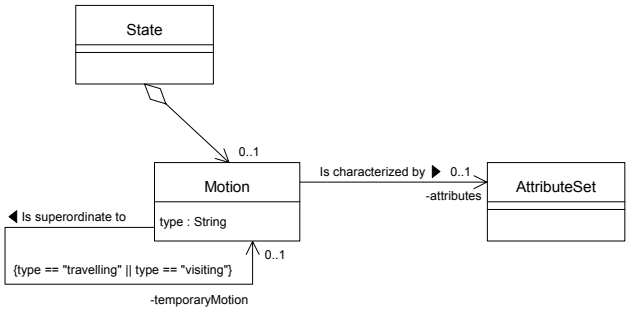

Figure 19: Motion model

The attributes that provide additional information regarding the motion of an entity can be manifold. In principle we can distinguish between general attributes each of the motions may possess and specific attributes which are only meaningful for particular motions. General attributes include the time the motion was begun at and the time it is expected to end at as

well as the direction the entity performing the motion is oriented towards. Attributes specific to one or more particular motions are, for example, the destination of a journey or visit, the project in the context of which a person is travelling or visiting, the means of transport used, the visited person or object, the purpose of a motion, etc. Since the number and the sorts of these attributes are virtually infinite and the attributes required in a specific application depend on the application's scope and purpose, it is neither possible nor desirable to define a complete set of possible attributes in advance. Instead, our goal is to provide a flexible and generic way of representing any possible attributes and to enable an easy integration of further attributes that may become relevant in the future.

For this reason our state model contains a generic template for the creation of attributes which is depicted in Figure 20. It is based upon an abstract `Attribute` class containing the attributes `name` and `description` that each `Attribute` object possesses. This class is furthermore associated with an `AttributeValue` class which represents the values of an attribute. The cardinality of this relationship is one to many, because an attribute may possess several values as, for example, an attribute named duration which is composed of the values for the start time and the end time of the motion it refers to. In the `AttributeValue` class the name, data type, and actual value of an attribute value are stored. By this means additionally required attributes can dynamically be defined as needed by simply instantiating a new `Attribute` object and setting the appropriate values for its name and description and its values' names, types, and contents. An alternative approach would be to dynamically create new subclasses of the `Attribute` class on demand as described in [LiYe99]. Due to the high complexity of this solution, however, we have dismissed this alternative. As a means to describe allowed values for attributes and to check whether a given `AttributeValue` object is a valid value for an attribute the `Attribute` class is furthermore associated with the abstract class `ValidityCondition` we have already explained in Section 4.1.3.

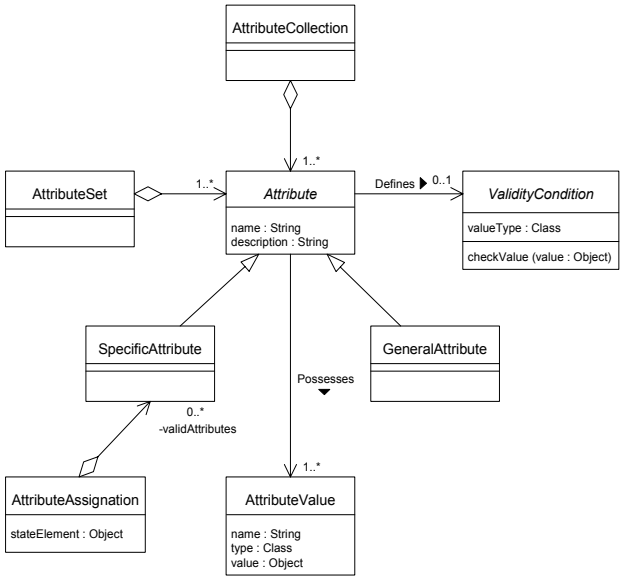

Figure 20: Attribute model

The `Attribute` class is the superclass of the `GeneralAttribute` and `SpecificAttribute` classes representing the abovementioned attributes suitable for every state element or for particular state elements only. Since specific attributes may only be attached to a subset of all state elements, the `AttributeAssignation` class serves as a means to represent these assignments between state elements and attributes. It therefore contains an object which is a state element and an association to the `SpecificAttribute` class by which the valid attributes that may be attached to this state element are referenced. Moreover, an `AttributeCollection` class is contained in the model in order to manage all attributes available in an application. We have been using the general term state element instead of motion during our explanations in this section, because attributes are also made use of to further characterize other elements of state as well. The modelling of attributes described here is therefore referred to in the following sections.

4.2.2 Activity

The state element we examine in this section is the activity of entities. In simple terms an activity denotes what an entity is doing. For most applications an entity's activity is the most important information about the entity's state. On the one hand, the activities being performed determine the information that is of particular usefulness to an individual. Consider, for instance, a consultant writing a report concerning a particular project. During this activity information objects such as e-mails or documents regarding the project are particularly relevant to the consultant. Accordingly, they should be presented to her in a prominent manner, e.g. by sorting the e-mail folders, highlighting certain services on the desktop, and notifying the consultant about document changes or incoming mails that affect this project. Another aspect of the influence of activity on information supply is that it also determines the extent to which information is considered a disturbance by a person. During certain occupations a person often does not want information to be delivered to her. This wish may refer to any information or only to particular pieces of information that are considered irrelevant and unwanted for the activity being carried out. For example, most businesspeople feel disturbed by incoming messages while conferring with colleagues or while having lunch with a customer. Yet, they may wish to receive very urgent messages if they concern the topic they are talking about. An entity's activity thus is a significant parameter in information demand profiles.

The examination and modelling of human activity has been a research topic in psychology, and a major theoretical foundation that exists in connection with this topic is activity theory. Activity theory is rooted in Russian psychology, and the work commonly associated with this theory is from Vygotsky [Vygo78], Luria, and Leont'ev [Leon78]. The main principle of activity theory is the unity of consciousness and activity, meaning that human activity generates consciousness. According to activity theory human activity is mediated by tools. These tools can be of various types; they can be both material and immaterial as, for example, language. Hence, an activity is composed of a subject, an object, and a mediating tool. The subject is a person or a group carrying out an activity, and the object is understood in the sense of an objective that motivates an activity. Humans are always part of a community, and this relationship is mediated by social rules like norms or conventions. In the community there are one or more people who share an object (or objective) which leads to a division of labour. The divi-

sion of labour indicates the distribution of tasks between members of the community. Therefore, later evolutions of activity theory have added the elements of rules, community, and division of labour to the view of human activity [Enge87] as depicted in Figure 21.

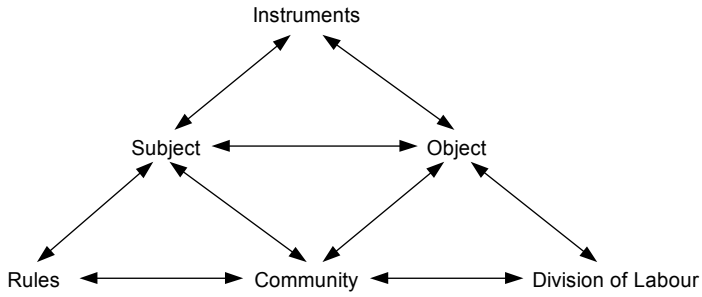

Figure 21: View of human activity after Engeström [Enge87]

Examining the concepts developed in activity theory we can state that this theory provides a widely used framework which, according to Nardi [Nard96], is one of the richest approaches for studies of context. This is due to its comprehensiveness and its consideration of elusive issues of consciousness. In CSCW research activity theory has been applied as a theoretical foundation for the design of workflow and groupware systems [FjLa02], [Bard98]. Yet, the main reason that makes this theory difficult to use for our purpose is its rather abstract and complex construction. With their roots in psychology and sociology, some of the elements of activity like social rules or division of labour are of minor relevance to software systems that support intelligent information supply. Furthermore, these elements, especially the community a subject belongs to as well as the division of labour, are reflected in role concepts or are administered by the profile management of information logistics and therefore are beyond the scope of the Context Component. As a consequence, our model of activity focuses on individuals, their motives, the – mainly material – tools employed to carry out an activity, and a description of what is being done by an entity at a given point in time. This corresponds to the upper part of the triangular depicted in Figure 21.

Furthermore, we recognize the need to represent a hierarchical arrangement of activities which corresponds to Leont'ev's distinction between activities, actions, and operations. While Leont'ev's main intention of this distinction is to separate collective activity from individual action, a hierarchical arrangement of activities in our approach is determined by a logical containment in the sense of a part-of relationship. A nested activity, i.e. an activity that is contained in a hierarchically higher activity, represents a complete act in itself, but it is only a part of what needs to be done to finish the containing activity. This also implies a shared objective and, although less important, an encapsulation of the activities with regard to time. A hierarchy of activities is, for example, given when a businessperson is taking notes while attending a presentation which again takes place while working. In our model, however, we do not make use of the terms action and operation explicitly. Instead, we speak of activities in any of these cases and represent what corresponds to Leont'ev's classification by a nesting of activities into one another. In addition, an entity may be carrying out several parallel activities of equal rank at the same time and with possibly different objectives, for instance when a person is listening

to music while reading. In particular circumstances some of these parallel activities may exclude each other in the respect that they cannot be carried out simultaneously. Since these exclusions, however, are highly dependent on the specific economic, social, technical, and/or personal conditions of entities and within companies employing an information logistic application, it is not possible to exclude a simultaneous execution of particular activities in advance. To give an example it is not uncommon for businesspeople to be flying and sleeping at the same time, whereas a pilot, however, will most likely be prevented from performing these two activities simultaneously. In Section 4.5 we describe how this interdependence among context elements' attribute values is represented in our model in a general manner. Furthermore, Chapter 5 provides mechanisms to ensure that only those combinations of activities which are permissible in a particular application and for a specific entity can be created during the context gathering process.

The number and types of activities entities can perform are infinite. Since the activities relevant to a particular application furthermore depend on the respective application area, it is not sensible to preset specified activities. Therefore, in our activity model depicted in Figure 22 we pursue an approach similar to that employed for attribute modelling introduced before.

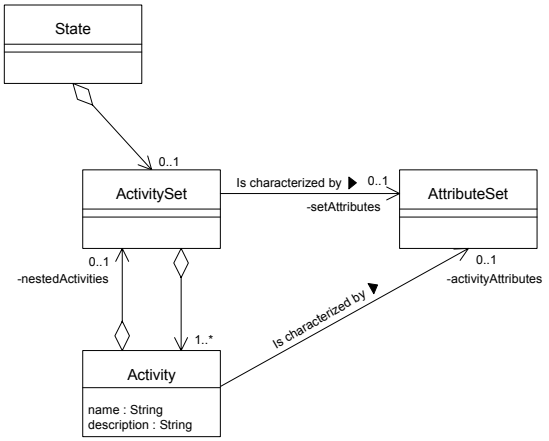

Figure 22: Activity model

Activities are represented by objects of the `Activity` class. They possess the attributes `name` and `description` which are character strings. An activity's name acts as its identifier; its description provides an additional annotation. The names of activities are expressed by verbs in the present participle tense as this grammatical form is used to express present existence, action, or occurrence in relation to the time indicated by the finite verb in its clause in most languages. In addition to singular activities, the model defines `ActivitySet` objects which serve as containers for activities of equal rank that are carried out simultaneously. An object of the aggregate class `State` may contain zero or one `ActivitySet` object. Nested activities are modelled by an aggregate relationship between an `Activity` object and an `ActivitySet` object containing one or more parallel activities that are part of the hierarchically higher `Activity` object they are associated with.

As mentioned in the previous section, the concept of attributes that provide further information about the individual elements of an entity's state is also made use of in the model for activity. Both the singular `Activity` objects and the activities grouped in an `ActivitySet` object may be associated with zero or one `AttributeSet` instance. An `AttributeSet` object associated with an `ActivitySet` object contains characteristics referring to all activities contained in the `ActivitySet` – and therefore is required to contain only objects of the `Attribute` class which are applicable to all activities –, whereas an `AttributeSet` object associated with a single `Activity` object provides information referring to the particular activity only. The general attributes that may be associated with any activity include the activity elements objective and tools identified in activity theory. Furthermore, the duration of an activity or of a parallel execution of more than one activity belongs to this type of attributes. Specific attributes may comprise the project a person is working on, the subject of an e-mail being written, the direction in which somebody is looking, the title of the book a person is reading, and many more.

4.2.3 Physical condition

The explicit consideration of entities' physical condition is particularly relevant to certain types of information logistic applications, for example in the areas of healthcare, wellness and fitness, or the protection of health and safety standards at work. In these types of applications data about the bodily functions of people are an important indicator that needs to be evaluated in order to provide desired information. This may include suggesting appropriate means in a fitness or diet programme, reminding people of their daily medicine, or sending alerts if a person is detected to be in a possibly harmful physical condition. Physical conditions may also play a role in conjunction with objects such as machinery, utensils, or other kinds of equipment. This is the case when there is no content service available in an information logistic application that processes the data gathered from these objects, i.e. when the data gathered from these objects are not to be supplied to users themselves, but only serve as a parameter to determine the supply of other information objects. An example of this is an information supply to sewerage workers who are sent a notification whenever a pipe is blocked, i.e. when the physical condition of a pipe has been detected to possess the value »blocked«.

The development of a model for the physical condition of entities also raises difficulties originating from the unpredictability and unlimitedness of the data concerning bodily functions we have already been confronted with in the models for motion and activity. Therefore, physical conditions are again modelled in a generic and flexible manner as illustrated in Figure 23. The `PhysicalConditionSet` class groups individual `PhysicalConditionType` objects that represent particular subsets of an entity's physical state. They possess a `name` and a `description` attribute, thus allowing to subsume certain bodily functions such as circulatory or respiration data in a single object of the `PhysicalConditionType` class. The individual data measured for a `PhysicalConditionType` object are represented by instances of the `ConditionValue` class. This class possesses the same attributes as the `AttributeValue` class that is part of the attribute model introduced above; it only differs from this class by its purpose, meaning that the values of its attributes only indicate physical conditions. There is also an association of the `PhysicalConditionType` class with the `ValidityCondition` class that manages the allowed values for the `ConditionValue` objects belonging to the

bodily functions represented by an instance of the `PhysicalConditionType` class. We do not consider additional attributes necessary for this state element, because the information about an entity's physical condition provided by the `PhysicalConditionType` and their associated `ConditionValue` objects is sufficiently expressive itself.

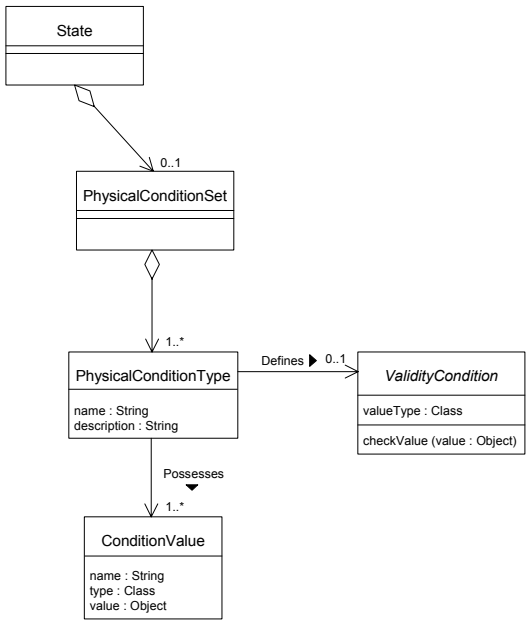

Figure 23: Physical condition model

4.2.4 Emotional condition

The fourth element of state we investigate is the emotional condition of persons. Although the emotions of an application's users are still paid little attention to in many areas of computer science, there are significant opportunities to improve information supply to individuals by explicitly considering their feelings during the design and development of IT systems.

The areas in which users' emotions may affect the supply of information principally include the design of user interfaces and the kind of information and services offered to the users. Most people are more or less frequently frustrated while using computer systems. Even the ongoing efforts to improve the usability of information technology have not been able to eliminate these feelings of frustration, and we can predict that this will hardly happen in the near future no matter how much attention is paid to creating user-friendly devices and software. Therefore, since a complete elimination of users' frustration is unforeseeable, applications need to provide a compensation for this inevitable feeling by adequately responding to the dissatisfaction of users. This behaviour is a means to improve the acceptance of applications and to adapt the way information and services are presented to users' particular needs and experience levels. An example of a prototype application responding to users' frustration by techniques like active listening and empathy is the interactive affect-support agent developed by

the Affective Computing Research Group [KlMo02]. In addition to taking into account feelings of frustration, the consideration of other emotional conditions individuals may be in is also important to achieve an optimized information supply. A person's feelings affect the way she works or learns, for example, or the type and amount of information she desires. An application supporting electronic learning, for instance, can offer a considerably improved service by adapting the speed it moves on in a course and the contents of a lesson to the fact whether the student feels tired, asked too little of, overtaxed, or full of impetus.

Similar to the state element of activity the examination and modelling of human emotion has mainly been investigated in psychology. The findings gained by psychological analyses of emotion include Green's belief-desire theory [Gree92] or the communicative theory of emotions by Oatley and Johnson-Laird [OaJo96]. A complete examination of the works on human emotion done in the area of psychology goes beyond the scope of this thesis; the dissertation by Wright [Wrig97] provides an overview of the psychological foundations of human emotion. All in all, the concepts and models developed in psychology focus on neurological stimulus processing and abstract representations of emotions in the human mind. Therefore, the findings of psychology are too abstract and complex to serve as a basis for a model for emotions to be used in computer applications. Although there has been continuous effort in the development of IT systems to make the systems more and more powerful and intelligent, the aspect of capturing, processing, and responding to users' emotions has hardly been gaining attention in computer science. Mainly in the context of artificial intelligence and, more recently, in software agents research this topic has been dealt with. Much of the work that has been done is based on a model described by Ortony, Clore, and Collins in 1988 [OrCl88]. The Oz project at Carnegie Mellon University, for example, aimed at building an architecture for agents that exhibit social behaviour including the support of emotions [BaLo94]. We have already mentioned the more recent research dealing with human emotion called Affective Computing [Pica00].

The models for emotion made use of in these and other systems [BaBr99], [Kope01] have in common that they define a number of discrete emotion types. Due to the variety and complexity of human feelings we, too, consider this an appropriate way to model human emotion. In contrast to the pre-defined set of emotion types employed by most other approaches, however, we again pursue a more flexible approach that allows for a dynamic extension of the emotions processable by an application and thus can be applied to various application areas.

Our model for emotional condition is shown in Figure 24. Since a person may be feeling more than one emotion at a time, the individual `EmotionalCondition` objects are grouped in an `EmotionSet` object. The `EmotionalCondition` class possesses two attributes specifying the name of an emotion and the intensity it is felt with. The intensity of an emotion is stated as a floating point number in the range from greater than zero to one. The greater the value of the `intensity` attribute is, the stronger the emotion is felt by the person. Emotions may be classified into categories; an example is to distinguish between positive and negative emotions or between feelings that are beneficial to a certain task and those that are a hindrance. Therefore, an `EmotionalCondition` object may be associated with one or more `EmotionCategory` objects which represent these categories of feelings. In the modelling of emotions, too, we make use of additional attributes providing further information about emotional conditions that does not belong to the emotional conditions themselves. At present time, the means to detect most of those attributes of emotions such as reasons for certain feelings,

objects an emotion is directed towards, or the duration of an emotion or a set of emotions are limited. Nevertheless, we associate both the `EmotionSet` and the `EmotionalCondition` classes with the `AttributeSet` class we have already described to allow for a representation of attributes that further characterize individual emotions and sets of emotions in our model.

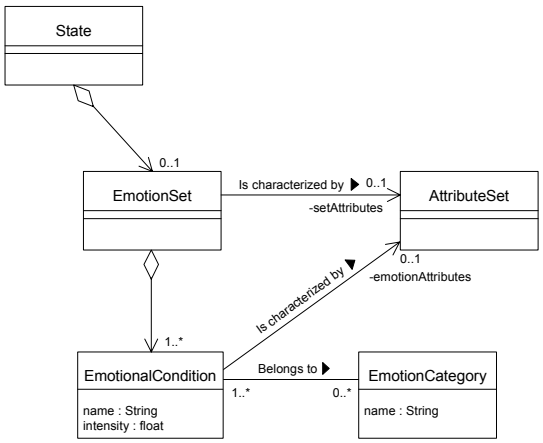

Figure 24: Emotion model

4.2.5 Summary

In the previous sections we have presented a model for the context element of state with its elements motion, activity, physical condition, and emotional condition. This section sums up the characteristics of the models for these four elements and provides a brief assessment of our state model which is illustrated in Figure 25 on page 82.

First, we have argued that the ways in which people are mobile have to be examined in order to suit information supply to users' needs. Following Kristoffersen's and Ljungberg's subdivision of mobility, we have identified the basic motion patterns an entity can be in, stationary and mobile, the latter being again subdivided into travelling, wandering, and visiting. These kinds of motions have been explained, and – where necessary – modifications and additions to Kristoffersen's and Ljungberg's classification have been made. These include the addition of a temporary motion which is performed and finished while another motion is going on. Based on this we have introduced a model for the motion of entities which represents the different kinds of motion by the type attribute of a `Motion` object. Since the mere representation of the different motion types is not sufficiently expressive, attributes that serve to further characterize a state element are associated with every motion. We have distinguished between general attributes which each kind of motion may possess and specific attributes which are only meaningful in conjunction with particular motions. Since the number and sorts of attributes required by a particular application cannot be foreseen in their entirety, we have developed a flexible and extensible way of representing any possible attribute. Our model contains an abstract `Attribute` class which provides the attribute's name and description. This class is associated with a class representing the values an attribute may take on. General and specific

attributes are represented by subclasses of the abstract `Attribute` class. To ensure that only valid specific attributes are assigned to a state element the model furthermore includes a class that contains the permissible assignments between state elements and attributes.

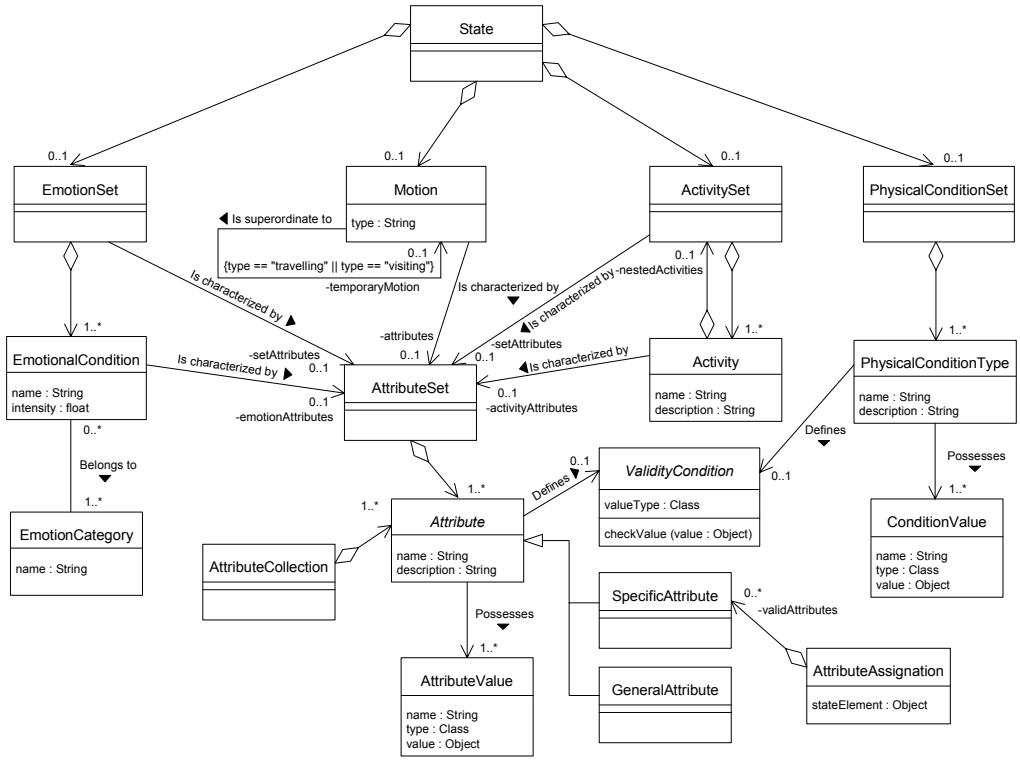

Figure 25: Overall state model

After that we have introduced a model for the state element of activity. As a basis for our model we have described and examined the results gained by the research area of activity theory. Our examinations have led us to the conclusion that only a part of the activity model proposed by activity theory is suitable for our purpose. We have introduced our model for activity which represents singular activities by objects of an `Activity` class with attributes for its name and description and furthermore allows for a grouping of several activities into sets. The requirement for a representation of hierarchically arranged activities we have pointed out is met by an aggregate relationship between activities and activity sets the elements of which have the role of so-called nested activities. Like in our motion model individual activities as well as activity sets may be further characterized by a number of attributes.

Subsequently, we have dealt with the state element of physical condition. Like the models developed for the state elements of motion and activity a model for the physical condition of entities also needs to take into account unforeseeable and possibly infinite data about bodily functions that may be relevant. Therefore, physical conditions are again modelled in a generic and flexible manner. The individual `PhysicalConditionType` objects that represent particular subsets of the physical state of an entity are grouped in a set. `PhysicalCondition-`

`Type` objects serve to put together certain bodily functions such as actions of the heart or circulatory data in a single object. The individual data measured for a `PhysicalConditionType` object are represented by instances of the `ConditionValue` class. The modelling of physical condition does not contain any additional attributes as the information provided by the `PhysicalConditionType` objects and their associated values alone is sufficiently expressive.

In the following section we have described the influence people's emotions have on the information they require. After a brief glance at the findings of psychology regarding human emotion and an overview of some related areas and projects in computer science, we have presented a model for the emotional condition of persons. Emotions are represented by objects of the `EmotionalCondition` class and possess a name and an intensity. Since emotions may occur simultaneously, the `EmotionalCondition` objects are grouped in a set. The `EmotionCategory` class has been introduced as a means to classify emotions; it is therefore associated with the `EmotionalCondition` class. Again our model for emotional condition comprises additional attributes that further characterize emotions.

The outstanding features of the object model for state include the ability to represent the individual elements of state that are relevant to information logistics in a generic and flexible manner. The different motion patterns of entities as well as the activities entities are carrying out and their physical and emotional conditions can be captured by our model. The complexity and high degree of abstractness these state elements are subject to by nature are taken into account by a model design that is not limited to a fixed and pre-determined set of permissible values, but instead allows for a dynamic integration of those types of the individual state elements that are required in a specific application. Furthermore, the concept of providing additional information concerning the state elements of motion, activity, and emotional condition by means of an extensible set of attributes leads to a high degree of expressiveness while at the same time preserving a separation of concerns between the actual state elements and the additional details that describe them more precisely.

The main advantages of this state model for information logistics are:

- It is possible to represent various types of the individual state elements and of attributes in the model, thus making it generally applicable to any application domain.
- Since state elements can be described as detailed as required, various degrees of complexity and precision are supported.
- The state model can easily be extended to application-specific types of activity, physical and emotional condition, and attributes.
- The representation of state is independent of sensor systems and therefore allows for an integration of various sensors that gather an entity's state.
- The ability to define attributes and types of a state as needed and to refer to them by their names makes the model very convenient to use and is beneficial to a high degree of user acceptance.
- Referring to state elements by their names facilitates the conversion of model data into a human-readable representation as well as access control to state elements.

The disadvantages resulting from the design of the model are the following:

- The state elements and attributes made use of by a specific application depend on the application domain. This may lead to a large number of objects representing state elements and attributes which have to be created and managed.
- The high flexibility and expressiveness of the state model can only be achieved at the expense of an increased model complexity which results in a risen amount of data and computations.
- A separate mapping of sensor data into state elements processable by the application is often required.

4.3 An Object Model for Reachability

According to our definition the reachability of a person denotes the sum of all communication media that are at her disposal and usable for communication at a given point in time. To optimize information supply an information logistic application has to select the most suitable communication medium for each delivery of information in consideration of the parameters defined in demand profiles as well as aspects of technical capabilities, cost, security, usability, and so on before the actual delivery can be carried out. It furthermore depends on the communication media's current status if they can be used for information delivery; they may be switched off, busy, defective, or their usage may not be allowed in certain circumstances.

A person's reachability is composed of the elements available devices, applications, and communication protocols. The term device is used to denote all hardware appliances a user can send and/or receive information with, including peripherals for the processing of this information. A communication protocol is a means to transmit data from a sender to a receiver, and the term application denotes a program providing a user or, if applicable, another application with functionalities needed to process information that has been delivered or made available.

Models for these elements of reachability already exist in parts as we have already mentioned in Chapter 3, in particular the Composite Capability/Preference Profiles (CC/PP) framework [ReHj99] and the Wireless Application Group User Agent Profile Specification [WAP01f] developed in connection with the Wireless Application Protocol (WAP). Yet, we have recognized the need to develop a separate model for reachability which is interoperable with existing approaches, but provides a more detailed and generally applicable approach.

Devices possess varying technical capabilities as regards storage capacity, display size, input facilities, and so on. Similar heterogeneity exists on the part of communication protocols that can be used for data delivery and on the part of the applications available on a device for information processing as well. An object model for the context element of reachability represents the capabilities each of the three elements of reachability possesses. Since there is a continuous progress in the development of communication media, however, the extensibility of the reachability model needs to be paid special attention to. It cannot yet be predicted which currently unknown characteristics communication media will possess or which present attributes of them may become irrelevant in the future. Therefore, our approach is to provide a generic model for reachability that allows for a flexible definition of the relevant characteristics of com-

munication media for any particular state of technological development. In parallel with this we present an instance of the reachability model which describes the current state of technology and can be employed in current information logistic applications.

4.3.1 Generic reachability model

When an information supply is to be carried out, at least one device has to be available to the receiver she can be delivered the information to. In addition, in order to establish a connection between a user's device and an information logistic application several communication protocols can be used. There are interdependences between communication protocols and devices as well as applications, because many applications or devices only support a limited number of communication protocols, and the availability of a communication protocol alone does not make a user reachable. As a result, the most suitable communication protocol for an information supply can only be selected taking the available devices and applications into account. Since the information that is to be transmitted to a user possesses particular characteristics as well, the individual pieces of information, too, pose certain requirements regarding their processing onto communication media. An information supply cannot be considered successful until the user has not only received a piece of information on a certain device via a certain communication protocol, but is also able to access and use this information. For this purpose several applications can be used. Applications also depend on the available devices and communication protocols.

Although we aspire to achieve the greatest possible generality in our model for reachability, we believe that devices, communication protocols, and applications are indispensable elements of reachability and will continue to be so for a long period of time. Therefore, it is always a combination of these three elements that constitutes a communication medium. In the reachability model the characteristics of devices, communication protocols, and applications are represented in order to provide other components with data that can be compared to the properties of the information that is to be transmitted and to users' preferences and demand profiles. This comparison results in a selection of the most suitable combination of device, communication protocol, and application for an information delivery and/or to an execution of an appropriate transformation of information into a suitable format.

As already mentioned, the relevant characteristics of next generation communication media cannot be foreseen. For this reason we model communication media's characteristics in a generic and extensible way using the same approach as in the model for the attributes of state elements. The generic object model for reachability is illustrated in Figure 26. A user's reachability, represented by an object of the `Reachability` class, consists of zero to many `CommunicationMedia` objects. When there is no `CommunicationMedia` object contained in a `Reachability` object, this means that there is no way to electronically deliver information to the user at the given point in time. As explained above, each `CommunicationMedia` object consists of the elements device, communication protocols, and applications. The central element of a `CommunicationMedia` object is the class `Device` as devices, in contrast to communication protocols and applications, are the only element of reachability the usage of which is not dependent on the other elements. Therefore, a `CommunicationMedia` object aggregates exactly one `Device` object and at least one object of each the `Communication-`

Protocol and the Application classes. We have chosen the plural form for the name of the CommunicationMedia class, because the objects of this class can each represent more than one way to reach a user if there are more than one CommunicationProtocol or Application objects contained in them.

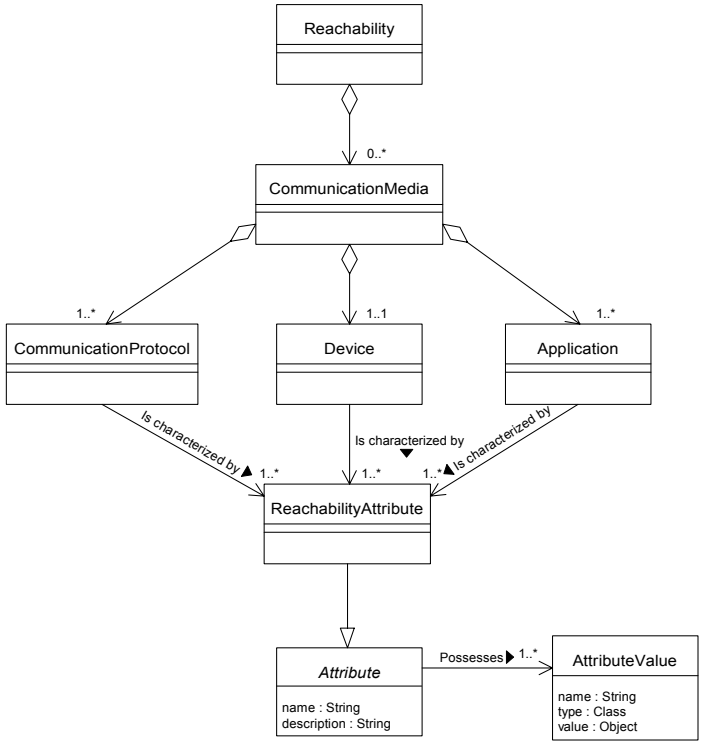

Figure 26: Generic reachability model

It is often useful to group the characteristics of devices, communication protocols, and applications into categories as, for example, supported security mechanisms of communication protocols or storage capacities, input or output facilities of devices. Therefore, we create another subclass of the abstract Attribute class explained in Section 4.2.1. Each object of this subclass called ReachabilityAttribute serves to represent a group of characteristics of the reachability elements. It inherits from its superclass an association to the AttributeValue class representing the names, types, and values of the specific characteristics belonging to this group. The classes Device, CommunicationProtocol, and Application are associated with one or more objects of the ReachabilityAttribute class which contain the relevant groups of these classes' characteristics. If a specific characteristic cannot meaningfully be assigned to a group, it is represented by a ReachabilityAttribute object with its value stored in the associated AttributeValue object. This representation may cause a redundancy of the characteristic's name in both objects if the value of the characteristic consists of a single datum, and it leads to a more intricate access to the characteristic's value. However, we believe that the flexibility gained by separating the groups of characteristics from their actual values, especially concerning extensibility, compensates for this drawback.

In a specific instance of the generic reachability model the available `ReachabilityAttribute` objects and their associated `AttributeValue` objects' names and types need to be managed to ensure that all relevant characteristics of communication media are known and that only these characteristics can be assigned to devices, communication protocols, or applications. This can be accomplished by maintaining containers for both the names of the available groups of characteristics and for the assignment of names, types, and contents of values to each group of characteristics. Based on this information requested objects of the `ReachabilityAttribute` and `AttributeValue` classes can dynamically be instantiated using the Factory design pattern [GaHe94].

Theoretically it would also be possible for the reachability model to not only provide the characteristics of communication media alone, but to also calculate indicator values which provide an assessment of particular aspects regarding a communication medium's suitability for information supply. These aspects may include the cost, duration, or usability of a communication medium for a specific user and a specific piece of information. For this purpose the `Reachability` class would have to offer methods that carry out the calculation of these indicator values. These methods would have to be passed particular parameters such as the size of a data file that is to be transmitted, its file type, or even user preferences or demand profiles. The parameters the methods would require may again change over time as technology evolves and in the future currently unforeseen parameters may determine the calculations of the indicator values. For this reason the methods would have to contain input parameters of a generic type in order to maintain the generality of the reachability model. This would cause the Context Component to be required to know about the structure of objects managed in other components such as output documents or users' preference profiles which would result in a tight coupling of components. Since this is a serious threat to the maintainability of information logistic applications, we have dismissed the idea of including a calculation of indicator values in our reachability model.

In addition to the characteristics describing communication media in general, there are further attributes relevant to the reachability model that are specific to the particular person whose reachability is described. These attributes refer to the execution of information delivery and contain the addresses required to set up a communication path between an information logistic application and a user. Such addresses may be e-mail addresses, telephone or fax numbers, IP addresses, and so on. Since they differ for each user of an information logistic application, they cannot be modelled as attributes of the communication media alone, but have to be seen in relation to the user. For reasons of clarity we illustrate them separately in Figure 27.

The addresses a user can be reached at depend on both devices and communication protocols. A user can be sent information to a particular device via more than one communication channel with each of the channels possessing a different address. Consider, for example, a cellular phone with different numbers for voice and fax communication. On the other hand, a communication protocol's address may be usable on various devices as, for instance, an e-mail address. Therefore, these addresses are not associated with a device, communication protocol, or user alone, but instead are attributes of the relation between users, devices, and communication protocols. In the reachability model addresses are represented by objects of the `AddressValue` subclass of the `AttributeValue` class. Due to the particular connection of the addresses with users, devices, and communication protocols the `AddressValue` class is

an association class of the ternary relationship between the classes `User`, `Communication-Protocol`, and `Device`. This special subclass for addresses is required in order to separate the associations to the `User`, `Device`, and `CommunicationProtocol` classes from the `AttributeValue` class as they are not needed for the representation of the general characteristics of communication media.

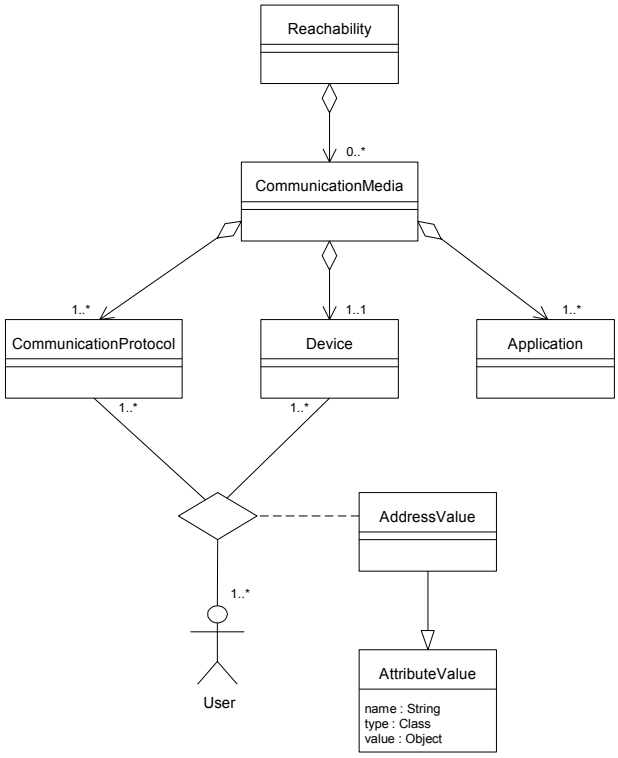

Figure 27: User-related attributes of the reachability model

4.3.2 Instance of the reachability model for current use

In this section we exemplarily present an instance of our generic reachability model that corresponds to the current state of technology. Today's communication media possess a variety of features in which they differ from each other. In the following we identify groups of communication media elements' characteristics – which are represented by `ReachabilityAttribute` objects –, and in addition we describe the individual distinguishing features these groups of characteristics consist of, corresponding to `AttributeValue` objects. The groups of current devices' characteristics are:

- Basic device characteristics

  This group of characteristics contains universal attributes that any device possesses and that describe some of the device's general properties. It consists of attribute values referring to a device's model, manufacturer, and CPU.

- Device type

  Devices can be subdivided into various types. Common device types are stationary computers, CarPCs, notebooks, Personal Digital Assistants (PDAs), cellular phones, pagers, or fax machines. Depending on the application area other special types of devices such as industrial bar code scanners or check-in terminals may also be relevant. The list of possible device types therefore is application-specific and may furthermore change over time as new types of devices are frequently introduced. Although the type of a device does not consist of several attributes, but of a singular value from a preset enumeration of device types, we model the device type as a separate `ReachabilityAttribute` object as the significance of this device characteristic differs from the other device features.

- Dimensions

  This group of characteristics contains attributes regarding the physical size of a device, its length, width, height, and weight. Especially for portable devices these data are important characteristics that affect the usability of these devices in mobile settings.

- Type of information processing

  Regarding the supported types of information processing we distinguish between devices that are limited to only receiving information and those that allow for a creation and modification of information as well. Devices of the former type are suitable for a delivery of information that must not or need not be modified by the recipient as, for example, notifications or news. Devices that allow for receipt, creation, and modification of information enable the user to actively create and change messages, documents, tasks, or other data. These devices therefore possess some kind of input facilities as explained below.

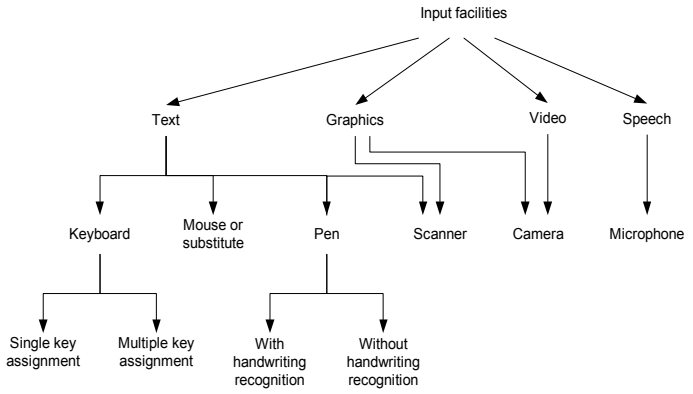

Figure 28: Current input facilities of devices

A device's input facilities enable the user to enter data on the device. Figure 28 shows an overview of common input facilities current devices possess. A combination of different pieces of input equipment is often to be found on a device. For this reason there is first of all a group of

characteristics containing the available pieces of input equipment of a device. In addition, the individual types of input equipment available on a device are represented by separate groups of device characteristics.

- Available input facilities

  The values this group of characteristics contains are the names of all input facilities available on a device. Except from mouses which do not possess any further relevant attributes, each of these names represents an individual group of characteristics which contains the specific characteristics of this piece of input equipment as described below.

- Keyboard

  For devices which possess a keyboard the dimensions of the keyboard (length, width, height, and weight) as well as the length and width of the individual keys are relevant attribute values belonging to this group of characteristics. Further attribute values indicate if the keyboard is capable of accepting alpha-numeric text or if it only accepts the entry of digits and if its keys can programmatically be assigned different labels and functions.

- Pen

  The term pen is used to denote any pointing equipment employed to enter data on a device by physically contacting the device's screen. The group of characteristics referring to pen input equipment contains an attribute value indicating if handwriting recognition is supported. Moreover, the pointing resolution of the pen, expressed as an enumeration type, is another attribute value belonging to this group.

- Scanning device

  The relevant attribute values of a scanning device are the scanning technology used, its horizontal and vertical resolution, bit depth, and the dimensions of the device, its length, width, height, and weight.

- Camera

  The characteristics of a camera include its image sensor (supported resolutions and formats), the camera's illumination and lens, attribute values referring to the images created by the camera (frame rate, supported formats, compression, colour depth), the camera's hardware (chip and storage), its dimensions and weight as well as the connectivity features of the camera.

- Speech input

  The speech input facilities of devices are characterized by attribute values for the maximum recording time, the recording technology, and for a discrete rating of the speech input's quality, again expressed as an enumeration type.

A device's output facilities serve to present data to the user. Thus, at least one piece of output equipment has to be available on any device. The output of information may be in visual or acoustic form or by means of paper. A combination of different output facilities on a device is

common. Analogous to the representation of input facilities one group of characteristics contains the available pieces of output equipment of a device, and the individual types of output equipment are represented by separate groups of device characteristics as described below.

- Available output facilities

  In this group of characteristics the names of all output facilities available on a device are listed as attribute values. For each of these names there is an individual group of characteristics containing the specific characteristics of this piece of output equipment.

- Display

  Relevant characteristics of devices with a display for visual output are the display's size, its horizontal and vertical resolution, dot pitch, refresh rate, and the number of colours or greyscales that can be shown. In addition, there is an attribute value that indicates whether the display is capable of displaying images or whether it only supports textual output.

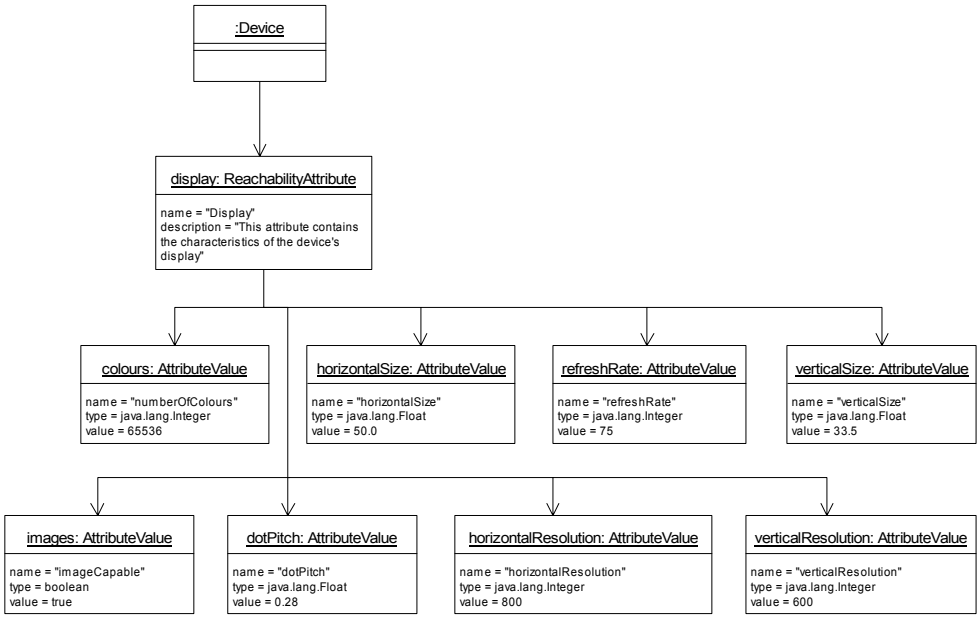

Figure 29: Exemplary representation of a display's characteristics

Figure 29 exemplarily illustrates the usage of our reachability model by depicting the representation of a particular device's display characteristics in the reachability model's instance for current use. The display itself is represented by a `ReachabilityAttribute` object; its attributes correspond to `AttributeValue` objects which contain the specific values of these attributes for the particular display. For those attributes that represent values expressing a unit of length or weight we have refrained from presetting a particular unit in the model. This is due to the fact that with regard to internationality country-specific settings like the unit of length used in an application should be stored as a central system setting.

- Audio output

  A relevant characteristic of a device's audio output equipment is its quality. Since the quality of audio output is usually not specified by technical measures, we represent it by a quality class with the available discrete quality classes expressed as an enumeration type.

- Print output

  The characteristics of output facilities for printing include their horizontal and vertical resolution, the printing technology used, the supported paper sizes, and the duration of a standard sized printout.

- Supported character sets

  This group of characteristics contains attribute values representing the supported character sets for both input and output, with each value being a character set name registered with the Internet Assigned Numbers Authority (IANA).

- Mobility

  Devices can be stationary or portable, the latter meaning that a user can take them with her to different places. This is indicated by an attribute value in this group of characteristics.

- Storage capacities

  The available storage capacities on a device significantly affect the size and complexity of the data that can be created, processed, and stored on the device. Since there are different types of storage media, there is a `ReachabilityAttribute` object for each type of storage available on a device whose `AttributeValue` objects state its type, the size of the storage medium as well as its access speed.

- Signalling facilities

  A device's signalling facilities indicate if and how it is capable of notifying the user about the arrival of new messages. In this group of characteristics therefore the types of supported signals which can be subdivided into acoustic, visual, and vibration signals as well as no support of signalling are represented by an attribute value whose type is an enumeration type.

- Energy supply and energy consumption

  Energy supply and consumption determine the duration a user may operate a device without having to recharge it. This is particularly relevant for devices without a permanent energy supply as it is the case with most portable devices. The operating time also depends upon the device's energy consumption which again is affected by the type of the device's utilization. This group of characteristics therefore includes an attribute value that indicates the type of the device's energy supply, permanent or temporary, and, for those devices with a temporary energy source, attribute values representing the average operating time both in case the device is being utilized as well as in case the device is in stand-by mode.

Below we describe the characteristics current communication protocols possess. In doing so, we only examine those characteristics that affect the selection of the most suitable communication medium for information supply. The groups of current communication protocols' characteristics are:

- Transmission speed

  The speed of data transmission is one of the main criteria to rate the capability of a communication infrastructure. It not only determines how long it takes to transfer information to a user, but it also sets the basic conditions for the kind of data that can be transmitted and the transmission cost. The transmission speed indicates the amount of informational events transmitted per unit of time, most commonly expressed as bits per second or a multiple of it. In our model the unit of measurement chosen for the attribute value of transmission speed is kilobits per second.

- Supported media types

  Information can be available in several media types [Wojc03]. In order to send a user information of a particular type a communication protocol that supports the transmission of this media type has to be chosen. We distinguish between the media types speech, data, text, graphics, video, and audio. The attribute values reflecting the supported media types of a communication protocol thus are selected from an enumeration of these types of media.

- Transmission quality

  One of this group of characteristics' attribute values rating the quality of data transmission is the error rate of a communication protocol. The error rate provides information about the amount of lost or corrupted data during a transmission. It is quantified by the ratio of improperly transferred data with the total amount of transmitted data, measured in bits, bytes, or blocks. In our instance of the reachability model the unit of measurement employed for a communication protocol's error rate is bits. As regards transmission quality it is furthermore relevant if a communication protocol supplies mechanism to detect and/or correct transmission errors. Therefore, the supported mechanisms for error detection and error correction are further attribute values indicating the transmission quality.

- Security

  Mechanisms for security and authentication play an important role in connection with communication protocols. In particular when a user is to be provided with access to confidential data, communication protocols that ensure a secure communication have to be employed. Aspects of security include, for example, authentication of users, data integrity, or confidentiality of a communication. In this group of characteristics the supported security mechanisms of a communication protocol are therefore stored as attribute values.

- Connection establishment

  In conjunction with communication protocols it is furthermore relevant if a device is permanently available for communication or if a connection has to be established first. This distinction is represented by an attribute value whose type is a boolean value. If a connection

establishment has to take place, a communication protocol can furthermore be characterized by the time it takes to set up a communication using this protocol.

- Communication path

  As regards this group of characteristics we can distinguish between communication protocols that allow for bidirectional communication and those that support unidirectional communication only. The possible directions of communication are represented by an attribute value belonging to this group. In addition to this, it is important to know whether the user actively has to initiate the retrieval of information when using a specific communication protocol or whether the information supply is initiated by the sender, i.e. the information logistic application. The former type of information supply is referred to as information pull, whereas the latter is called information push. Another attribute value concerning the communication path therefore serves to represent whether a communication protocol employs push or pull mechanisms or both of them for information supply.

- Costs

  The costs of transmitting information not only depend on the communication protocol employed, but also on other factors such as the duration of a transmission. Yet, we represent the transmission costs for a fixed amount of time or data, respectively by an attribute value in this group of characteristics as they are an important quantity allowing to estimate the total costs of a communication. This allows for the selection of the least expensive communication protocol in given circumstances and for the determination if in case of a restricted budget an information supply is to be carried out at all.

Finally, we describe the representation of applications in the reachability model's instance. Since there is a variety of different applications, it is advisable to create subclasses of the `Application` class to be able to classify an application type's specific characteristics as belonging to this type of application. In the following we explain the characteristics of three subclasses of the `Application` class we have modelled, the `Wap` class for WAP browser applications, the `Browser` class for conventional HTML browsers, and a `Software` class representing all other types of applications. If necessary, this class can be replaced by more specific subclasses of the `Application` class or can again serve as a superclass for further subclassing.

General characteristics each application possesses are represented by groups of characteristics which are associated with the `Application` class itself. These groups of characteristics are:

- Basic application characteristics

  This group of characteristics consists of the attribute values application name, version number, producer, and operating system name and version number.

- Application type

  As already mentioned, there are numerous types of applications such as Content Management Systems, browsers, Enterprise Resource Planning systems, e-mail clients, and many more. The relevant types of applications that have to be taken into account in the reachability model's instance depend on the application area of each particular information logistic

application. The type of an application is represented by an attribute value whose type is an enumeration type consisting of the available types of applications.

- Supported content types

  Applications support a specific set of formats data have to comply with in order to be processable. These data formats which we refer to as content types are essential as information logistic applications have to ensure that the data a user is to be provided with are available in a processable content type. In this group of characteristics the content types an application supports and accepts are stored as attribute values. Each of the content types is expressed as a MIME type according to RFC 2045 [FrBo96].

The groups of characteristics relevant for WAP browsers are as follows:

- Supported WML versions

  In addition to the supported WAP version which is stored as a basic attribute value of applications and hence is inherited by the `Wap` subclass, the supported versions of the Wireless Markup Language (WML) are attribute values belonging to this group of characteristics.

- Supported WMLScript versions

  This group of characteristics contains the supported versions of WMLScript.

- Supported WTA version

  In this group the supported version number of the Wireless Telephony Application (WTA) user agent is contained.

- Push message and deck sizes

  When transforming information into a format processable by the user's WAP browser, the maximum size of push messages the WAP browser can handle as well as the maximum size of a WML deck that can be downloaded, both expressed as number of bytes, are important data information logistic applications have to take into account.

- Device class

  The attribute value stored in this group enables a distinction between classes of so-called device profiles as identified in the WAP 2.0 specifications. Although the term device class may suggest that this attribute value refers to devices rather than to applications, the classification of the device class also provides information about the features of applications as, for example, their support of telephony functionality. Various classes of client and server devices have been defined for the current conformance release of WAP [WAP02].

- Supported WMLScript libraries

  This group of characteristics provides a list of the supported WMLScript libraries.

- Supported WTAI libraries

  The Wireless Telephony Application Interface (WTAI) enables applications to execute functions of a mobile telephone using WML or WMLScript and aims at making the user inter-

face and the handling of the application customizable. The WTAI is accessible in various libraries, each supporting particular functions. Therefore, this group of characteristics contains a list of the supported WTAI library names as specified by WAP WTAI and its addendums [WAP01], [WAP01a], [WAP01b], [WAP01c], [WAP01d], [WAP01e] as attribute values.

The groups of characteristics relevant for HTML browsers are:

- Additional version numbers

  Apart from the version number of the browser itself, the versions of HTML, of the Extensible HyperText Markup Language (XHTML), and – if applicable – of the JavaScript language supported by the browser are represented by attribute values contained in this group.

- Frames and tables support

  There are two attribute values belonging to this group which indicate whether the browser is capable of displaying frames and tables, respectively. Both attribute values are expressed as boolean values.

- Java support

  With regard to the Java programming language the support of Java applets and the JavaScript language is relevant. Therefore, in this group again two boolean attribute values indicate whether the browser supports Java applets and JavaScript, respectively.

- Supported XHTML modules

  The World Wide Web Consortium (W3C) describes XHTML as a »*reformulation of HTML 4.0 as an application of XML*« [AlBo01]. In this group of characteristics the XHTML modules a browser supports as, for example, »xhtml-image-1.mod« are contained.

- Supported encodings

  The attribute values belonging to this group represent the transfer encodings a browser supports. Each transfer encoding is expressed as a name as specified in RFC 2045.

- Supported character sets

  As already explained in conjunction with device characteristics, this group contains attribute values representing the supported character sets of a browser.

Finally, the characteristics of the subclass for all other kinds of applications are:

- Supported encodings

  The attribute values belonging to this group represent the transfer encodings an application supports, as explained in connection with HTML browsers.

- Supported character sets

  The attribute values belonging to this group represent the character sets an application supports, as explained in connection with devices and HTML browsers.

- Supported audio input encoders

  For those applications that support audio input in general this group of characteristics contains the supported audio input encoding formats as, for example, the G.711 standard.

- Supported video input encoders

  Analogous to audio input encoders the `ReachabilityAttribute` object corresponding to this group contains, if present, the video input encoding formats supported by an application. Examples of video input encoders are MPEG-1 or MPEG-2.

4.3.3 Summary

In this section we conclude our elaborations on the object model for the context element of reachability by summing up the model's distinctive characteristics and providing an assessment of it. Figure 30 which shows the overall reachability model accompanies this summary.

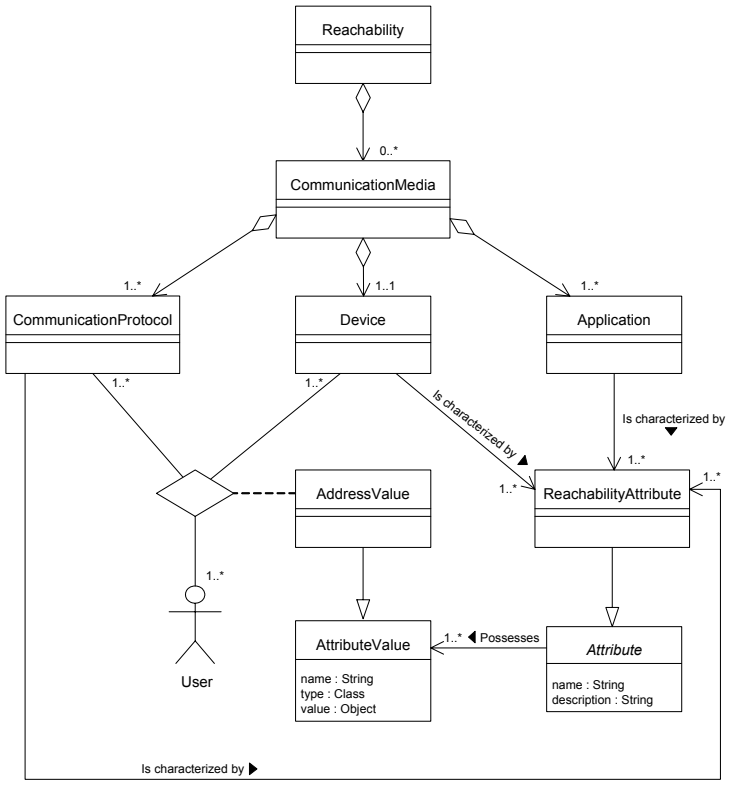

Figure 30: Overall reachability model

Based on reachability information information supply can be optimized with regard to the most suitable communication medium employed for the delivery of information. In the previous sections we have first of all identified the reachability elements available devices, applications, and communication protocols. Our object model for reachability represents the

characteristics each of these three elements possesses. We have pointed out that the rapid changes in the technological development of communication media require the reachability model to be exceptionally extensible. Therefore, we have presented both a generic model for reachability that allows for a flexible definition of the relevant characteristics of communication media for any particular state of technological development as well as an instance of the reachability model which describes the current state of technology.

Since devices, communication protocols, and applications are considered to remain indispensable elements of reachability, it is always a combination of these three elements that constitutes a communication medium in our model. We have explained that in the model a user's reachability is represented by an object of the `Reachability` class which consists of zero to many `CommunicationMedia` objects. A `CommunicationMedia` object aggregates exactly one `Device` object and at least one instance of each the `CommunicationProtocol` and the `Application` classes. The characteristics devices, communication protocols, and applications possess can be grouped into categories such as storage capacities, costs, etc. These groups of characteristics are represented by objects of the `ReachabilityAttribute` class which is a subclass of the abstract `Attribute` class we have already explained in the previous sections. Since the `ReachabilityAttribute` class inherits from its superclass an association to the `AttributeValue` class, the objects of this class serve to represent the names, types, and values of the specific individual characteristics that belong to a group.

Apart from the characteristics that describe communication media the addresses required to set up a communication between an information logistic application and a user are further relevant attributes represented in the reachability model. Since these addresses are different for each user of an application, they are modelled in relation to both communication media and users. Addresses are represented by objects of the `AddressValue` subclass of the `AttributeValue` class. The `AddressValue` class is an association class of the ternary relationship between users, communication protocols, and devices as an address to reach a user at may refer to both one or more devices and one or more communication protocols.

In connection with the modelling of reachability mechanisms for the management of the available attributes and their values' names, types, and contents as well as for the correct assignment of them to devices, communication protocols, or applications have been explained. We have also substantiated why a calculation of indicator values which provide an assessment of a communication medium's suitability for information supply cannot be carried out within the scope of the reachability model. Therefore, the `Reachability` class does not provide any operations for the rating of a communication medium's capabilities except those to access the attributes of the reachability model's classes.

After our description of the generic reachability model we have illustrated the usage of this model by presenting an instance of it which reflects the current state of technological development. In doing so, we have identified and described the currently relevant characteristics of devices, communication protocols, and applications. These characteristics have been grouped, and their representation according to the reachability model has been shown.

The most distinctive feature of our object model for reachability is its ability to represent the characteristics that constitute a person's reachability in a flexible and extensible manner. Not only the current state of technology is reflected, but rather the model provides for applicability regardless of technological evolution. In addition to this, the modelling of reachability we have presented is not restricted to particular application domains or types of communication media, but ensures an unlimited range of use.

Below we list the advantages of our reachability model at a glance:

- The model is generally applicable to any kinds of communication media.
- The applicability of the model remains unaffected by future developments in communication media technology.
- Instances of the reachability model representing exactly those characteristics of communication media relevant to a specific application can easily be created.
- The determinants of reachability can be described as detailed as required.
- Ease of use and user acceptance are achieved by the ability to define communication media's characteristics as needed and to refer to them by their names.

The design of the reachability model entails the following disadvantages:

- Instances of the reachability model may contain a large number of communication media's characteristics that have to be managed, maintained, and possibly created manually.
- The reachability model's high degree of flexibility and generality can only be achieved at the cost of an increased model complexity which results in a risen amount of computations required to access the individual values of a communication medium's characteristics.

4.4 An Object Model for Surroundings

According to the definition given in Section 2.2.2.2 the surroundings of an entity are one or more pieces of information about the entity's environment. The environmental conditions the context element of surroundings represents may, for example, be the degree of humidity, the intensity of light, the temperature, or the content of particular substances in the air, water, or soil. Although several existing projects in research and industry sense and process this kind of environmental data as discussed in Section 3.1, a comprehensive and general model for the representation of an entity's surroundings has not been developed so far. This section therefore fills this gap by presenting a universal model for the context element of surroundings.

Above we have already mentioned some pieces of information that characterize the surroundings of an entity. As with the other elements of context, however, a design of the object model for surroundings that is based upon a complete list of the individual environmental data it covers is neither feasible for all kinds of applications nor does it comply with our pursuit of generality and flexibility. The pieces of environmental data specific application domains require as well as the technological development regarding sensors for the gathering of these data can hardly be overlooked in their entirety. A model for surroundings that relies upon a limited,

pre-determined set of environmental data would become unusable as soon as a previously not considered datum needs to be processed. Since this kind of modelling does not allow for a smooth and easy extensibility, we consider it crucial to pursue a more generic approach.

The surroundings of an entity are described in terms of singular pieces of information about its environment. Thus, the mechanisms employed for the modelling of state and reachability attributes are a suitable means in connection with the modelling of surroundings as well. Each environmental datum can be regarded as an attribute of an entity's surroundings, and the sum of these data describing the entity's surroundings completely consists of a set of such attributes. Therefore, we make use of the concept of attributes in our object model for surroundings which also leads to a continuous utilization of concepts in the modelling of context.

Due to the fact that we have already explained the modelling of attributes in connection with other context elements we consider it sufficient in this section to describe our object model for surroundings less extensively than the models for the other context elements before. The model for surroundings is depicted in Figure 31. The class `Surroundings` which represents the surroundings of an entity is associated with an `AttributeSet` object. This class serves as a collection of all individual pieces of information about the entity's environment which are represented by `SurroundingsAttribute` objects. The class `SurroundingsAttribute` is a subclass of the abstract `Attribute` class and inherits both its attributes `name` and `description` as well as its association to the `AttributeValue` class which serves to store the names, data types, and values of a particular environmental condition. The light condition an entity faces can, for example, be represented by an object of the `SurroundingsAttribute` class whose `name` attribute contains the character string »light condition«. This object can be associated with two `AttributeValue` objects one of which quotes the light condition in terms of discrete character strings such as »daylight«, »dusk«, and so on, whereas the other describes the intensity of light more accurately as numeric values expressed in lux.

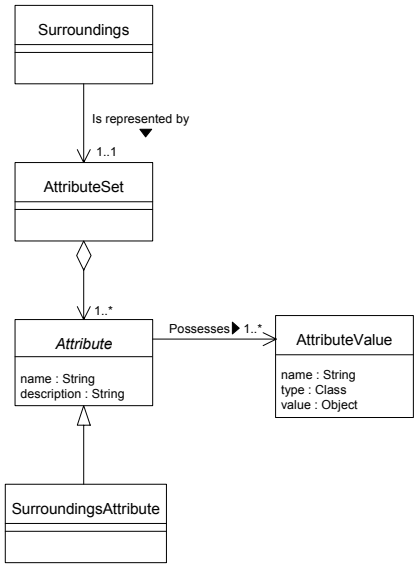

Figure 31: Object model for surroundings

Apart from the common methods to access and modify the attributes of the classes the model for surroundings consists of the `Surroundings` class does not possess any further operations. A comparison of a sensed datum about the environment with one that has been specified in a demand profile can easily be carried out using standard operators which compare the names of the respective `SurroundingsAttribute` objects and check the `value` attributes of the associated `AttributeValue` objects for equality or for being within a specified range (see also Section 6.2.1).

Since the aspects of an entity's surroundings can be regarded as independent singular data, their modelling is less complex than the modelling of the other context elements. Furthermore, the central components of the object model for surroundings have been described in detail in the previous sections. For these reasons the explanations concerning the model for surroundings given in this section have been kept brief. A separate summary of this section's contents is therefore not considered necessary. The advantages and disadvantages of our model for the context element of surroundings correspond to those pointed out in previous sections where we have introduced our concepts for modelling attributes. Again, our model is a generic and flexible one which provides for sensor independence, extensibility, and usability and facilitates access control to the individual data. This is, on the other hand, achieved at the expense of an increased amount of computations that may occur.

4.5 Summary and Assessment

This section concludes our elaboration on the object model for context by a brief summary followed by a comparison of our model with other existing models for context described in Chapter 3. In doing so, special emphasis is put on the way the context model proposed in this thesis overcomes the restrictions other results of research in the area of context modelling are still subject to.

Our object model for context consists of separate models for the context elements location, state, reachability, and surroundings. These elements have been identified by us as those aspects of an entity's context that have so far not been considered and are not dealt with by other components of information logistic applications. The selection of the relevant context elements has been based on scenarios for information logistic information supply [Lien01] and various discussions with interested parties from both academia and industry. Although the context elements dealt with in this thesis have up to now covered all the considered application areas of information logistics, an expansion of the object model for context may become necessary in case currently unforeseen aspects of an entity's context have to be taken into account as well. For this reason an entity's context is modelled as an aggregation of context elements as shown in Figure 32 on page 102.

All context elements implement a generic interface called `ContextElement`. This allows for a simple extension of the context model which can be accomplished by providing models for new context elements and making these new elements implement the `ContextElement` interface. As already mentioned, a `Context` object itself merely aggregates at least one `ContextElement` instance. The aggregation of context elements in a `Context` object means

that the aggregated context elements are equally valid, i.e. that all of them have been detected at or specified for a particular point in time. Correspondingly, the context elements that constitute an entity's context are connected with each other by an implicit conjunction.

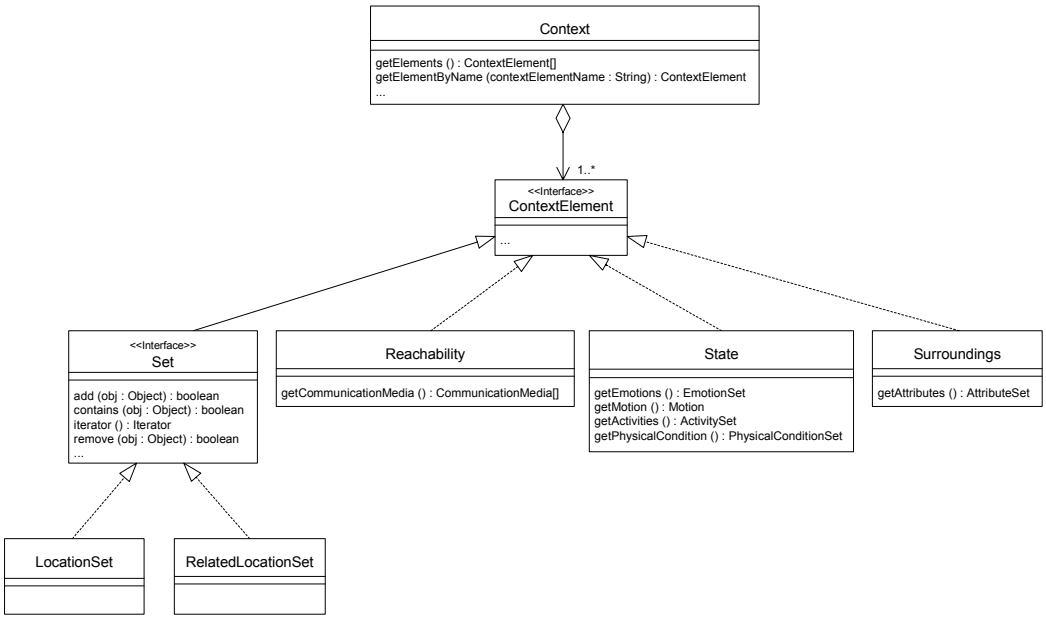

Figure 32: Top-level structure of the context model

As we have already mentioned, there is an interdependence among context elements' attribute values. The value of one context element's attribute may determine another element's attribute value or may confine its permissible range. In the following chapter we present mechanisms which allow to infer one context element's attribute value from another's and which ensure that the context gathering process results in only valid combinations of attribute values. However, context instances are not only generated during the context gathering process. They are also defined by users or other parts of information logistic applications as parameters in information demand profiles. Thus, the context model has to capture this interdependence among context elements' attribute values to prevent invalid combinations of values from being defined. For this purpose we introduce the concept of context constraints. Context constraints represent the interdependence among attribute values of context elements on the basis of logical implications. The fact that when a person is in a shaft below the ground, she cannot use her cellular phone, for example, corresponds to the logical term `(location=shaft)` $\rightarrow$ `¬(reachability=cellular phone)`. Context constraints serve to represent and evaluate such terms. By this means our model is able to capture the interdependence among context elements' attribute values and to ensure that only valid contexts can be defined.

The modelling of context constraints is shown in Figure 33. A `ContextSpecification` interface – which is described in detail in Chapter 5 – serves to represent conditions a context has to fulfill, similar to the `ContextElementSpecification` mentioned in Section 4.1.6. It

is made use of as a parameter in information demand profiles. Each `ContextSpecification` object refers to an entity. As mentioned above, context constraints capture the permissible or compulsory values of context elements' attributes depending on determinative data. They are represented by instances of the class `ContextConstraint`. This class is an association class attached to the association path between the `ContextSpecification` interface and the `Entity` actor. `ContextConstraint` objects aggregate at least one `ConstraintEntry` instance. The class `ConstraintEntry` serves to represent individual logical terms like the above exemplary one. It consists of determinants and implications. A `Determinant` instance represents the data that determine a context element's attribute value, while an `Implication` object corresponds to the resulting possible range of this value. Determinants may consist of an arbitrary number of context elements' attribute values each of which is attached an operation that is to be applied to the respective value (such as *equals* or *is greater than*, for example) as well as of entities. These constituents are interconnected by the sentential connectives *and*, *or*, or *not*. Entities can be part of determinants, because some value combinations may be possible or inadmissible for particular persons, groups, or roles, and the like only. Implications are composed of attribute values with operators attached to them that are interconnected by sentential connectives in the same way as determinants.

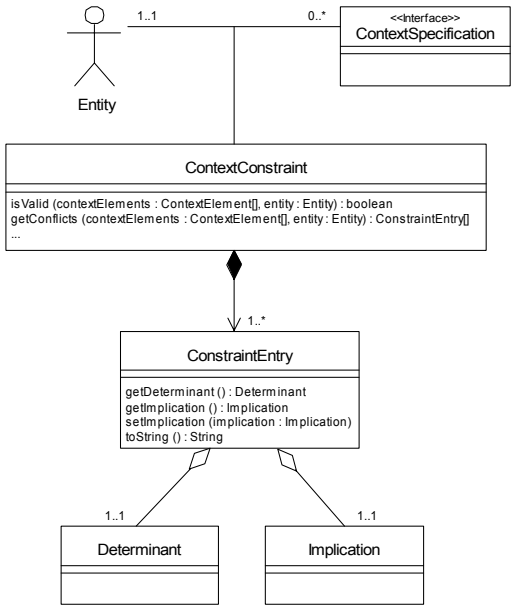

Figure 33: Context constraints

To facilitate the processing of context constraints a conversion of the logical terms into an equivalent unified form such as Horn clauses [Horn51] may be carried out. The constituent parts of determinants and implications are not shown in Figure 33 for reasons of clarity. A `ConstraintEntry`'s determinant and implication are implicitly connected by a logical implication. The logical terms represented by `ConstraintEntry` objects can be evaluated by instances of the class `ContextConstraint`. For this purpose this class provides methods that indicate whether the context elements defined in a `ContextSpecification` object

with reference to a particular entity are valid and which, if any, conflicts with the logical terms exist. The interdependence among context elements' attribute values is often highly application-specific. Since our context model, however, aims at universal applicability, we do not explicitly pre-define a set of context constraints. Instead, we prefer to give developers the opportunity to provide application-specific constraints, as a result of which the flexibility and applicability of information logistic applications is increased. In Chapter 5 a general rule language tailor-made for the specific needs of context-aware applications is presented the usage of which also suggests itself in conjunction with context constraints.

All models for the individual context elements have in common that they possess a high degree of generality and flexibility providing for sensor independence, extensibility, usability, and facilitated access control. These merits may, on the other hand, lead to a raised amount of computations and an increased model complexity. Since a detailed assessment of the models for the individual context elements can be found in the previous sections, the following elucidations focus on correlating the context model introduced in this thesis with proposed models for context that have been developed within the scope of other research activities.

An outstanding feature of our context model is the fact that it deals with the relevant context elements of location, state, reachability, and surroundings in a comprehensive manner. Unlike many other approaches including the TEA project, the Context Toolkit, or the SOLAR system our model provides a formal and structured representation of each of these context elements. It takes potentially complex contextual information into account and allows to describe context with different levels of precision and granularity. Features such as the support of different coordinate systems for locations including symbolic ones, the explicit modelling of prepositions, or the provision of operations to transform locations and to determine spatial relationships between them are to our knowledge unique to our model and are not provided by any other existing approach. Similarly, our context model so far is the only approach that allows to represent entities' state, reachability, and surroundings by means of an extensible set of context attributes that can be defined according to the specific needs of applications. Thus, in contrast to Affective Computing or Sentient Computing, for example, the variety of contextual information is explicitly taken into account. Furthermore, our context model is independent of any particular technology employed to determine context and as a result overcomes the restrictions other approaches, for example those employed in the TEA project or the Coordinated Adaptation Platform, suffer from. While Henricksen et al.'s or other models make simplifying assumptions concerning context attributes that restrict their generality, our model for context can be applied to any application domain and enables an easy adaptation to the demands of specific environments. Context elements and their attributes are referred to by their names which allows to convert contextual information into a human-readable representation and furthermore facilitates access control. The formal and structured character of the model as well as the operations that can be called on locations – and in addition on contexts and context elements as we explain in Section 6.2.1 – enable information logistic applications to easily compare contexts or context elements with each other. Consequently, the model for context we have presented in this chapter both fulfills the requirements identified in Section 2.4.1 and, due to its distinguishing features, overcomes the shortcomings of existing approaches.

5 Context Gathering Techniques

In order to adapt information and service supply to entities' contexts it is obvious that these contexts have to be known. Information logistic applications therefore need means of obtaining context data from various sensors as pointed out in Section 2.4. This task is within the responsibility of the Context Component. The Context Component obtains context data from external context sensors and, if need be, transforms, augments, and filters these data to provide meaningful contextual information to other components of information logistic applications. This chapter provides the required techniques for an efficient context gathering which we have also outlined in [Hase04]. These techniques are based on a three-tiered hierarchy of components involved in the context gathering process. The layers of this hierarchy – sensor adaptors at the bottom, above them virtual sensors, and context builders at the top – are explained in the following from the bottom up.

5.1 Sensor Adaptors

In this section first of all potential context sensors and their relevant capabilities are identified and examined. Due to the large number and great diversity of context sensors no complete examination of all these systems can be given. This section therefore illustrates some representative examples of sensor systems that can be employed to detect the context elements dealt with by the Context Component. It furthermore provides criteria which serve to assess the sensors' suitability for use in a particular information logistic application independent of the respective state of technology. In addition to this, the process of transforming the raw data obtained from context sensors into context objects is illustrated. Subsequently, this section focuses on the integration of context sensors into the Context Component. For this purpose the concept of sensor adaptors is introduced, and an object model for the necessary data structures and the relationships between them is presented. The other tasks that have to be carried out in conjunction with context gathering such as the aggregation of context data and derivation of contexts etc. are dealt with in Section 5.2 and Section 5.3, respectively.

Definition 15: Context sensor

A context sensor is any hardware and/or software system that provides data about the entire or a part of the context of one or more entities.

Given this definition, the aforementioned heterogeneity of context sensors becomes obvious. The definition implies that context sensors are not limited to systems that are commonly associated with the task of determining context and that have reached a certain degree of spreading in various applications such as the Global Positioning System (GPS) or the IEEE 802.11 Wireless LAN standard. They may rather include other electronic facilities the usage of which as context sensors does not immediately suggest itself such as electronic travel plans, room or car reservations, surveillance equipment, and many others.

5.1.1 Overview of possible context sensors

This section provides a brief overview of some exemplary types of context sensors that can be employed to detect the context elements handled by the Context Component. Apart from the examples given below it is worth mentioning that any aspect of the context of persons can also be gathered by explicit user input. The context sensors made use of in this case are software applications that offer an interface to people by means of which they manually report their entire context or some relevant context attributes.

5.1.1.1 Sensors for the context element location

Various systems are available for the detection of an entity's whereabouts. We can divide these location sensors into two groups. The first group contains genuine positioning systems, i.e. systems which have been developed specifically for the task of determining entities' locations. In contrast to this we classify systems that essentially serve a purpose other than position detection, but that can be employed to gain this type of information as well into the second group of systems which we call derived positioning systems.

The following types of positioning systems can be identified (see also [Ward98] and [Leon98] for further details):

- Infra-red-based systems

  These systems employ sender/receiver stations and mobile tags for position detection. Since infra-red communication requires a line of sight between sender and receiver, it is mostly used indoors. One of the most prominent infra-red-based location systems is the Active Badge system developed at Olivetti Research Laboratory [WaHo92b].

- Cellular radio systems

  This type of systems operates by cell base stations regularly sending radio signals which are received by mobile devices. Depending on the size of the cells these systems often provide data with a limited position accuracy. Representative systems of this category are cellular phone networks such as GSM and Wireless LAN networks.

- Satellite-based radio systems

  This position detection technique is based on satellites which constantly broadcast time and position signals that can be picked up by receivers on or near the earth's surface. Due to signal reflection these systems can only be used outdoors. Satellite-based position detection is made use of in the Global Positioning System (GPS) [Well86].

- Terrestrian radio systems

  These systems are based on techniques similar to those of satellite-based radio systems. The different methods used in terrestrian radio systems to calculate an entity's position are described in [Zhao97]. Examples of such systems are the wireless E911 emergency call service in the United States and, although targeted on communication over short distances, RFID (Radio Frequency Identification) systems.

- Ultrasonic-based systems

  Position detection by means of ultrasound calculates an entity's three-dimensional position based on the distance between ultrasonic senders and receivers. Either the sender or the receiver can take on the fixed part of the system; it can, for example, be installed in the ceiling of a building like in the Bat system developed at AT&T Laboratories [WaJo97].

- Optical systems

  In this kind of systems detectors such as cameras or photodiodes observing the entities that are to be located are employed. Like infra-red-based systems optical positioning requires a line of sight between the detector and the entity. Usually the reliability of currently available optical systems suffers from an increase in the number of entities within the observed area. Optical location detection is, for example, made use of in Microsoft's EasyLiving project [KrHa00].

- Electromagnetic systems

  These systems consist of both a transmitting and a receiving antenna. The transmitting antenna has a fixed position and possesses field coils which induce currents in the coils of the receiver [RaBl79]. Since these currents vary with the relative position and orientation of the receiving antenna, they are used to calculate position and orientation information.

Apart from these types of position trackers several other systems originally implemented for a different purpose can serve as sensors for the context element of location as well. Some examples of such derived positioning systems are:

- Chip card readers or other systems for admission control or registration of working time which provide information about the stay of people or objects in buildings
- Electronically administered reservations of rooms, official or rental cars or other means of transport, and other shared equipment
- Electronically administered (business) trip application forms and plans
- Electronic agendas and schedules containing appointments along with the locations they take place at
- Indices providing the positions of fixed entities such as machinery or other equipment. This includes plans of buildings or factories, maps, directory services, and so on.

Location data can in addition be gathered on the basis of data concerning other context elements as well. The login status and terminal of a user in a network or the busy state of people's stationary telephones can, for example, be utilized to infer users' locations. The context sensors employed in these cases are, however, reachability sensors as they only provide data about the communication media users have at their disposal. The generation of location data on the basis of data provided by these sensors is carried out by means of augmentation processes which are not considered context sensors for the context element of location. Such augmentation mechanisms can be made use of for all context elements. Since they are not based on the data of a context sensor for the respective context element, we do not consider them in the scope of this section, but explain these mechanisms in Section 5.2 instead.

5.1.1.2 Sensors for the context element state/motion

To detect the motion of an entity some of the positioning systems introduced above can be made use of. Optical systems are particularly suitable for this purpose as, for example, the use of the Medusa architecture [WrGl94] described by Nelson [Nels98]. The primary motion attributes that can be gathered by these sensors are the direction of an entity and the entity's motion speed in case it is mobile. In addition, a few sensors have been developed that inherently provide data about the movement of entities. In the Mediacup project [BeGe01], for example, metal ball-switches integrated into coffee cups can detect the cup's orientation as well as the fact whether the cup is placed on a surface or whether it is carried or held.

Because of the significantly smaller number of potential sensors for entities' motions compared to the available sensors for location, the gathering of this context element is to a large extent dependent on the derivation of motion patterns from locations, activities, or other context elements which can also provide more sophisticated motion attributes.

5.1.1.3 Sensors for the context element state/activity

For the activity of an entity there are likewise relatively few sensors available that are capable of directly determining this context element. The activity of devices such as computers or cars may be ascertained by querying their internal list of active processes or by querying monitoring devices or applications. These types of sensors can also provide activity attributes such as the performed task's progress, the point in time it has begun at, the resources consumed, reject rates, etc. When aggregated with information about the person using or operating the object at a given time, human activities can be gathered. In addition, software applications supporting business processes such as workflow systems can also supply information about users' activities based on their underlying process model.

Another option for activity detection is again the usage of electronic agendas and schedules. Since electronic schedules often allow for arbitrary entries, this type of context sensor may require additional restrictions to be made. A fixed syntax and terminology for schedule entries or interpretations have to ensure the comparability of a sensed activity with an activity defined in an information demand profile.

5.1.1.4 Sensors for the context element state/physical condition

The detection of entities' physical condition relies to a great extent upon hardware devices. Specific bodily functions of humans can be gathered by medical instruments such as blood-pressure and heart rate testers, measuring devices for respiration frequency and skin conductivity, to name but a few [HePi98]. A prerequisite for these instruments to be utilized as context sensors is that they possess an interface for electronically passing on the collected data. Due to the great diversity of sensors for physical condition and of the measuring methods they use a classification of these systems seems futile and is therefore not given in this thesis.

For the detection of objects' physical condition similar sensors exist. The internal state of the objects can be obtained from themselves or from administrative units providing information about the objects' capacity and load, the status of their functions, possible dysfunctions, etc.

5.1.1.5 Sensors for the context element state/emotional condition

The emotional condition of people is a context element that eludes any automatic detection by sensors. Unless explicitly reported by persons the gathering of this context element is fully dependent on derivation mechanisms based on the physical condition of people or other factors as we explain later on.

5.1.1.6 Sensors for the context element reachability

The reachability of a person is composed of both the communication media being in her possession – such as a cellular phone or a notebook – as well as those that are available to her at her current location. Providing complete information about a person's reachability therefore requires the aggregation of data from reachability sensors with data from location sensors as a person's location has to be known in order to find out which communication media are available to her at this place. To determine the communication media belonging to a person or to a location, respectively, an electronic list of the relevant persons' and locations' devices, the applications installed on the devices, and the supported communication protocols along with their relevant attributes is required. This information does not necessarily have to be obtained from one single sensor. Indeed, it is more likely to have a separate data source for the basic data about people's and locations' communication media and one or more other, probably different, data sources containing greater details concerning the media's attributes.

In addition, the reachability of persons may change rapidly, especially in mobile environments where technologies such as mobile agents or ad-hoc networking [RoPi00] require quick adaptation to changes in available applications, bandwidth, and so on. Since the mentioned lists of communication media and their capabilities cannot reflect dynamically varying reachability characteristics, in particular changes in network environments, device, application, and network monitors are further relevant sensors for this context element. Several commercial and research tools for network monitoring exist which can be employed for this purpose.

5.1.1.7 Sensors for the context element surroundings

The detection of the context element of surroundings is similar to that of physical condition. An entity's surroundings can also be ascertained by various hardware systems such as photodiodes, temperature sensors, microphones, air pressure sensors, sensors for the concentration of particular substances such as carbon monoxide or ammonia in the air, soil, or water, etc. Many simple sensors like these have, for example, been made use of in the TEA project [ScAi99]. The variety of environmental data that can be gathered by these sensors and the heterogeneity in detection techniques employed prevent an exhaustive classification of these systems from being feasible within the scope of this thesis.

5.1.2 Criteria for context sensor assessment

From the point of view of the Context Component context sensors are external data sources. The implementation details of sensors are therefore beyond the scope of information logistic applications and are not examined in this thesis. Yet, in order to select suitable context sen-

sors and to efficiently obtain and process their data the examination of context sensors with regard to certain criteria provides valuable information. It is a sign of the context sensors' aptitude for the application's purpose and indicates how the data gathered from them have to be dealt with by the Context Component. Hence, this section introduces criteria that serve to assess context sensors. The main stress of the explanations is put on those aspects which are particularly relevant for the processing of sensor data within the Context Component.

Hightower and Borriello [HiBo01] have developed the following classes of properties for the assessment of sensors for the context element of location:

- Physical vs. symbolic location data

  This criterion distinguishes between sensors that provide location data in the form of geographic coordinates and sensors providing symbolic location data.

- Absolute vs. relative location data

  Absolute location data are based on a uniform representation system for all determined locations. In contrast to this there can be a separate reference system for every sensed location if the location data provided by a sensor are relative.

- Position-finding system part (infrastructure vs. tracked object)

  The process of determining an entity's location can be carried out by either the location detection infrastructure (e.g. a beacon) or by the entity that is to be located itself (e.g. a GPS receiver).

- Accuracy and precision

  A sensor's accuracy indicates the smallest area it can determine, while its precision gives information about the percentage of location detection processes achieving this accuracy.

- Scale

  To assess the scale of a sensor its covered area per unit of infrastructure as well as the number of locatable entities per unit of time are considered.

- Recognition

  This property refers to a sensor's ability to recognize certain features of the tracked entity such as its identity, colour, shape, etc.

- Costs

  Hightower and Borriello distinguish between time costs, space costs, and capital costs a sensor causes. Time costs take a sensor's installation and administration into consideration, while space costs cover the infrastructure and hardware needed. The sensor's price and the incurred staff costs belong to capital costs.

- Limitations

  Under this property restrictions concerning a sensor's usability are subsumed as, for example, GPS sensors' limitation of being usable only outdoors.

These properties of location sensors can to a large extent be applied to sensors for the other context elements as well to achieve a classification of context sensors as a result. Merely the distinction between physical and symbolic data proves to be of little use to other context sensors than those for location. Moreover, this distinction is not sufficient in order to be applied to the Context Component. With regard to our object model for location the property should additionally examine the coordinate system a location sensor's data refer to.

The aim of Hightower's and Borriello's identification of these properties is to put criteria for the selection of suitable sensors at the disposal of location-aware applications' developers. Accordingly, these properties reflect a view of the potential context sensors which corresponds to the view of the operators of a context-sensitive application. They primarily assist in answering the question of which sensors to make use of in a particular application. In addition to this way of inspecting context sensors, we recognize the need to further examine them from the Context Component's point of view. Here the question of how to integrate context sensors and how to process the data supplied by them comes to the fore. With regard to this question some of the properties mentioned above play a minor role as, for example, costs and the part of the system carrying out context detection, while additional sensor characteristics can be identified which have not been considered by Hightower and Borriello. Therefore, we add the following criteria for context sensor assessment to the property list given above:

- Inherently supplied context data

  Since the Context Component processes data referring to different context elements instead of being tied to one particular context element as in Hightower's and Borriello's approach, it is essential to examine which context element(s) a sensor is able to inherently provide data for. The term inherent means that the data can be obtained directly from a sensor without making use of additional derivation mechanisms. A measuring instrument for persons' pulse rate, for instance, inherently supplies data about the context element of state/physical condition; an electronic schedule can inherently be obtained data about the context elements of state/activity and possibly location from. The `ruser` service on the UNIX operating system provides data about persons' reachability by determining the login status and terminals of users, and so on. In contrast to other approaches which regard this service as a location sensor [Leon98] we believe that location information can only be obtained from this service by combining its information with a directory relating terminal names to locations. Therefore, we consider this sensor a sensor which inherently provides data concerning the context element of reachability.

- Finiteness and value range of the supplied context data

  Whenever the Context Component is queried for contextual information, it has as a rule several sensors at its disposal that it may utilize for context detection. In order to assess whether a sensor is suited to be employed for the execution of a particular request it is important to know whether the value range of data supplied by the sensor is finite or not. If the value range is infinite – e.g. in the case of an electronic schedule allowing for arbitrary entries –, the sensor can be made use of for all queries concerning the context element(s) it supplies. A finite value range, however, requires the Context Component to compare it to the parameters of the request that has to be fulfilled. This is particularly necessary if the

Context Component is queried asynchronously and is requested to fire an event if a context or a context element satisfies certain conditions. An example of a sensor with a finite value range is an infra-red-based location sensor installed in a building's rooms. This sensor can only supply data concerning the rooms its components are installed in.

- Entity types and instances

  The data a context sensor supplies can refer to different entities and types of entities, respectively. A sensor for the concentration of ammonia in the atmosphere installed at a particular location, for example, provides data about the surroundings of a location. In the same way, a car's GPS sensor supplies location data with regard to an object or – if a corresponding coordinate system exists in the application – another location. Administrative equipment observing the status of telephones is an activity (and possibly reachability) sensor for devices, and so on. By means of combining different sensors' data with each other the data supplied by a sensor can be related to other entities. The mentioned GPS sensors installed in company cars combined with a sensor managing the employees' reservations of the cars can, for instance, provide data about the physical location of the employees, i.e. persons, as we explain in greater detail in Section 5.2.1.

  For this reason every sensor has to be examined with respect to the entity types such as persons, locations, devices, objects, etc. it can provide context data for. In addition, it has to be considered whether the value range of the individual instances of these entity types is finite or infinite. Since most sensors' coverage is limited, usually the former is the case. Consequently, information concerning the entity instances is also relevant to the Context Component in order to assess whether a sensor can be employed to fulfill a request for the detection of a particular entity's context or context element.

- Reliability

  The abovementioned information concerning a sensor's accuracy and precision can be complemented by information about its reliability. The reliability of a context sensor denotes the probability with which the determined context data are actually correct. Usually infra-red-based location systems can, for example, only detect the presence of tracked objects, but cannot sense that these objects leave the area covered by them, leading to a decrease in the sensor's reliability. Sensors for other context elements often possess a limited reliability, too. Due to events at short notice, for instance, a person may be carrying out a different activity than the one entered in her calendar. Therefore, information about a sensor's reliability also affects the selection of sensors used to fulfill a request made to the Context Component and furthermore allows to optimize the processing of context data both inside the Context Component itself as well as in other components of information logistic applications.

- Synchronous vs. asynchronous

  Another important aspect for the integration of context sensors into the Context Component is the fact whether a sensor supports synchronous requests only, asynchronous requests only, or both types of requests. If only synchronous queries are supported, the sensor has to be polled cyclically in case the Context Component has to satisfy an asynchronous request made by a client. The other way round, events received from sensors which

only support asynchronous requests may have to be buffered in order to satisfy synchronous requests made to the Context Component with the aid of these sensors.

- Availability

  Some sensors may be available at certain times only. Due to service maintenance or a temporarily limited operation down times may occur during which a sensor cannot be used. Recognizing these periods in advance, provided that they are foreseeable, can contribute to optimizing processes within the Context Component. Sensors which are unavailable at the moment a context detection has to be carried out are not considered as a result of which unnecessary processing is avoided.

- Additional data

  Apart from data concerning context elements some sensors additionally provide further information which may be relevant to the Context Component. An electronic schedule, for instance, is able to supply the expected duration of an activity, i.e. its period of validity. We also subsume metadata about the sensor's ability to provide data referring to contexts in the future, i.e. whether it supports context prediction, and about the sensor's (average, maximum, and minimal) response time under this criterion.

These additional criteria serve the Context Component as a basis to determine if a sensor is suitable for context detection whenever a request is made to it. The consideration of these criteria enables an efficient dynamic acquisition of context data. We come back to them in Section 5.1.4.2 when explaining the relevant data structures used to select appropriate sensors for a request.

5.1.3 Transformation of sensor data into context data

Since context sensors provide data in diverse formats, transformation mechanisms have to be employed in order to create objects according to the context model on the basis of these data. In the same way data conversions have to be carried out whenever asynchronous requests are made to the Context Component requiring it to fire an event when an entity's context or context element meets certain criteria. In these cases the request parameters supplied to the Context Component specifying the criteria have to be mapped to sensor data.

Due to the considerable heterogeneity concerning context sensor data and interfaces the concrete actions that are necessary in a particular information logistic application to transform sensor data into context data and vice versa cannot be set in advance in a standardized way. There is rather a need for sensor-specific processes of data collection and transformation. However, some fundamental steps required in conjunction with every transformation along with aspects which generally have to be paid attention to can be identified. A description of these steps and considerations is given in this section, resulting in a guideline for the process of sensor integration and sensor data transformation.

In order to transform sensor data into context data the following measures have to be carried out in preparation before the actual transformation can take place:

1. The sensors that are to be employed for context detection have to be selected and installed on a suitable platform.
2. For each sensor the data supplied by it must be examined thoroughly, and a detailed understanding of their syntax and semantics has to be developed. It is advisable at this point to give initial thoughts on which objects and attributes of the context model a sensor's data correspond to. If the sensor employs different identifiers for the entities the context data refer to than the information logistic application, it must also become clear which identifiers used by the sensor represent which entity in the information logistic application. These relations should be recorded in what we call a mapping table which is described in greater detail below. It is possible, though, that at this stage no relations between sensor data and context element objects can be recognized. This is the case if the data supplied by the sensor cannot directly be mapped to the context model and to entity identifiers, respectively. An indoor location sensor installed in a building's rooms, for example, which provides data in the form of (Beacon-Id, Tag-Id) pairs does not supply any data which can directly be converted into context model objects or entity identifiers. Before this mapping can be carried out the identifiers for the beacons and tags have to be interpreted and enriched with further information to be able to extract entity identifiers known to the information logistic application and location coordinates referring to a suitable room coordinate system from the data. This procedure is explained in step 4.
3. Provided that a sensor possesses a programming interface (API) this interface also has to be examined and understood in order to be aware of how data can be obtained from the sensor. If there is no API available, programme code has to be implemented which speaks the sensor's protocol and enables the Context Component to communicate with the sensor.
4. Additional data and data sources required to interpret and enrich sensor data and entity identifiers have to be identified, examined, and, if need be, implemented. The implementation involves both a creation of the data source as well as a programming of an API if the data source itself does not possess one. Coming back to the example of the indoor location sensor mentioned above, a data source for the mapping of tag identifiers to person or object identifiers and a data source providing information about the rooms beacons are installed in are needed. The types of data sources employed can be manifold; the required data may, for instance, be obtained from directory services, databases, files, maps, and so on. During the execution of this measure a detailed understanding of the data sources made use of, their programming interfaces, and the data contained in them along with their storage and structure has to be gained. On the basis of the tasks carried out at this point the mapping table created during step 2 can be completed by relating sensor data to data stored in the additional data sources which are again related to context model objects and their attributes and entity identifiers.

Since the mapping table that is constructed during steps 2 and 4 plays a significant role for sensor data transformation, we illustrate this concept by exemplarily outlining a mapping table for the mentioned indoor location sensor. As we have already pointed out, no direct mapping of sensor data to context data is possible, but rather additional data sources have to be made use of. In our example we assume that the sensor's tags are only carried by users and that the tag identifiers supplied by the sensor are mapped to user identifiers known to the information logistic application by means of properties files. We further assume that to

resolve the beacon identifiers a relational database is employed which contains a table with columns for the beacon identifiers and the associated room coordinates. The exemplary mapping table is shown in Table 2.

Sensor datum	Data source type	Data source name	Data source path	Data source data	Context element	...
Beacon-Id	Database	SensorDB	jdbc:oracle: thin:@ahost: 4711	Room (BeaconId, Name, Number, Type)	Location. Room Coordinate-System (Name= Room Name, Number= Room Number, Type= Room Type)	
Tag-Id	Properties file	UserTag-Mapping. properties	c:\abc\def	TagId=UserId	-	

Table 2: Sketch of an exemplary mapping table for an indoor location sensor

The table contains a row for each datum the sensor supplies. Its columns denote the relevant information required to accomplish the mapping such as the names, types, and locations of the data sources employed, the way the data provided by a data source are structured and relate to the sensor data as well as the manner in which these data are mapped to context elements or entities. Further columns providing additional data may also be present and could be used to implement the data transformation in a dynamic way, but have been omitted in the example for reasons of clarity.

The measures described above have to be carried out once for each sensor as a prerequisite for sensor integration and data transformation. Therefore, they constitute a basic foundation which has to be built before an information logistic application is deployed. The results gained from these measures remain valid until the data, their format, or the interfaces of the sensors or data sources change. After these steps the Context Component is ready to carry out the actual transformations. In contrast to the previous measures this involves steps which repeatedly have to be performed by the Context Component at run-time. These further steps are:

5. The sensor data that are to be transformed have to be made available to the Context Component, i.e. a request to one or more context sensors has to be made. At this stage a transformation of entity identifiers may already have to be carried out, either before or after the request is made, depending on the sensor's interface.

6. If applicable, the obtained sensor data may have to be stored persistently. A storage of these data allows for a later analysis of entities' successive contexts. It may serve to predict future contexts and may contribute to an increased efficiency of context gathering processes. In addition, a storage is necessary for sensors that can only be queried asynchronously in order to process synchronous requests made to the Context Component with the aid of these sensors as well. There are two alternative ways of storing data about entities' contexts. Either sensor data, i.e. data which are not yet transformed, or context data can be stored. The former type is recommended in case the sensor data are made use of infrequently and therefore their transformation would cause an unusually high expenditure. If the storage of context data is opted for, however, this step does not have to be carried out, but the storage has to take place after step 11 instead.

After these steps the actual transformation of sensor data and – unless this has been done before – of entity identifiers can be carried out. In doing so, the mapping table constructed during the initial measures plays a significant role. For the transformation of sensor data into context data the following necessary steps can be identified:

7. In case not all data supplied by the sensor are relevant for the transformation that is to be carried out, i.e. refer to the request that is to be fulfilled, the relevant data have to be extracted. The mentioned indoor location sensor may, for example, send notifications about all movements of the entities it tracks. During this step those sensor data that refer to the entities and context elements currently under consideration have to be extracted.
8. On the basis of the relations between sensor data and context elements established in the mapping table and during the classification of sensors objects according to the context model have to be created for the context elements concerned. In addition, the required associations between these objects have to be established as well.
9. Those sensor data which can directly be mapped to context data need to be assigned to the attributes of the objects created in the previous step. Again the mapping table is referred to in order to determine how sensor data and context data relate to each other.
10. On the basis of the data provided by the data sources made available in step 4 an interpretation and enrichment of those sensor data which cannot directly be converted into context data has to take place. The data sources must be accessed and passed the data received from the sensor, and the additionally required data need to be obtained.
11. Analogous to step 9 the data obtained in the previous step have to be assigned to attributes of the context model objects created before. Once again this is done on the basis of the relations gathered from the mapping table.

After these steps have been finished, context data in the form of complete objects according to the context model are available. These objects may at this stage be stored if during step 6 the persistent storage of context data was opted for. They may also be returned as method results for further processing within the Context Component or become attributes of event objects which are sent to registered event listeners as explained in the following section.

As a parameter of asynchronous requests made to the Context Component conditions concerning the entire context or one or more context elements may be specified by clients. When these conditions become true, the Context Component fires an appropriate event. If in this

case an asynchronous sensor which can be passed these conditions is employed for the observation of the specified context or context elements, the abovementioned steps 7 to 11 have to be run through in opposite direction. However, the transformation steps can either take place in the direction from sensor data into context data or vice versa if the sensor employed can be queried synchronously or if it is an asynchronous sensor which supports the registration of listeners without any conditions as parameters only. Yet, in case of asynchronous requests there is always the need to make additional comparisons after the transformation steps have been carried out in order to check whether the conditions specified by the client match the data supplied by the sensor.

5.1.4 An object model for sensor adaptor integration

This section introduces sensor adaptors as an essential building block for context gathering. It describes how this layer of the context gathering process is structured to efficiently handle the tasks of integrating various heterogeneous context sensors and obtaining data from them and of sensor data transformation. For this purpose the necessary structures along with the relationships between them are presented in an object model.

5.1.4.1 Representation of sensor adaptors

A sensor adaptor is a piece of software which encapsulates the interface, implementation, and data formats of a specific context sensor and provides a uniform interface to clients. The adaptor forwards the requests it receives to the sensor and performs additional tasks such as data transformation [GaHe94].

For each context sensor that is to be integrated into an information logistic application and the data of which are to be used for context detection at least one adaptor must be created. The concept of sensor adaptors as a means to encapsulate context sensors has the advantage of decoupling context sensors' implementation details from other parts of the Context Component. Since these details only need to be known to sensor adaptors, the impacts of a modification of sensor interfaces or data are minimal and limited to the adaptors only. In addition, the abstraction of sensor data implemented by the sensor adaptors provides other parts of the Context Component with data in the format they expect and understand, i.e. with data according to the context model. The adaptors' interface which is described shortly allows other parts of the Context Component to easily access gathered context data. Furthermore, this uniform interface can be utilized by various parts of the Context Component in the same manner. For this reason sensor adaptors allow for a reuse, combination, and customization of sensors to support a variety of aggregation, derivation, or other processing mechanisms.

As we have already mentioned, the data supplied by a sensor adaptor may refer to any entity such as a car, device, location, and so on. The combination of several sensors' data in order to obtain context data related to a different entity does not belong to the tasks of sensor adaptors, but is rather explained in conjunction with virtual sensors in Section 5.2.1.

The representation of sensor adaptors in our object model is shown in Figure 34. All sensor adaptors implement the general interface `Sensor` which provides methods common to every sensor adaptor. Each `Sensor` object possesses an identifier and a name as well as cor-

responding methods to access these attributes. It furthermore can be initialized, started, and stopped. During the initialization process a `Sensor` object performs measures to become ready for operation such as registering itself with other objects as described below. Starting a `Sensor` object causes it to connect to the actual context sensor and to other required resources and to become capable of accepting and processing incoming context data and requests for context detection, whereas the `stop()` method of `Sensor` objects causes a sensor adaptor to release any connections and resources and to terminate its operation.

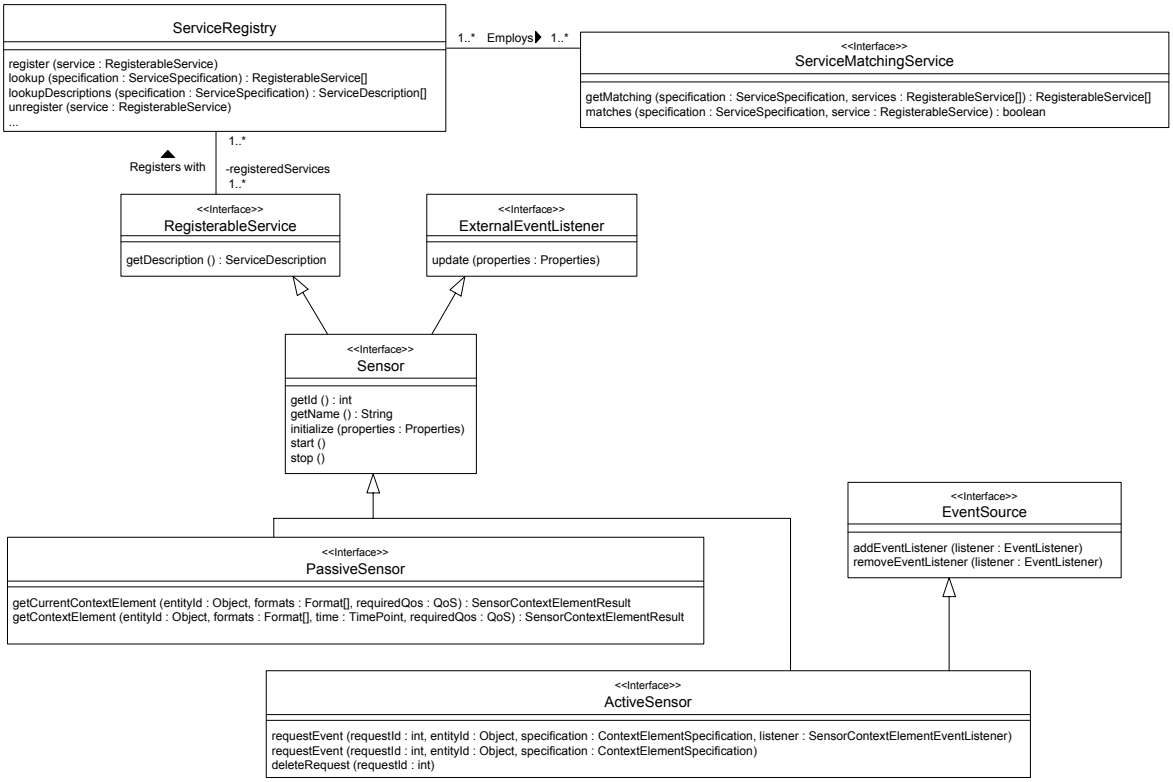

Figure 34: Sensor adaptors

Sensor adaptors are managed in a service registry and therefore register themselves with a `ServiceRegistry` object. The `Sensor` interface thus inherits from the `RegisterableService` interface. We have already described this registration mechanism in Section 4.1.5 and do not elaborate on it any further in this section. There may be changes in the data supplied by context sensors compelling sensor adaptors to adjust their associated `ServiceDescription` object. These changes occur when the entities and context data known to the sensor change. This may, for example, be the case when new users are added to a company's centralized electronic schedule, when a beacon of an indoor location sensor is reinstalled in a different room, when a company's car is driven by a different executive than before, and so on. In order to deal with these cases appropriately by changing their service description and reregistering with the service registry sensor adaptors must be notified of them. For this purpose the interface `Sensor` also inherits from the interface `ExternalEventListener`. The

`update()` method of this interface allows external administrative tools to send notifications about relevant context sensor changes to the respective sensor adaptors. This approach provides the opportunity to dynamically update context sensors' capabilities without the need to restart the Context Component.

The object model for context sensor integration confines classes implementing the `Sensor` interface to supplying data for only one context element each. This restriction is imposed in order to separate the process of detecting individual context elements from the aggregation and combination of context data which may refer to various context elements. It facilitates the discovery of suitable sensor adaptors and enables an unattached processing of transformations for each particular context element which results in an increased maintainability of sensor adaptors. A sensor supplying data for more than one context element such as an electronic schedule that provides data about entities' activities and locations is represented by one sensor adaptor per context element. In these cases it is up to the programmer of the adaptors to take care of a reasonable implementation of programme code required by more than one adaptor – e.g. code to establish a connection to a sensor – to avoid redundancies.

The interface `Sensor` has two direct subinterfaces, `ActiveSensor` and `PassiveSensor`. The `ActiveSensor` interface represents adaptors for sensors which support asynchronous queries, meaning that they are capable of firing events, whereas adaptors for sensors supporting synchronous requests are represented by `PassiveSensor` objects. For sensors which can be queried both synchronously and asynchronously adaptor classes implementing both of these interfaces are created.

The interface `PassiveSensor` defines two methods which allow the clients of a synchronous sensor adaptor to obtain information about the context element the adaptor supplies data for with reference to a particular entity and particular formats. The concept of formats is described below. While the method `getCurrentContextElement()` returns the context element for the present moment, the method `getContextElement()` is passed a point in time, represented by a `TimePoint` object, the context element is to be determined for. Both methods can furthermore be passed additional parameters, represented by a `QoS` object, specifying quality of service constraints which have to be observed during the context gathering and data transformation process within the sensor adaptor. We give a detailed description of the `QoS` data type shortly when explaining the parameters passed to the `ActiveSensor` interface's methods. The return value of both methods defined in the interface `PassiveSensor` is a `SensorContextElementResult` object. It is explained further on in Section 5.1.4.3. Please note that when an adaptor is created for a context sensor that only supports asynchronous queries and the adaptor by means of a storage of events received from the sensor becomes capable of processing synchronous requests as well, then the adaptor must accordingly implement the interface `PassiveSensor`.

Since an asynchronous sensor adaptor fires events, the interface `ActiveSensor` extends the interface `EventSource`. By means of this interface's methods clients can register themselves as listeners for events fired by an implementing class. In contrast to other event handling mechanisms, for example in the Java programming language [HoCo02], however, the mere registration of clients via these methods does not cause the adaptors to fire any events to them yet, but only indicates the general willingness of clients to receive events from the

respective event source. In order to induce a sensor adaptor to actually fire an event the interface `ActiveSensor` defines two `requestEvent()` methods. By means of these methods a client specifies particular conditions the fulfillment of which causes the adaptor to fire a corresponding event. These conditions are represented by the `entityId` and `specification` parameters of the `requestEvent()` methods. The `specification` parameter is a `ContextElementSpecification` object containing conditions the gathered context data are to fulfill. We come back to the `ContextElementSpecification` interface shortly, but first complete our explanations concerning the interface `ActiveSensor`. The `entityId` parameter, an `Object`, specifies the entity the context data gathered by the sensor adaptor are to refer to. Another parameter is a request identifier making it easier for the client to match an incoming event against the request the event refers to. One of the `requestEvent()` methods also allows for an explicit specification of the client the requested event is to be sent to by means of the `listener` parameter which is a `SensorContextElementEventListener` object. This data structure is described in Section 5.1.4.3. The usage of this method overrides the event source's default behaviour of sending events to all registered listeners. The `ActiveSensor` interface moreover defines a corresponding method to delete a previously made request. Events fired by `ActiveSensor` objects are `SensorContextElementEvent` objects which are also dealt with in Section 5.1.4.3.

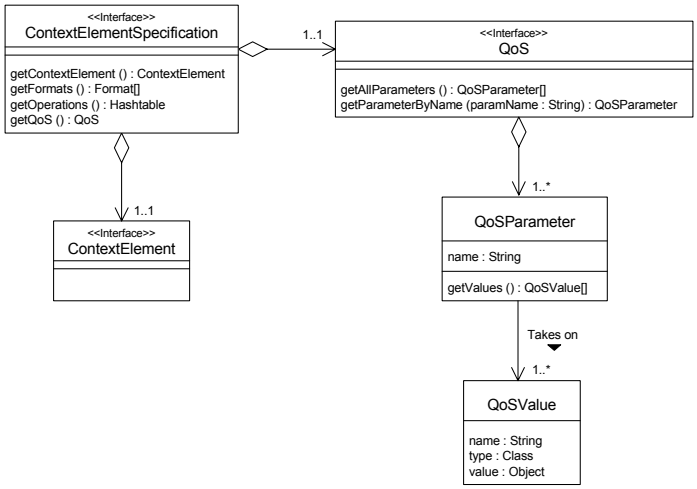

Figure 35: Context element specification

A `ContextElementSpecification` instance aggregates a `ContextElement` object as depicted in Figure 35. It thus serves as a means to specify the conditions a context element has to fulfill in order to cause an asynchronous sensor adaptor to fire an event. This specification can be made in such detail as required by assigning values to exactly those attributes of the contained `ContextElement` object which are relevant to the adaptor's client. In addition to this, the `ContextElementSpecification` object furthermore aggregates a `QoS` object. We have already mentioned this type in conjunction with the `PassiveSensor` interface as a `QoS` object is passed to the methods of this interface as well. The `QoS` interface is used to specify particular parameters concerning the execution of the method it is passed to. These parameters do not necessarily relate to a context element directly, but rather must be

taken account of by a sensor adaptor during the context gathering and data transformation processes it carries out. Currently, relevant parameters contained in the `QoS` object for sensor adaptors' execution are the minimum reliability of the context data supplied, the maximum response time of the adaptor, and an indication of whether additional information concerning the expected period of validity of the supplied context data is to be provided. These parameters reflect some of the criteria for the assessment of sensor adaptors identified previously and are again referred to in Section 5.1.4.2. The `QoS` interface aggregates one or more `QoSParameter` objects. These objects serve to represent particular execution parameters such as reliability or response time and can be accessed by means of the `QoS` interface's methods. Each `QoSParameter` instance is associated with one or more `QoSValue` objects representing the specified permissible values of this parameter.

The `ContextElementSpecification` interface furthermore possesses methods to access the formats of a context element. Formats – which are also passed as parameters to the methods of a `PassiveSensor` object as mentioned above – are required in conjunction with the context element of location. They allow a client to specify the coordinate system supplied location data are to refer to. Although no further applications of formats could have been identified so far, a `Format` interface has been defined in the object model to enable the model to be extended to other formats context elements may possess in the future. As can be seen in Figure 36, a `ContextElement` object may possess zero or more than one formats. The interface `Format` is associated with a `FormatValue` class which contains the name, type, and contents of a format's value. In its current range of application the `value` attribute of the `FormatValue` class is always assigned a `CoordinateSystem` object.

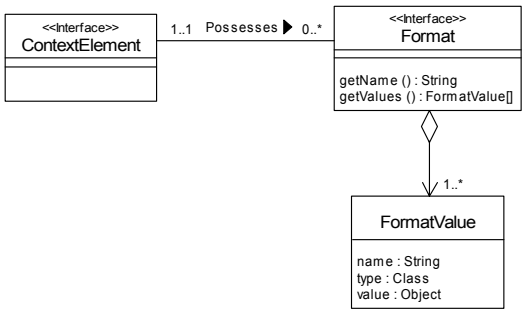

Figure 36: Context element formats

In addition to the formats described above, required return values of operations invoked with particular parameters on the gathered context data may be specified in a `ContextElementSpecification` object as well. The implicit default operation on the gathered context data is a comparison for equality with the context element passed to a sensor adaptor in a `ContextElementSpecification` object. If, however, a different or an additional operation is to be performed on the gathered context data, for example if a gathered location is required to be within a certain distance of another location or a physical condition or surroundings value must be greater or smaller than a given threshold, the particular operation that is to be evaluated by the adaptor along with its permissible return values need to be specified explicitly. For this purpose a `ContextElementSpecification` object also con-

tains a `Hashtable` object which can be obtained via the `getOperations()` method of the `ContextElementSpecification` interface. In this `Hashtable` object the operations the gathered context data need to comply with are specified and related to the `ContextElement` object or to that substructure of it they refer to.

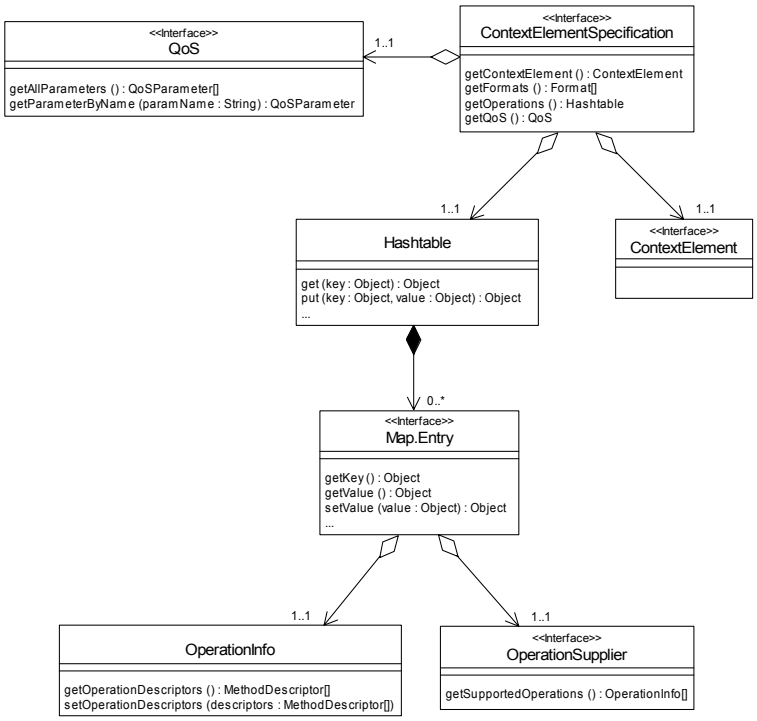

Figure 37: Complete context element specification

Since a specified operation needs not necessarily refer to an entire context element, but may also be applied to a particular object which is part of a `ContextElement` object, this explicit relation between operations and the structure they refer to is necessary. Consider, for instance, a `ContextElementSpecification` concerning the state and therein the physical condition of a person which demands that the person's pulse frequency must be greater than 130 beats per minute. In this case the operation does not refer to the entire context element as this is a `State` object which cannot be compared to the specified target value. It rather has to be applied to the particular `ConditionValue` object representing the person's pulse frequency. In our object model all classes that provide operations apart from a comparison for equality with another object of the same class therefore implement an interface called `OperationSupplier`. It contains a method to obtain all operations supported by the implementing class. Hence, the keys in the `Hashtable` instance contained in a `ContextElementSpecification` are `OperationSupplier` objects. Their belonging operations are represented by `OperationInfo` objects which constitute the values of the `Hashtable`. The class `OperationInfo` provides methods to obtain and to set information about operations represented by `MethodDescriptor` objects. Each instance of the `MethodDescriptor` class contains the name of the operation it represents as well as the names, types, and values

of its parameters and return value and further information. Thus, a `ContextElementSpecification` object may also aggregate a `Hashtable` object the keys of which are `OperationSupplier` objects representing the context element or a substructure of it the specified operations, represented by instances of the `OperationInfo` class in the `Hashtable`'s values, refer to.

Figure 37 shows again the `ContextElementSpecification` interface, at this point with all its associations. Since the `MethodDescriptor` class is part of the `java.beans` API [Hami97], we refrain from explaining this class in detail.

5.1.4.2 Representation of sensor adaptors' capabilities

As we have already mentioned, sensor adaptors register themselves with a service registry. This registry serves clients to discover appropriate adaptors for a particular request. Since sensor adaptors implement the `RegisterableService` interface, they possess a `ServiceDescription` object providing information about the adaptor, its interface and capabilities. This description is used by the service registry or, strictly speaking, the `ServiceMatchingService` object associated with it (see Section 4.1.5) for the discovery of suitable sensor adaptors by matching it against the `ServiceSpecification` object passed to the service registry by the client. Both the `ServiceDescription` and the `ServiceSpecification` classes have been defined as base utility classes in the information logistics framework. Their details are therefore beyond the scope of the Context Component and are not dealt with in this thesis. However, both classes possess a `capabilities` attribute which represents the offered and required abilities of a sensor adaptor, respectively. This attribute is a `Properties` object which needs to be assigned meaningful key-value pairs for the particular application area of a `ServiceDescription` or `ServiceSpecification` object. In this section we describe which entries the `capabilities` attribute contains if employed in conjunction with sensor adaptors. For a `ServiceDescription` object the following entries have been defined based on the criteria for sensor assessment:

- Key »ContextElementName«

  This entry provides information about the context element the sensor adaptor supplies data for. Its belonging value is a character string containing the name of the context element.

- Key »Downtimes«

  The value belonging to this key is a vector containing the times the sensor adaptor is not available at. The vector's elements are `TimeInterval` objects.

- Keys »MinResponseTime«, »AvgResponseTime«, »MaxResponseTime«

  These entries are only relevant for passive sensor adaptors. They provide information about the maximal, minimal, and average response time of the sensor adaptor in milliseconds. The data type of their values is `Long`.

- Key »Prediction«

  By means of this entry a sensor adaptor indicates whether it is able to determine context data with reference to future points in time. The entry's value is a `Boolean` object.

- Key »SensorCapabilities«

  The value belonging to this entry is a table containing further sensor adaptors' capabilities. The table has the following columns:

  - EntityIds: This column contains the identifiers of the entity instances context data can be gathered for, expressed as a vector of objects.
  - EntityTypes: This column contains the names of the entity types context data can be gathered for, represented by a vector of character strings.
  - Formats: A vector of `Format` objects in this column states the context element's formats the sensor adaptor can determine.
  - ValidityPeriod: The boolean value contained in this column indicates whether the sensor adaptor is capable of providing information about the gathered context data's expected period of validity.
  - MinReliability: In this column the minimal reliability of the context data supplied by the sensor adaptor is reported. Its data type is a `Float` object in the range between greater than zero to one with a value of one signifying the data's complete certainty.
  - MaxReliability: In this column the maximum reliability of the context data supplied by the sensor adaptor is indicated.
  - ContextElements: This column contains a vector of `ContextElement` objects the sensor adaptor can recognize. It serves as a means to optimize the discovery of suitable adaptors for an asynchronous request. Since such a request is accompanied by a `ContextElementSpecification` containing a `ContextElement` object that is to be detected, the information contained in this column helps to avoid supplying clients with references to sensor adaptors that cannot detect this particular instance. The elements of the vector may be either complete `ContextElement` objects in case the sensor supplies discrete, finite values or may be context element parts associated with a validity condition to express the value range of recognizable data.

  The rows of the table which is the value belonging to the »SensorCapabilities« key thus provide information about which context data can be supplied by the respective sensor adaptor for which entity types and instances in which formats with which reliability and whether information about the period of validity of these data can be given.

We have already mentioned that the `capabilities` attribute of a sensor adaptor's service description has to be updated when the adaptor is notified about changes concerning the context sensor it accesses via the `update()` method of the `ExternalEventListener` interface. In this case a reregistration of the sensor adaptor with the service registry has to be carried out as well to provide the registry with the updated `ServiceDescription` object.

For `ServiceSpecification` objects used by sensor adaptors' clients to specify the required properties of an adaptor the `capabilities` attribute contains the following entries:

- Key »ContextElementName«

  This key's value contains the name of the context element that is to be determined.

- Key »Mode«

  This entry is used to specify whether a synchronous or an asynchronous adaptor is needed. The value belonging to this key is an enumeration type containing the entries »active« and »passive«.

- Key »Prediction«

  By means of this entry a client states if it requires context data for a future point in time.

- Key »Formats«

  In this entry the required formats of the context element that is to be determined may be specified if applicable.

- Key »MaxResponseTime«

  If set the value belonging to this key stipulates the maximum allowable response time of the sensor adaptor.

- Key »EntityType«

  This entry can be employed to specify the entity type context data are to be gathered for.

- Key »EntityId«

  By means of this entry a client may announce that context data are to be gathered for a specific entity instance the identifier of which is contained in the value belonging to this key.

- Key »Reliability«

  This entry can be employed to specify the minimal reliability of the requested context data.

- Key »ValidityPeriod«

  The value belonging to this key may be used by a client to indicate that it requires information about the context data's expected period of validity.

- Key »ContextElement«

  By means of this entry a `ContextElement` instance can be provided the recognition of which is demanded from the sensor adaptor.

  If not stated otherwise the data types of these keys' values are the same as in the `ServiceDescription` object explained above.

5.1.4.3 Representation of sensor adaptors' results

As already mentioned, the values returned by the `PassiveSensor` interface's methods are `SensorContextElementResult` instances. They each aggregate a `ContextElement` object being the result of the context gathering process. This object can be obtained via the `getContextElement()` method of the `SensorContextElementResult` interface. In addition, information concerning the effective values of the `requiredQos` parameter passed to synchronous sensor adaptors when being queried is also provided in the result. For this purpose a `SensorContextElementResult` object also aggregates a `QoS` object which can be accessed via the `getEffectiveQoSValues()` method and may serve clients to assess adaptors' results more exactly and to optimize the further processing of these results.

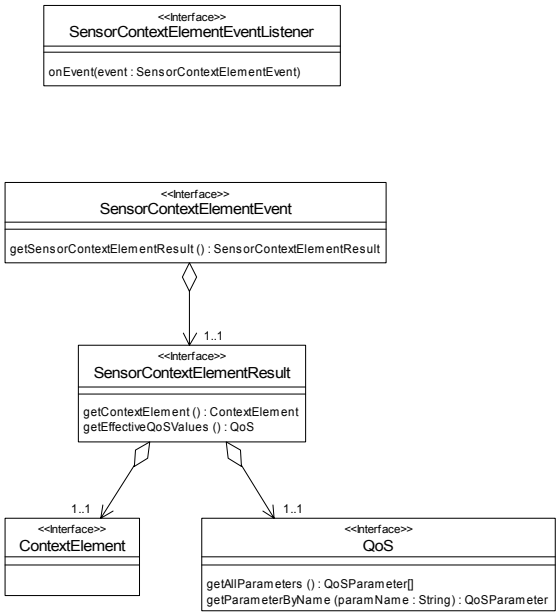

Figure 38: Sensor adaptors' results

Asynchronous sensor adaptors fire events of the type `SensorContextElementEvent`. These events each aggregate a `SensorContextElementResult` object and possess a method to access it. Our object model defines a corresponding interface `SensorContext-ElementEventListener` that must be implemented by classes wishing to receive this type of events. When an event arrives, the method `onEvent()` is called and passed the event as a parameter. Figure 38 shows the structures for the representation of sensor adaptors' results.

5.2 Virtual Sensors

We have already mentioned that the process of context gathering is composed of several tasks which have to be carried out one after the other. At the bottom level the gathering of data from context sensors and their transformation have to take place as described above.

The results gained from these tasks serve as a basis for the combination, pre-aggregation, and derivation of context data. These functions are performed by what we call virtual sensors. In this section we first of all describe the individual duties of virtual sensors in greater detail. Since the derivation of context elements is an important and potentially complex issue, we give this task special attention by illustrating the mechanisms that can be employed to carry out context data derivation and after that by describing our approach to a dynamic implementation of context data combination and derivation mechanisms within the Context Component. In addition, this section, too, contains an object model for the integration of virtual sensors into the Context Component.

Definition 16: Virtual Sensor

A virtual sensor is a piece of software that obtains data from sensor adaptors and, if need be, combines and pre-aggregates these data. It furthermore uses them to perform context data derivation. A virtual sensor provides a uniform interface to clients, as a result of which the existence of sensor adaptors is hidden from higher layers.

Virtual sensors therefore are the clients of sensor adaptors. They augment the data obtained from them and make the results of this processing available to other parts of the Context Component. Clients can interact with virtual sensors without any knowledge of the existence of sensor adaptors. This separation of concerns hides the complexity of the tasks involved from virtual sensors' clients and improves the maintainability, stability, and flexibility of the Context Component.

5.2.1 Context data combination

The combination of context data obtained from several sensor adaptors is needed to relate the data provided by one or more sensor adaptors to a different entity than the one they are supplied for. We have already mentioned an example of this task: The data of a sensor adaptor providing GPS coordinates for company cars can be combined with data regarding the current reservations of these cars to obtain the locations of persons in the form of geographic coordinates. Another example is determining a person's reachability by means of a sensor adaptor that provides information about this person's location which is combined with the data of an adaptor supplying information concerning the reachability of this location, i.e. the communication media installed at it. In these cases the context data provided by a sensor adaptor correspond to the entities a different adaptor's data refer to. By means of equating these data the combination of several sensor adaptors' context data is carried out.

5.2.2 Context data pre-aggregation

Another essential part of the context gathering process is the pre-aggregation of context data obtained from various sensor adaptors. It is often the case that the Context Component is requested to determine several pieces of information concerning an entity's context. On the one hand this occurs whenever the Context Component is requested to ascertain more than one context element or even the entire context of an entity as far as it can be detected. On the other hand there are also context elements which can be detected by more than one sensor adaptor each of which may provide – possibly different, complementary or contradic-

tory – results regarding the context element of an entity. In both cases it is necessary to fit together several sensor adaptors' data to take all requested context elements and all available elements, attributes, and values of each context element into account.

This pre-aggregation of context data differs from their combination by the fact that data which are pre-aggregated refer to the same entity, whereas data that are combined do not. The pre-aggregation of context data thus connects these data in parallel; in contrast to this, data combination may be a multi-stage process during which data are first obtained from one sensor adaptor and are then employed as a parameter for a request made to a different adaptor or for the evaluation of its results. The pre-aggregation may therefore be regarded as a horizontal integration of context data, while context data combination means a vertical integration. In order to process a request made to the Context Component context data pre-aggregation and combination may also be used in parallel by pre-aggregating data gained from a combination process with context data obtained from other sensor adaptors.

The term pre-aggregation of context data is used in conjunction with virtual sensors, because at this level of the context gathering process requests made to virtual sensors are forwarded to an arbitrary number of sensor adaptors. Virtual sensors thereupon receive context data from several adaptors and fit them together in a vector as we explain in greater detail in Section 5.2.5.1. A filtering of these data and their final aggregation into context or context element instances that are returned to the Context Component's clients takes place after that on the next higher level; these tasks will be dealt with in Section 5.3.

The central localization of context data combination and pre-aggregation at the virtual sensor level frees higher layers within the Context Component from the task of interacting directly with sensor adaptors and the service registry managing them. In addition to this, virtual sensors' clients are provided with context data of higher value without having to concern themselves with the way they are generated. As a result, the maintainability of the Context Component is increased. Likewise, the component's complexity is reduced, because higher layers need to interact with one single application part only in order to obtain context data.

5.2.3 Context data derivation

In the previous two sections mechanisms have been described which augment the data provided by individual sensor adaptors without changing them. The derivation of context data goes a stage further than this. When context data are derived, data received from sensor adaptors serve as a basis for inferences about entities' context elements that differ from those the received data refer to. Context data derivation also takes into account the interdependence among context elements' attribute values that exists when an attribute value of a particular context element determines another context element's attribute value or value range. As we have outlined in Section 5.1.1, derivation mechanisms can contribute to context gathering to a great extent; in the case of the context element of emotional condition context derivation even is the only means by which this context element can be detected.

The derivation mechanisms usable in a particular application strongly depend on the application domain and the specific conditions under which the application is used. For this reason no complete enumeration or classification of these mechanisms can be given. In the following

subsections we rather expound the principles of context data derivation and illustrate them by some examples in order to give an overview of the potential application areas of context data derivation.

5.2.3.1 Derivation of the context element location

Since numerous context sensors are available for the detection of entities' locations, the derivation of location data from other context elements only plays a minor role. Nevertheless, it is possible to determine the context element of location by means of derivation mechanisms as well. This is particularly reasonable for locations that cannot be recognized by the available sensors or may be employed to increase the value of location data by adding the results gained from derivation mechanisms to the data received from existing location sensors.

Location data can in principle be derived from all other context elements and combinations of them. If under certain prerequisites such as the persons involved, the reference to a project, the time of day, and so on an activity (or motion, reachability, etc.) generally takes place at a particular location, location data can be inferred from the existence of this activity (motion, reachability, etc.) and of the other factors at a particular point in time. If, for example, in a company meetings concerning the project »Intranet« always take place in the project manager's office, then when a project team member's activity has been detected to be »meeting« with a project attribute the value of which is »Intranet«, it can be inferred that this person is in the office of the project manager. In the same manner, when a person's reachability contains a communication medium that is available only at one particular place or at a few places, this information can – if necessary after combining it with other context data such as the building this person is located in – be used to infer the person's current room. Depending upon how tightly the existence of other context elements is tied to a particular location such derivation mechanisms may possess a varying degree of reliability. If, for instance, a salesperson feels especially comfortable at a particular customer's, this salesperson's emotional condition of well-being may admit of inferring that she is at this customer's. However, the location data received from this derivation only possess a small degree of reliability.

5.2.3.2 Derivation of the context element state/motion

An entity's motion can also be derived from all other context elements and their combinations. When an entity is located in a means of transport, its motion is that of travelling. The motion of visiting can be derived from the fact that the entity is at a fixed location outside its home location. An entity that is reachable via various Wireless LAN Access Points in quick succession is very probably wandering, to name but a few examples.

The derivation of an entity's motion from other context elements than location, and the derivation of temporary motions are often less reliable. It may thus be reasonable to combine several sensor adaptors' data to increase the reliability of a derived motion. A chair's physical condition, i.e. the weight effected on it, can, for instance, be combined with its location and the persons who are at this location along with, if necessary, their activities to derive from these data that a person is sitting on a chair and her motion therefore is »stationary«.

5.2.3.3 Derivation of the context element state/activity

A very common example of the derivation of persons' activity is to consider a person to be meeting whenever she is in a meeting room and at least one other person is present [NeEl91]. In this case location data serve as the basis for the derivation of this context element. Yet, provided that particular values of other context elements are strongly connected with particular activities, all other context elements and combinations of them may also be employed to derive an entity's activity. Another example based on data concerning both the context element of location and that of physical condition is to define that when Mr. Miller is in the city park and his pulse rate is at least 130 beats per minute, then Mr. Miller's activity is »jogging«.

5.2.3.4 Derivation of the context element state/physical condition

Similar to locations the physical condition of entities is mainly detected by context sensors which inherently provide this kind of data. Consequently, the derivation of entities' physical condition is of minor importance. It is chiefly employed if no suitable sensors are available or if the expressiveness and reliability of sensor data are to be increased by augmenting them with results of derivation mechanisms.

The only context element entities' physical condition can very rarely be derived from is that of reachability. All other context elements may quite possibly, provided that conditions analogical to those mentioned above when discussing the derivation of location and activity are fulfilled, serve as a basis to derive physical conditions. From the fact that a person is in a gymnasium or in bed, for example, conclusions regarding certain value ranges for their pulse and breath frequency etc. can be drawn. Emotional conditions are often accompanied by typical characteristics of bodily functions as well. In addition, the physical condition of entities can also be inferred from activities, motions, or the surroundings of these entities. The derivation of physical condition, though, is often tainted with a certain unreliability as it is usually restricted to providing only value ranges as results.

5.2.3.5 Derivation of the context element state/emotional condition

As already mentioned, the determination of this context element is fully dependent on derivation mechanisms. Inferences concerning persons' emotional condition can in particular be drawn from their physical condition. The skin conductivity, pulse or breath rate of a driver may, for example, be taken as input values in order to derive a driver's exposure to stress [HePi00]. The physical condition of objects, combined with data about who is using them, may also provide information regarding persons' emotional condition. The amount of pressure exerted on a telephone receiver while making a phone call, for instance, can serve as a measure for the strain of the person using the phone [Pica00]. In addition, derivations can be made on the basis of people's activities such as frowning or head-shaking. Further examples of how emotional condition can be derived are given in [CoDo00] and [CoKa98].

A derivation of emotional condition based on the context elements of location, motion, surroundings, and – to a smaller extent – reachability can possibly be carried out as well (e.g. when Mr. Miller is at the dentist's, he is very nervous). Such mechanisms depend, however, to a high degree on the application domain and the individual persons they are applied to.

5.2.3.6 Derivation of the context element reachability

Deriving an entity's reachability from other context elements is rather unlikely to be possible. Usually at least a minimum knowledge of existing communication media is required in order to make meaningful inferences concerning this context element which means that suitable context sensors have to be available. It can, for example, be derived that a person writing a letter on her computer is reachable via this computer. Without a corresponding context sensor that provides data about the reachability attributes and in particular the address attributes of this computer, however, this information alone is of little use. For this reason derivation mechanisms for the context element of reachability can only be employed to a limited extent and have in any case to be combined with data supplied by existing reachability sensors.

5.2.3.7 Derivation of the context element surroundings

The determination of this context element is mainly based on context sensors. For many application domains derivation mechanisms for entities' surroundings cannot be identified or possess a very limited reliability. In certain application domains, however, it may perhaps be sensible to make use of such derivations by associating other context elements' attribute values with particular data about the environment. An information logistic application supporting firemen may, for instance, derive from the activity »running« of the fire engine's water pump and from the fact that the fire engine is somewhere away from the fire station that there is fire in the firemen's surroundings.

5.2.4 Definition of rules for context data combination and derivation

The examples of context data combination and derivation given in the previous sections have made it clear that these mechanisms often strongly depend on the application domain of an information logistic application, the specific conditions prevailing in this domain, and on the individual entities, in particular users, the mechanisms refer to. Organizational procedures, individual preferences and habits of people, economic conditions, and many other factors considerably affect how and on the basis of which data context elements can be combined and derived. What is more, these conditions may be subject to rapid changes. For this reason the way combination and derivation mechanisms are implemented has to take the need for a consideration of application-specific conditions and for a quick changeability of the mechanisms into account. Above all, this means that instructions concerning context data combination and derivation cannot be implemented once before the information logistic application is deployed on the assumption that they will last. Rather an opportunity for a dynamic, application-specific definition and modification of combination and derivation mechanisms has to be provided. It should enable qualified personnel of the application's operator or even users themselves to define how context data are to be combined and derived. As a consequence, changes can be reacted to more quickly and with reduced maintenance costs. In addition, the application's acceptance is increased by making context gathering processes more transparent and enabling people to define exactly those mechanisms they need the way they need them. In this section we thus describe our approach to an implementation of combination and derivation mechanisms that meets the abovementioned requirements.

The dynamic processing of combination and derivation mechanisms resembles the processing of rules in rule engines [Tous03]. Both approaches have in common that conclusions are drawn from a given set of facts, the application logic being externalized which enables declarative programming. Conditions are checked against a knowledge base – which in our case is the data supplied by sensor adaptors –, and actions are carried out if these conditions are fulfilled. Regarding virtual sensors this means that they create or gather context data by means of combination and derivation processes and make them available to their clients. For this reason it suggests itself to make use of existing mechanisms employed for rule definition and processing in the area of rule engines. Due to the great diversity of existing rule languages the need for a standardized general representation of rules has been emerging [Wagn02]. The provision of a vendor-neutral, open language standard based on an XML representation of rules is the goal of the Rule Markup Initiative the participants of which are developing Rule Markup Language (RuleML) [HiBo04], [BoTa01]. At present time, however, the expressiveness of RuleML is still very limited, and none of the results achieved so far possesses a sufficient maturity and power to allow it to be adopted for our purpose.

We have decided to represent application-specific combination and derivation mechanisms by means of eXtensible Markup Language (XML) [BrPa04] files which are read, interpreted, and executed by virtual sensors at run-time (see also Section 5.2.5). In these XML files rules specifying how to combine and derive context data are defined. For this purpose we have created a language based on the Simple Rule Markup Language (SRML) developed at ILOG, S.A. [Cove01], [Thor01]. The language defined by us adds constructs to SRML required for the combination and derivation of context data in consideration of the Context Component's specific conditions which cannot be represented by the current scope of SRML. The resulting language is called Context SRML; its Document Type Definition (DTD) is quoted in the Appendix. The modifications we have made to SRML are marked in colour in the DTD.

The root element of a Context SRML document is a `ruleset` which consists of a set of rules. Each rule has a description as well as a condition part and an action part. A rule's description serves to facilitate the discovery of those rules that are to be taken into consideration during the processing of a request made to a virtual sensor. In the condition part of a rule at least one condition has to be specified. Context SRML distinguishes between simple conditions, not conditions (both adopted from SRML), lookup conditions, and multistage conditions. The first three types of conditions consist of expressions on the basis of which data concerning sensor adaptors are examined, while multistage conditions associate a condition referring to data provided by sensor adaptors (a simple condition or a not condition) with a condition specifying parameters for the discovery of suitable adaptors (a lookup condition). They may furthermore contain instructions for the storage of sensor adaptors' results in a so-called `resultHash`. A rule's action part consists of actions which may be variable declarations or assignments as well as instructions to create, modify, or delete objects. This part of the language has entirely been adopted from SRML without modifications. The mentioned expressions made use of throughout all parts of a rule can be variables or fields to which values can be assigned, constants, arithmetic or boolean expressions as well as operations on objects.

In the following we explain Context SRML in greater detail by illustrating its syntax, scope, and expressiveness with the help of two exemplary rules. In the first example a rule for the combination of data concerning the context elements of surroundings and location is

described which allows to gather data referring to entities' surroundings. In natural language this rule states that when an entity is within a radius of two kilometres of a surroundings sensor, then the data supplied by this sensor refer to the entity's surroundings.

A rule's description first specifies the context elements that are supplied as results of the rule's execution. This is done by means of an element named `contextElement` which has an attribute providing the name of the respective context element's class. In the present case this is the context element of surroundings. The element may also contain an enumeration of the individual context element instances that may result from the rule's execution. Since this is not required for the rule we are currently dealing with, because any possible surroundings data are covered by it, we explain this option later on in connection with the second rule we describe. Furthermore, information concerning the formats of the supplied context elements may be contained in a `contextElement`. Since formats currently apply to the context element of location only, this optional element is not contained in the examples we present in this section. As can be seen in the Appendix, however, the `format` element simply consists of an expression specifying the individual formats that can be provided in relation to a context element. In addition to the supplied context elements, a rule's description may also provide information regarding the reliability of the mechanisms defined in the rule, its priority, and the entity instances the rule refers to. Since the rule currently studied is not limited to specific entities, this part of a rule's description is also explained later on. Below the description part of our first exemplary rule is shown.

```
<ruleset>
   <rule name="SurroundingsExample">
      <description>
         <priority>
            <constant type="int" value="2"/>
         </priority>
         <reliability>
            <constant type="float" value="0.9"/>
         </reliability>
         <contextElement className="State.Surroundings"/>
      </description>
   ...
   </rule>
</ruleset>
```

The condition part of the rule first instructs the virtual sensor processing it to obtain context data concerning the context elements of surroundings and location by means of the `simpleCondition` element. This element's `className` attribute specifies the context element that is to be obtained, and the optional `objectVariable` attribute represents, if present, the instruction to assign the received results to a variable with the specified name. While the first simple condition refers to context data obtained from sensor adaptors, the second one is related to data contained in the adaptors' `ServiceDescription`. This is expressed by means of the `lookupCondition` element which demands that data supplied by the sensor adaptors employed to gather the context element of surroundings have to refer to the entity type location. The body of the lookup condition consists of two binary expressions instructing the virtual sensor to check whether the supplied context element's class equals the specified constant value »State.Surroundings« and to ensure that the entity type the data refer to equals the constant »Location«. Since in this case conditions concerning the data supplied by

sensor adaptors are associated with conditions referring to the adaptors themselves, these conditions are enclosed by a `multistageCondition` element. This makes it easier for the virtual sensor to recognize that the simple conditions and the lookup condition the multistage condition is composed of relate to each other. After the data have been obtained the virtual sensor is furthermore instructed to create a `Hashtable` object and to insert those matching pairs of location and surroundings data into this object that fulfill the abovementioned condition, i.e. to relate surroundings data to the location they have been detected at. The section of the rule's condition part which represents these instructions is quoted below.

```
<conditionPart>
   <multistageCondition>
      <simpleCondition className="de.fhg.isst.ilog.context.state.Surroun-
                                 dings" objectVariable="surr"/>
      <simpleCondition className="de.fhg.isst.ilog.context.location.Loca-
                                 tion" objectVariable="sensorLoc"/>
      <lookupCondition className="de.fhg.isst.ilog.frame.util.ServiceDe-
                                 scription">
         <binaryExp operator="equals">
            <field name="ContextElementName"/>
            <constant type="string" value="State.Surroundings"/>
         </binaryExp>
         <binaryExp operator="equals">
            <field name="EntityType"/>
            <constant type="string" value="Location"/>
         </binaryExp>
      </lookupCondition>
      <resultHash>
         <variable name="surr"/>
         <variable name="sensorLoc"/>
      </resultHash>
   </multistageCondition>
   ...
</conditionPart>
```

After that a further condition specifies that an entity's location must be within a radius of two kilometres of a location surroundings data have been received for. For this purpose again a simple condition is employed. It instructs the virtual sensor to obtain data about the location of the entity currently under consideration, i.e. the entity specified in the request made to the virtual sensor on the basis of which this rule is executed, from appropriate sensor adaptors. Please note that the condition does not explicitly specify the identifier of the entity location data are to be obtained for. Context SRML generally stipulates that if not explicitly specified otherwise requests to sensor adaptors are to be made with those parameters virtual sensors have been passed when queried themselves. This specification keeps the syntax of the language simple and manageable and avoids unnecessary information. The body of the simple condition is composed of two binary expressions nested in another one. They additionally demand that the `quantity` attribute of the `Distance` object returned by the `getDistanceTo()` method of the `Location` object representing the entity's whereabouts must have a maximum value of two, measured in kilometres. This method is to be passed each location contained in the `Hashtable` object created before, as indicated by the `method` element. The parameters that are to be passed to the method are specified by the two inner binary expressions with the `field` element signifying the parameter name and the following `con-`

stant or variable element representing the parameters' values, in this case the location surroundings data are available for and the required unit of measurement. As a result of the examination of this condition those locations that do not meet the specified requirement are deleted from the Hashtable object along with their associated surroundings data. This is done because all conditions in a rule's condition part except from not conditions are implicitly connected by a conjunction, and therefore the entries in the Hashtable object may only be preserved if they fulfill this second condition as well.

```
<conditionPart>
   ...
   <simpleCondition className="de.fhg.isst.ilog.context.location.Location">
      <binaryExp operator="smallerOrEquals">
         <method name="getDistanceTo">
            <binaryExp operator="equals">
               <field name="measurement"/>
               <constant type="string" value="km"/>
            </binaryExp>
            <binaryExp operator="equals">
               <field name="otherLoc"/>
               <variable name="sensorLoc"/>
            </binaryExp>
         </method>
         <field name="quantity">
            <constant type="float" value="2.0"/>
         </field>
      </binaryExp>
   </simpleCondition>
</conditionPart>
```

Both the multistage condition referring to the surroundings data obtained from sensor adaptors and the simple condition referring to the location of entities have to be fulfilled to execute the rule's action. Concerning our example this means that the client is returned those surroundings data that refer to a location not farther than two kilometres away from the location of the entity under consideration. This instruction is specified by the assertobj element in the rule's action part. It induces the virtual sensor to return the variable named sensorLoc.surr which refers to the entries of the Hashtable object constructed before. As explained above, matching pairs of location and surroundings data have been inserted into this Hashtable object according to the first condition. During the examination of the second condition those entries that do not meet its specified requirements have been removed. Therefore, the instruction contained in the rule's action part can be limited to simply returning the variable to the client. The action part of the rule is shown below.

```
<actionPart>
   <assertobj>
      <variable name="sensorLoc.surr"/>
   </assertobj>
</actionPart>
```

By means of a second example we illustrate a rule for the derivation of context data from data referring to other context elements. For this purpose the Context SRML representation of the rule already mentioned in Section 5.2.3.3 in a slightly different form is explained. In

natural language it can be formulated as follows: »If the users represented by the identifiers 4711 and 4712 are in the city park and their pulse rate is at least 130 beats per minute, then these users are jogging.«

In this case, apart from information about the rule's priority and reliability as well as the name of the context element provided by its execution, the description of the rule additionally contains information about the context element instance that can be detected by means of this rule. This is due to the fact that in contrast to the rule explained before in this case not all possible values of the given context element can be detected, but only the activity »jogging«. As can be seen below, the rule's description also provides information regarding the individual entity instances the rule can be applied to. The `entity` element is made use of for this purpose; it possesses an attribute specifying the name of the class the entity's type is represented by. Within this element the possible values of the entity instances' `id` attribute are set. These individual entity identifiers are connected to each other by a disjunction signifying that the rule can be applied to any of the given entity instances.

```
<ruleset>
   <rule name="JoggingExample">
      <description>
         <reliability>
            <constant type="float" value="0.75"/>
         </reliability>
         <entity typeClassName="User">
            <naryExp operator="or">
               <binaryExp operator="equals">
                  <field name="id"/>
                  <constant type="int" value="4711"/>
               </binaryExp>
               <binaryExp operator="equals">
                  <field name="id"/>
                  <constant type="int" value="4712"/>
               </binaryExp>
            </naryExp>
         </entity>
         <contextElement className="State.Activity">
            <binaryExp operator="equals">
               <field name="name"/>
               <constant type="string" value="jogging"/>
            </binaryExp>
         </contextElement>
      </description>
      ...
   </rule>
</ruleset>
```

The first condition in the rule's condition part induces the virtual sensor processing this rule to obtain data referring to the context element of location from suitable sensor adaptors. Furthermore, the virtual sensor is to examine whether the locations represented by the data equal the city park by comparing the locations' attributes with given values. In addition to this, the sensor adaptors employed are required to supply data with a reliability of at least 80 per cent and with reference to a coordinate system representing premises. Therefore, this first condition is a multistage condition.

```
<conditionPart>
   <multistageCondition>
      <simpleCondition className="de.fhg.isst.ilog.context.location.Loca-
                                 tion">
         <binaryExp operator="equals">
            <field name="Coordinates.Value.Value"/>
            <constant type="string" value="city park"/>
         </binaryExp>
         <binaryExp operator="equals">
            <field name="Coordinates.Value.Dimension.Name"/>
            <constant type="string" value="Name"/>
         </binaryExp>
      </simpleCondition>
      <lookupCondition className="de.fhg.isst.ilog.frame.util.Service-
                                 Description">
         <binaryExp operator="greaterOrEquals">
            <field name="Reliability"/>
            <constant type="float" value="0.8"/>
         </binaryExp>
         <binaryExp operator="equals">
            <field name="Format"/>
            <constant type="class" value="de.fhg.isst.ilog.context.location.
                                          PremisesCoordinateSystem"/>
         </binaryExp>
      </lookupCondition>
   </multistageCondition>
   ...
</conditionPart>
```

The second condition defined in this rule causes data concerning the context element of physical condition to be obtained from sensor adaptors. These data, too, are to be compared to certain values specified in the expressions the condition consists of.

```
<conditionPart>
   ...
   <simpleCondition className="de.fhg.isst.ilog.context.state.Physical-
                              ConditionSet">
      <binaryExp operator="equals">
         <field name="PhysicalConditionType.ConditionValue.Name"/>
         <constant type="string" value="pulse rate"/>
      </binaryExp>
      <binaryExp operator="greaterOrEquals">
         <field name="PhysicalConditionType.ConditionValue.Value"/>
         <constant type="int" value="130"/>
      </binaryExp>
   </simpleCondition>
</conditionPart>
```

If the rule's conditions are fulfilled, its action part stipulates by means of the `assert` element that an `Activity` object is to be created. The `assignment` element contained in this instruction furthermore induces the virtual sensor processing the rule to assign the character string »jogging« to the `name` attribute of this object. After this object is created, it is returned to the virtual sensor's client. The action part of the rule is shown below.

```
<actionPart>
   <assert className="State.Activity">
      <assignment>
         <field name="name"/>
         <constant type="string" value="jogging"/>
      </assignment>
   </assert>
</actionPart>
```

The definition of rules for the combination and derivation of context data can on the one hand be carried out by manually creating Context SRML files and making them available to virtual sensors. This approach, however, is only suitable for administrators with knowledge of programming or XML. To enable users to define customized rules and to facilitate the task of rule definition the creation and modification of rules may also be performed by means of a graphical user interface that generates Context SRML files according to user input. The design and implementation of such an interface, however, is beyond the responsibility of the Context Component. Therefore, we do not deal with this topic in greater detail within the scope of this thesis, but restrict ourselves to ensuring that the Context Component is able to provide all information needed by other components to implement such a user interface.

5.2.5 An object model for virtual sensor integration

Analogous to Section 5.1.4 in this section an object model for virtual sensors as another building block for context gathering is presented. Since virtual sensors are the clients of sensor adaptors, the object model described in the following refers to a layer of the Context Component which is superordinate to that of sensor adaptors.

5.2.5.1 Representation of virtual sensors

As explained in the previous sections, a virtual sensor combines and/or pre-aggregates data received from sensor adaptors and derives – possibly on the basis of rules – one or more context elements from them. These tasks of virtual sensors can also be applied in combination with each other. Since virtual sensors combine and pre-aggregate context data, they are – in contrast to sensor adaptors – not limited to providing data which refer to one single context element each.

The representation of virtual sensors in our object model is shown in Figure 39. The model defines a `VirtualSensor` interface that has to be implemented by every virtual sensor. This interface contains basic methods which serve to access a virtual sensor's name and identifier. Since all virtual sensors obtain data from sensor adaptors, they interact with the service registry managing the available adaptors. The service registry is employed by virtual sensors to receive references to sensor adaptors capable of satisfying a specific request.

Like the `Sensor` interface the interface `VirtualSensor` has two subinterfaces for virtual sensors that support synchronous and asynchronous queries, respectively. The interface `PassiveVirtualSensor` serves to represent virtual sensors that can be queried synchronously. It defines methods which enable a client to query a particular context element – specified by a character string in the `contextElementName` parameter – or the entire context of an

entity. A query referring to an entity's entire context is carried out by querying all available context elements of the entity. At this stage of the context gathering process no `Context` object is created by passive virtual sensors, because the final aggregation and filtering of context data is still to take place as we explain in Section 5.3. The requests made to synchronously query virtual sensors may again refer to the present moment or to a different point in time specified by a `TimePoint` object. Further parameters of these methods are the identifier of the entity the context or context element of which is to be determined, the required formats of context elements, and required quality of service values concerning the virtual sensors' execution. These parameters have already been explained in conjunction with sensor adaptors. The return value of the `PassiveVirtualSensor` interface's methods is a vector of `SensorContextElementResult` objects.

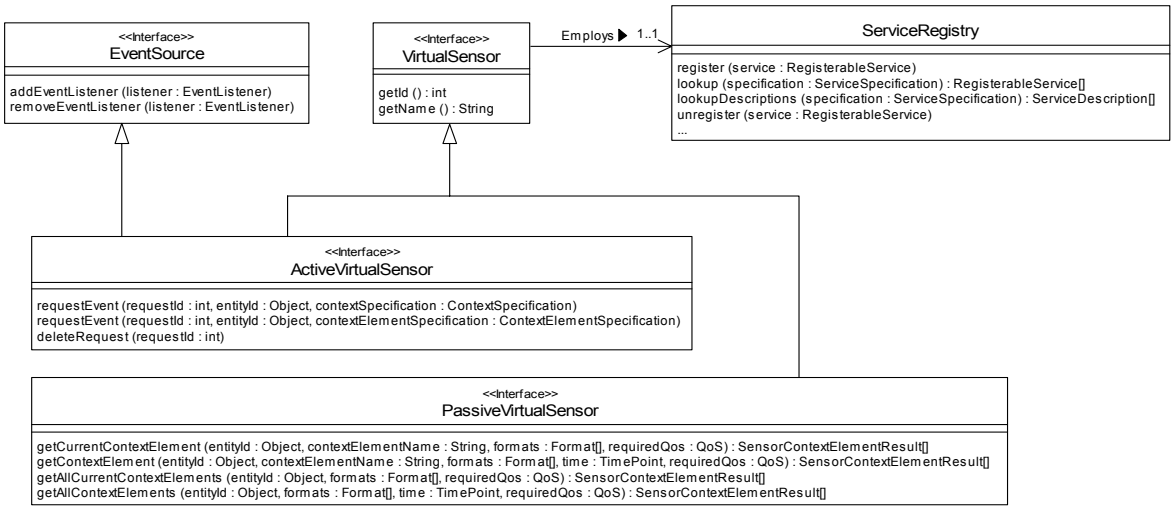

Figure 39: Virtual sensors

The interface `ActiveVirtualSensor` is implemented by virtual sensors that can be queried asynchronously. It defines methods by means of which clients can register themselves for particular events with a virtual sensor or delete previously made requests. These methods are similar to those of asynchronous sensor adaptors; for this reason we do not explain the corresponding parameters in detail again in this section. However, there are two significant differences to the interface for asynchronous sensor adaptors: On the one hand, since a virtual sensor is able to provide data for more than one context element, it is not only capable of firing events related to a single context element, but can also fire events with respect to entire contexts. Hence, apart from a `requestEvent()` method which is passed a `ContextElementSpecification` instance the `ActiveVirtualSensor` interface defines another `requestEvent()` method containing a `ContextSpecification` object as a parameter. The `ContextSpecification` class is explained shortly. In addition to this, virtual sensors' events are always sent to a single dedicated client. Therefore, a parameter specifying a particular listener is not needed in the `requestEvent()` methods and hence is omitted. Yet, the interface `ActiveVirtualSensor` also extends the `EventSource` interface by means of which a listener makes itself known to virtual sensors. The events `ActiveVirtualSensor`

objects fire are either `SensorContextElementEvent` objects – if an event with regard to a context element is requested – or `SensorContextEvent` objects in case the request refers to an entire context. The class `SensorContextEvent` is explained in Section 5.2.5.4.

In a `ContextSpecification` object the conditions a context has to fulfill to cause an event to be fired by an `ActiveVirtualSensor` are specified. Since a context is represented by an aggregation of context elements, a `ContextSpecification` object consists of one or more `ContextElementSpecification` objects as shown in Figure 40 and possesses a method to obtain them. Each `ContextElementSpecification` object represents the conditions made to a particular context element the context is to contain.

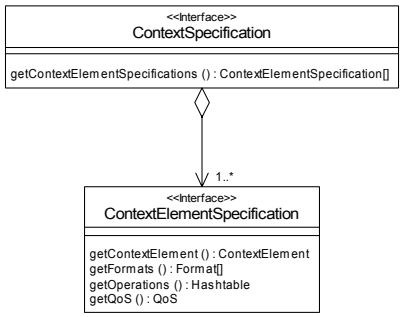

Figure 40: Context specification

The object model for virtual sensor integration specifies that synchronous virtual sensors are committed to querying synchronous sensor adaptors only, while asynchronous virtual sensors solely consult asynchronous sensor adaptors. This restriction facilitates the communication within the Context Component, because a temporary storage of events within virtual sensors as well as a cyclic querying of synchronous sensor adaptors become unnecessary. If required, such transformations are only permitted to take place within the adaptors themselves.

5.2.5.2 Representation of structures for the creation of virtual sensors

In contrast to sensor adaptors virtual sensors are not managed in a service registry, but rather are created dynamically by specialized classes. Since requests made to virtual sensors may refer to any possible context element or to combinations of context elements and since in addition a previously unknown number of rules is to be taken into consideration for the processing of a request, the management of virtual sensor instances in a service registry is neither reasonable nor practicable. It would require the Context Component to create virtual sensor instances in advance which due to the great diversity of potential request parameters' values and of existing Context SRML rules cannot be carried out with reasonable effort.

Instead, for the creation of virtual sensor instances the Abstract Factory Pattern [GaHe94] is employed. Clients wishing to make requests to virtual sensors first contact a specific object responsible for the creation of an appropriate virtual sensor instance that is able to fulfill the client's request. The structures necessary to implement this mechanism within the Context Component are illustrated in Figure 41.

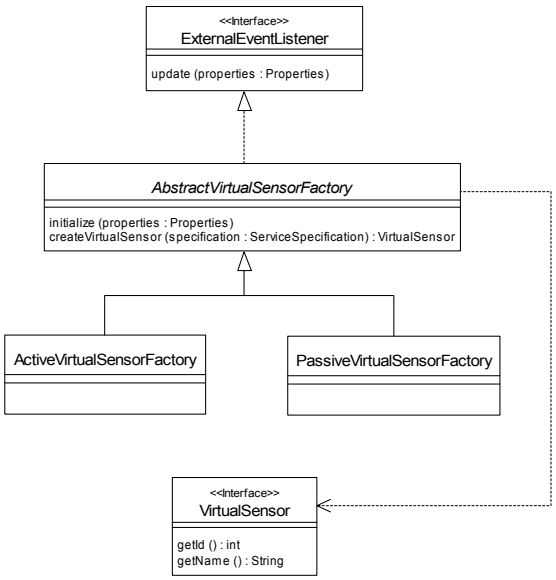

Figure 41: Virtual sensor creation

For the purpose of `VirtualSensor` object creation the class `AbstractVirtualSensorFactory` has been defined. It is an abstract class containing general methods needed to create virtual sensor instances. Two concrete factories, each responsible for the creation of virtual sensors supporting a particular request mode, inherit from the `AbstractVirtualSensorFactory` class: An `ActiveVirtualSensorFactory` class which serves to create `ActiveVirtualSensor` objects and a `PassiveVirtualSensorFactory` class responsible for the creation of `PassiveVirtualSensor` objects. A client can initiate the creation of a `VirtualSensor` object by means of the `create()` method defined in the `AbstractVirtualSensorFactory` class. When calling this method, the client passes a `ServiceSpecification` object containing the capabilities a virtual sensor is required to possess to the factory. The structure of this object's `capabilities` attribute differs from the `capabilities` attribute used in connection with sensor adaptors. In the following section we describe how this attribute looks like when being employed in conjunction with virtual sensors. On the basis of the capabilities specified in the `ServiceSpecification` object the factory searches for relevant Context SRML rule files and dynamically creates a `VirtualSensor` object which is forwarded these rules. This `VirtualSensor` object is then returned to the client which can make the desired request to it. Apart from the `create()` method explained above the class `AbstractVirtualSensorFactory` also contains a method named `initialize()`. During its initialization a factory reads the existing rule files and parses their description part. On the basis of the information provided in this part of the rules the factory can determine whether a rule suits the client's requirements and is to be passed to the virtual sensor the factory creates.

Like the interface `Sensor` the `AbstractVirtualSensorFactory` class implements the interface `ExternalEventListener`. As a result, a factory can be notified via this interface's `update()` method of changes made to existing rule files.

5.2.5.3 Representation of virtual sensors' capabilities

As mentioned above, the `AbstractVirtualSensorFactory` class' `create()` method is passed a `ServiceSpecification` object by means of which the factory finds suitable rule files and creates a `VirtualSensor` instance capable of processing the request the client intends to make. In the `capabilities` attribute of the `ServiceSpecification` object the client specifies the properties the virtual sensor is required to possess. Since in contrast to sensor adaptors virtual sensors are able to provide data referring to more than one context element and to entire contexts as well, this attribute's structure differs from the one defined in conjunction with sensor adaptors. A mere listing of the required capabilities in parallel is no longer possible, because these capabilities may refer to more than one context element and may be different for each. Therefore, the `capabilities` attribute of `ServiceSpecification` objects used for the creation of virtual sensors contains the following entries:

- Key »EntityType«

  This entry specifies the type of the entity context data are going to be requested for.

- Key »EntityId«

  By means of this entry a client indicates the identifier of the entity instance its request is going to refer to.

- Key »Mode«

  This entry is used to specify if a synchronous or an asynchronous virtual sensor is needed.

- Key »VirtualSensorCapabilities«

  The value belonging to this entry is a table containing the further capabilities the virtual sensor is required to possess. The columns of the table are:

  - ContextElementName: In this column the name of the desired context element is provided.
  - Formats: This column may provide information about the required formats of the context element.
  - Reliability: In this column the minimal reliability of the context data supplied by the virtual sensor can be specified.
  - ValidityPeriod: This column indicates whether the virtual sensor is to be capable of providing information about the gathered context data's expected period of validity.
  - Prediction: By means of this entry the client indicates whether it requires context data referring to future points in time.
  - ContextElements: This column may contain the individual `ContextElement` objects the virtual sensor is to be able to determine. It is only relevant in connection with asynchronous requests.

  Again each row of the table specifies a context element that is to be provided by the virtual sensor along with the conditions it has to fulfill.

5.2.5.4 Representation of virtual sensors' results

Since virtual sensors support requests referring to both context elements and entire contexts, additional structures for events, results, and listeners referring to `Context` objects are required. These structures are depicted in Figure 42.

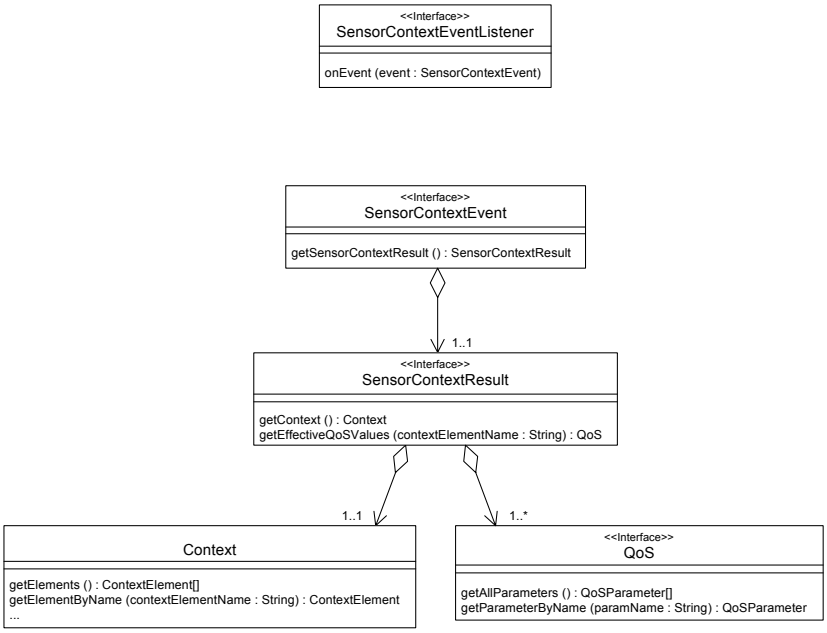

Figure 42: Virtual sensors' results

In addition to result structures referring to context elements which are made use of by virtual sensors as well, the interfaces `SensorContextEvent`, `SensorContextResult`, and `SensorContextEventListener` employed in conjunction with contexts have been defined. Analogous to the results of sensor adaptors events related to contexts are represented by `SensorContextEvent` objects. Each of these objects aggregates a `SensorContextResult` instance. In a `SensorContextResult` object a `Context` instance is contained which represents the context that has been determined. In addition to this, the effectively measured values for the stipulated quality of service parameters can be obtained from the `SensorContextResult` object. Since these parameters refer to context elements instead of contexts as a whole, more than one `QoS` object can be aggregated in a `SensorContextResult` object. These quality of service parameters are accessed by means of the `getEffectiveQoSValues()` method which is passed the name of the context element the parameters are to be returned for.

5.2.5.5 Implementation of virtual sensors

The object model for virtual sensors mainly specifies interfaces of virtual sensors and of additional structures required in connection with them. It leaves the details of how an implementation of corresponding classes is carried out up to the programmer of an information logistic

application. With regard to the information logistics framework this opportunity to use various implementations is a desired feature as it allows for an adaptation of applications to variable requirements and domains and as a result promotes the flexibility of information logistic applications. Nevertheless, in this section we propose a model for the implementation of virtual sensors as well. This model shows an implementation alternative which takes into account the potential complexity virtual sensors may possess and aims at serving as a guideline for the implementation of virtual sensors.

The proposed implementation first of all contains two classes each of which implements one of the interfaces defined for synchronous and asynchronous virtual sensors, respectively. These classes, `ActiveVirtualSensorImpl` and `PassiveVirtualSensorImpl`, both inherit from an abstract `VirtualSensorImpl` class providing necessary structures to its subclasses. The class `VirtualSensorImpl` apart from the standard constructor also has a constructor which is passed the condition part and the action part of Context SRML rules relevant for a particular request. The data type of this parameter is a vector of `org.w3c.dom.Node` objects [HoHe03], i.e. a vector containing the condition and action parts of rules.

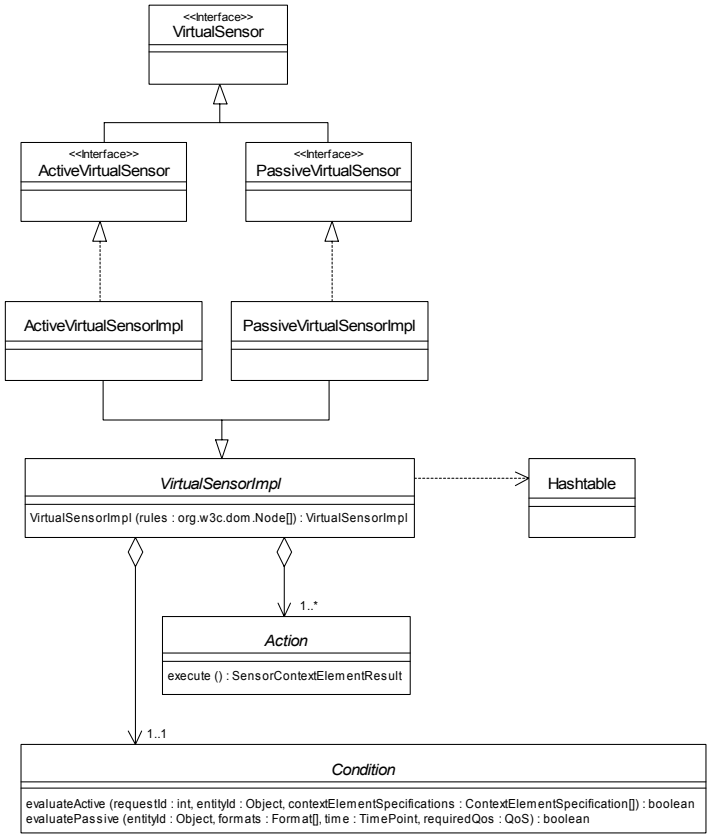

Figure 43: Basic implementation of virtual sensors

The basic idea the implementation of virtual sensors is based upon is a design in accordance with the Rule Object Pattern [Arsa01]. As can be seen in Figure 43, a `VirtualSensorImpl` object aggregates a `Condition` object and one or more `Action` objects. `Condition` objects are responsible for the gathering of context data from sensor adaptors and for the analysis and evaluation of these data. This is carried out on the basis of Context SRML rules' condition parts – if suitable rules exist – and the request parameters passed to the `Condition` objects. These parameters are described shortly in greater detail. After the context data have been gathered and examined by the `Condition` objects, `Action` objects undertake the tasks of pre-aggregating these data and creating appropriate return values. This may again involve the consideration of relevant Context SRML rules' action parts. The exchange of data between the `Condition` and `Action` objects is carried out by means of a `Hashtable` object created by an instance of a `VirtualSensorImpl` subclass and passed by it to the appropriate instances of the `Condition` and `Action` classes. In this `Hashtable` object the results determined by the conditions are stored. After the conditions have finished their tasks, these results are processed by the actions which can access and modify them.

The class `Condition` again is an abstract class. It possesses two methods which induce the gathering and evaluation of context data, one of these methods being used when the virtual sensor is queried synchronously, the other one employed for asynchronous requests. Both methods are each passed through the parameters the virtual sensor itself has been passed when being queried. Due to the existence of specialized subclasses of the `Condition` class neither the name of the context element that is to be evaluated nor a specification concerning a context as a whole are needed as parameters to these methods. Both methods return a boolean value indicating whether the condition is fulfilled.

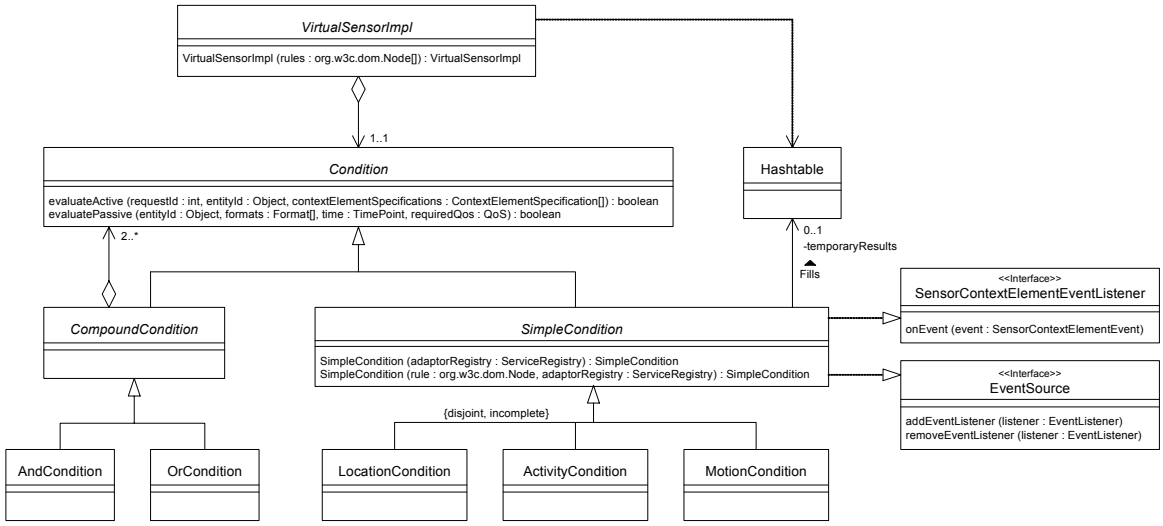

Figure 44: Condition part of the virtual sensor implementation

The `Condition` class along with its subclasses is depicted in Figure 44. Both direct subclasses of the `Condition` class are also abstract. `SimpleCondition` objects each represent a singular condition, whereas a `CompoundCondition` object is composed of at least

two other conditions. A SimpleCondition object is responsible for the gathering of context data and for checking up on conditions with reference to one single context element. For this purpose the class SimpleCondition possesses two additional constructors. Both contain the ServiceRegistry object managing sensor adaptors as a parameter. Since SimpleCondition objects independently query sensor adaptors and evaluate the results received from them, they need to be passed the ServiceRegistry object in order to be able to discover appropriate adaptors which can be queried for the required context data. In addition, one of the two constructors contains a further parameter representing the condition parts of relevant Context SRML rules that have to be taken into consideration by the SimpleCondition object. The concrete subclasses of the SimpleCondition class have been constructed on the basis of the context elements the conditions that are to be checked refer to. Accordingly, the subclasses LocationCondition, MotionCondition, ReachabilityCondition, etc. exist, three of which are exemplarily shown in the figure.

In order to be able to make use of SimpleCondition objects in conjunction with asynchronous requests the SimpleCondition class implements the interfaces EventSource and SensorContextElementEventListener, enabling its instances to both receive events from sensor adaptors and to fire events to clients. If context data which are relevant for further processing are gained during the evaluation of a condition, they are stored in the Hashtable object associated with the SimpleCondition class. On the one hand this occurs for context data which have been received as results of requests made to sensor adaptors on the basis of the request parameters passed to a virtual sensor alone. On the other hand a resultHash element in an Context SRML rule also induces context data to be stored in the Hashtable object if they fulfill the conditions specified in the rule.

The concrete subclasses of the CompoundCondition class represent boolean operators connecting the individual conditions contained in a CompoundCondition object. The AndCondition class is employed for asynchronous requests referring to more than one context element instance, while instances of the OrCondition class are made use of in conjunction with synchronous requests. Please note that in order to obtain exhaustive information concerning the context elements of an entity all conditions interconnected by a disjunction must be evaluated, even if one or more other conditions the respective OrCondition object is composed of have already returned a positive confirmation.

Condition objects are created by objects of the VirtualSensorImpl subclasses. When being passed rule files, the VirtualSensorImpl object has to create a SimpleCondition object for each condition specified in these files and combine them, if need be, in a suitable CompoundCondition object. Additionally, based on the request parameters passed to it by the client the VirtualSensorImpl object is also responsible for the creation of appropriate Condition objects which represent the client's demands.

When the examination of the conditions is finished, the actions a VirtualSensorImpl object aggregates are carried out. The class Action is an abstract class as well. It possesses an execute() method by means of which the performing of an action is induced. The result of this method is a SensorContextElementResult object. The actions a VirtualSensorImpl object aggregates are shown in Figure 45.

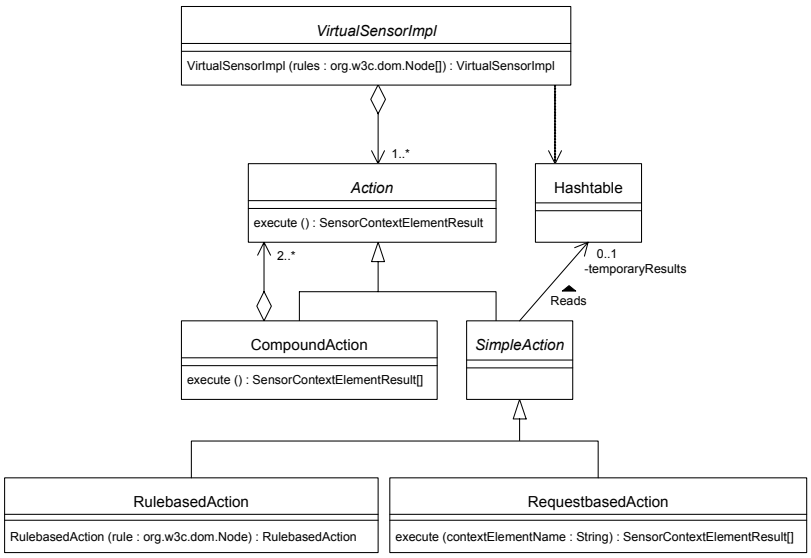

Figure 45: Action part of the virtual sensor implementation

Analogous to the `Condition` class the `Action` class possesses two direct subclasses, `SimpleAction` and `CompoundAction`, the former representing atomic actions, while `CompoundAction` objects consist of at least two `SimpleAction` objects. `SimpleAction` objects are responsible for the pre-aggregation of the gathered data and for the execution of Context SRML rules' action part. In order to perform this task they may access the `Hashtable` object filled by the `SimpleCondition` objects. `SimpleAction` objects furthermore construct a `SensorContextElementResult` object in which the results gained by them are stored. Subclasses of the abstract class `SimpleAction` are the classes `RulebasedAction` and `RequestbasedAction`. A `RulebasedAction` object generates results on the basis of a Context SRML rule's action part. Accordingly, it can be passed the corresponding part of the rule in its constructor. Since it does not necessarily have to operate on the `Hashtable` object created by the conditions – as, for example, the second rule we have explained in Section 5.2.4 –, the cardinality of the association between the `SimpleAction` and the `Hashtable` class is zero to one. In contrast to this, a `RequestbasedAction` object merely fits together the temporary results gained by the conditions concerning one particular context element in one or more `SensorContextElementResult` objects. In the `Hashtable` object it accesses for this purpose may be various results received from sensor adaptors with regard to this context element so that the `RequestbasedAction` object, too, must provide more than one `SensorContextElementResult` object. For this purpose the class `RequestbasedAction` overrides the `execute()` method defined in the `SimpleAction` class and can thus additionally return a vector of `SensorContextElementResult` objects as a result of this method's execution. The overridden `execute()` method is furthermore passed the name of the context element data are to be returned for as a parameter.

The `CompoundAction` class is employed to string together a number of `SimpleAction` objects each representing a singular part of the entire task that has to be performed. It is responsible for managing the individual `SimpleAction` objects and coordinates their exe-

cution to ensure a correct sequence of actions and a provision of valid results. Since the `SimpleAction` objects it aggregates may provide a vector of `SensorContextElementResult` objects as mentioned above, the `CompoundAction` class also overrides the `execute()` method to be able to return a vector of `SensorContextElementResult` objects as well representing the sum of all return values obtained by the `SimpleAction` objects a `CompoundAction` object aggregates.

Like `Condition` objects `Action` objects are created by instances of the `VirtualSensorImpl` subclasses. For each action defined in the action parts of the rules passed to it a `VirtualSensorImpl` object creates a `RulebasedAction` instance. In addition, a `RequestbasedAction` object is created for each context element the request made to the virtual sensor refers to. The creation of appropriate `CompoundCondition` objects to combine the individual `SimpleAction` instances is also carried out by the `VirtualSensorImpl` objects.

Since in the model we have introduced above `Action` objects do not have knowledge of whether the request made to the virtual sensor is a synchronous or an asynchronous one, they are not capable of creating the events the virtual sensor may have to fire to its client. Therefore, this model imposes the creation of event objects containing the appropriate type of result object on the `VirtualSensorImpl` instances.

5.3 Context Builders

At the top layer of the context gathering process there are mechanisms for the filtering and final aggregation of context data. As a rule, these mechanisms are only needed in conjunction with synchronous requests made to the Context Component. This is due to the fact that in case of asynchronous requests an event is immediately to be fired when an entity's context or context element fulfills specified conditions and no further data are determined except from those that have been explicitly requested. Context builders are responsible for the elimination of redundancies and contradictions from the data supplied by underlying layers. After this filtering has taken place, they aggregate the remaining context data into objects that are returned to the Context Component's clients. Similar to the previous sections this section first describes the tasks that have to be carried out at this layer of the context gathering process and then illustrates in an object model how these tasks are reflected in the design of the computational elements belonging to this layer.

Definition 17: Context builder

A context builder is a piece of software that filters and aggregates the context data supplied by virtual sensors. It optimizes the data provided by the Context Component by deciding upon which context data are to be returned to the Context Component's clients.

A context builder produces those context data the Context Component supplies to its clients, i.e. to other components of information logistic applications. The existence of context builders allows the lower layers of the context gathering process to concentrate on the task of context data acquisition and supply, each with a specialized focus, without having to pay attention to the assessment of the results they produce. Not until all context data that can be determined

have been made available context builders carry out the final preparation of the data. Thus, the tasks of context data acquisition and context data preparation are separated as a result of which the Context Component becomes more stable and maintainable by allowing for a modification of each task's implementation without affecting other functionalities.

5.3.1 Context data filtering

The filtering of context data supplied by virtual sensors involves both selecting from a number of alternative or even contradictory pieces of information about an entity's context as well as eliminating duplicate pieces of contextual information. In order to carry out this task a context builder is required to possess knowledge concerning the permissible combinations of context elements' attribute values. If, for example, an entity has been detected to be both at the railway station and in a train, no filtering is required as the entity may perfectly be at both locations at the same time. If, however, the data a context builder is provided with indicate that an entity is at the railway station and in the office, these data contradict themselves and require filtering. In the same manner a businessperson may be bathing during her leisure time, but if this activity has been detected while the person is working, the data indicating it are very likely to originate from a malfunction of a sensor adaptor or a virtual sensor and need to be eliminated. Moreover, the permissible context elements' attribute values and combinations of them may depend on the application domain. Consider, for instance, the simultaneous execution of the activities »sleeping« and »flying« we have mentioned in Section 4.2.2. In order to obtain information about permissible or inadmissible combinations of context elements' attribute values context builders thus make use of the context constraints introduced in Chapter 4.

Apart from information concerning permissible combinations of context elements' attributes and their values the measured values for the quality of service parameters provided by the underlying layers may also serve as a means to assess and filter context data. In particular the reliability of the supplied data may in the case of contradictory pieces of information be referred to in order to select the context data which are most likely to be true. In contrast to this, an elimination of duplicate pieces of contextual information in principal is a less complex task requiring a comparison of context elements for equality. Yet, since some context elements possess a nested structure and may additionally be characterized by `Attribute` objects, this comparison has to be carried out for all attribute values a context element consists of. If, for example, information referring to different `Attribute` objects is available for the same motion or emotional condition, the data concerning all different `Attribute` objects have to be preserved and taken over into the aggregation process.

As can be seen from the examples given above, the process of context data filtering may be a rather complex one which may require plenty of additional information to be available to context builders. Since in addition to this the assessments that have to be made depend on the application domain and on the specific conditions prevailing in this domain, universally applicable filtering methods cannot be provided. The design of context builders rather needs to be open to various policies for context data filtering, allowing for different implementations of how and what is to be filtered, depending on the particular application area.

5.3.2 Context data aggregation

The final step carried out within the scope of the context gathering process is the aggregation of context data. During the execution of this task those context data that have passed the context filtering process are merged in result objects that are ready to be returned to the Context Component's clients. The aggregation of context data on the one hand may involve the generation of results referring to individual context elements which is done based on the different data available for a context element. On the other hand, at this stage of the context gathering process objects representing entire contexts may also be created. For this purpose the already filtered data concerning the individual context elements the context is composed of have to be aggregated in a `Context` object. Whether a context element or an entire context is to be generated in the course of the context data aggregation is dependent on the request made to the Context Component. Thus, context data aggregation involves both the aggregation of several context elements in order to generate a `Context` object as well as the aggregation of attributes and values a single context element is composed of.

5.3.3 An object model for context builder integration

This section presents an object model for context builders. In this model the remaining structures involved in the entire context gathering process along with the relationships between them are described. In consideration of the layered structure the building blocks for context gathering are arranged in the elements of the object model explained below represent the highest layer, being superordinate to that of virtual sensors.

Context builders as the third building block for context gathering are provided with the results gained by virtual sensors, filter and aggregate the data contained in these results, and supply information concerning context elements or entire contexts which are returned to the Context Component's clients. While carrying out its tasks, a context builder may have to adhere to policies specifying conditions regarding the filtering of context data. Although context builders operate on the data provided by virtual sensors, they do not interact with them directly. Instead, so-called context gatherers are defined in the object model for context builder integration. Context gatherers serve as an intermediate layer by means of which a decoupling of context data acquisition performed on lower tiers and the preparation and return of these data carried out by context builders is achieved. As a result, context builders do not need to know the interface of virtual sensors and do not have to attend to obtaining data from them, but may rather focus on their actual tasks.

As can be seen in Figure 46, clients of virtual sensors are `ContextGatherer` objects. The class `ContextGatherer` possesses the same methods as the Context Component itself; in brief, these methods allow a client to query the entire context or a specific context element of an entity both synchronously and asynchronously. Since the parameters passed to these methods have already been explained in the previous sections and since the interface of the Context Component is dealt with in detail in Section 6.2.1, we refrain from describing these parameters any further in this section. As mentioned above, the `ContextGatherer` class merely serves the purpose of separating the acquisition of context data from the filtering and aggregation of these data and thereby reduces the complexity of the individual classes involved in the context gathering process. Since it interacts with virtual sensors, a context

gatherer may receive events fired by them. Therefore, the class `ContextGatherer` implements the interfaces `SensorContextEventListener` and `SensorContextElementEventListener`.

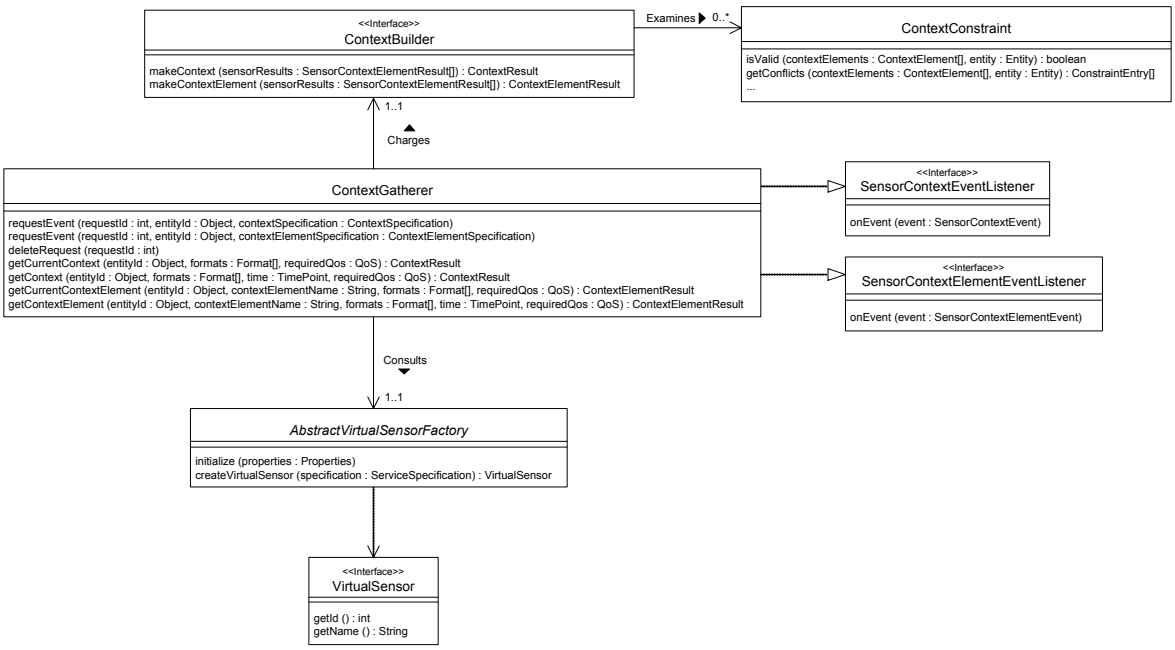

Figure 46: Context builder and context gatherer

Each `ContextGatherer` object is associated with a `ContextBuilder` object. The interface `ContextBuilder` has to be implemented by classes carrying out the filtering and aggregation of context data as explained above. For this purpose this interface defines two methods both of which are passed the `SensorContextElementResult` objects returned by the virtual sensors. If all of these objects contain a `ContextElement` object referring to the same context element, i.e. if the request made by the Context Component's client required the determination of a single context element, the method `makeContextElement()` is called and returns a single aggregated and filtered `ContextElementResult` object. If, in contrast, the client's request referred to an entity's context as a whole, the `ContextElement` objects contained in the results passed to a context builder contain information related to different context elements. In this case the `makeContext()` method of the `ContextBuilder` interface is responsible for creating a `ContextResult` object on the basis of these data. As mentioned above, classes implementing the `ContextBuilder` interface need to have additional information concerning allowed combinations of context elements' attribute values at their disposal. For this purpose the `ContextBuilder` interface is associated with the `ContextConstraint` class we have already described.

The object model explained above does not explicitly specify how all potential criteria for context data filtering are to be encoded, interpreted, and brought into action. It rather provides a general interface allowing for different implementations of policies concerning the execu-

tion of filtering processes. As we have argued in connection with virtual sensors as well, this is due to the fact that application-specific implementations are to be allowed to increase the flexibility and universality of information logistic applications. We have already mentioned in conjunction with the context model that Context SRML may be employed for the definition of filtering policies. By means of this language a `ContextBuilder` implementation may define how a filtering of context data is to take place. Since we have already illustrated examples of Context SRML and a model of how these rules are made use of in conjunction with virtual sensors, we refrain from providing an implementation for context builders at this point.

5.4 Summary and Assessment

In this chapter we have presented techniques for an efficient gathering of context data within the Context Component of information logistic applications. This section concludes our explanations regarding this topic by resuming the main characteristics of our approach and comparing it with the existing approaches described in Chapter 3. In doing so, we point out the differences between our solution and the related ones, thereby reasoning how our approach contributes to overcoming existing restrictions. The overall model for the structures involved in the context gathering process is depicted in Figure 48 on page 154. For reasons of clarity the proposed implementation of virtual sensors is not shown in this figure.

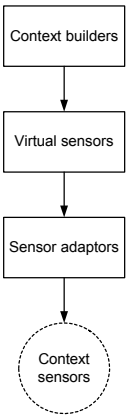

Figure 47: Hierarchy of context gathering components

Our context gathering techniques are based on a three-tiered hierarchy of specialized subcomponents as depicted in Figure 47. We have explained the individual layers of this hierarchy from the bottom up. First, numerous hardware and software systems are available which provide data concerning the entire or a part of an entity's context. We refer to these systems as context sensors. In the figure they are represented by a circle with a dashed line accentuating that context sensors are external services which are not part of the Context Component.

The main tasks on the lowest layer of the context gathering process, covering the so-called sensor adaptors, are to integrate these external sensors into the Context Component and to make their data available to higher layers in a common format for further processing. In order

to carry out these tasks sensor adaptors require an in-depth understanding of the context sensors that are to be employed as well as of the data provided by them, their interfaces, mode of operation, and so on. Therefore, we have first provided an overview of existing context sensors and context gathering technologies, subdivided according to the context elements the Context Component deals with. After that we have identified criteria which serve to assess context sensors and their suitability for use in information logistic applications. Apart from referring to the criteria for the evaluation of location sensors provided by Hightower and Borriello we have identified further criteria which facilitate the assessment of context sensors from the Context Component's point of view. Since the data supplied by context sensors usually differ strongly in formats, sensor adaptors need to convert these data into a uniform representation. This is achieved by transforming sensor data into objects according to our context model. Therefore, the steps required in connection with sensor data transformation have been identified and described, providing a guideline for the execution of this task. After that an object model containing the relevant structures concerning sensor adaptors along with the relationships between them has been presented.

The usage of sensor adaptors as a means to integrate heterogeneous context sensors and to make available the data provided by them entails the following main benefits:

- The implementation specifics of context sensors, their interfaces and data formats are hidden from other parts of the Context Component. Context sensors may therefore change with minimal impact as the interaction with them is limited to sensor adaptors.
- Sensor adaptors provide higher layers of the context gathering process with data according to a uniform and known format, thereby suiting the needs of other parts of the Context Component.
- The concept of sensor adaptors makes the Context Component sensor-independent as any context sensor can be integrated into it by means of an adaptor, and the transformations carried out by sensor adaptors ensure a uniform representation of the gathered data.
- The uniform interface of sensor adaptors provides easy access to context data and allows for a reuse, combination, and customization of sensor adaptors.

On the layer above sensor adaptors virtual sensors represent another building block for context gathering. On the basis of the context data provided by sensor adaptors virtual sensors pre-aggregate and combine these data and derive further information concerning an entity's context from them. In the section dealing with virtual sensors we have first explained these tasks in greater detail. Among them the derivation of context data plays the most prominent role as it is extremely application-specific and may bring about a particularly high complexity. As a means to determine mechanisms for context data combination and derivation we have presented Context SRML, a rule language based on the Simple Rule Markup Language (SRML). This language can be used to specify rules defining how context data are to be combined and derived. After briefly explaining Context SRML in general we have illustrated its scope and expressiveness by means of two examples. Finally, an object model for the integration of virtual sensors into the Context Component has been presented. This model consists of the relevant computational elements required to represent virtual sensors as well as to create them. It is accompanied by a model describing a proposed implementation of virtual sen-

sors. This implementation reflects the structure of Context SRML and, based on the Rule Object Pattern, provides a flexible means of representing the possibly complex structures required in connection with rule-based context data combination and derivation.

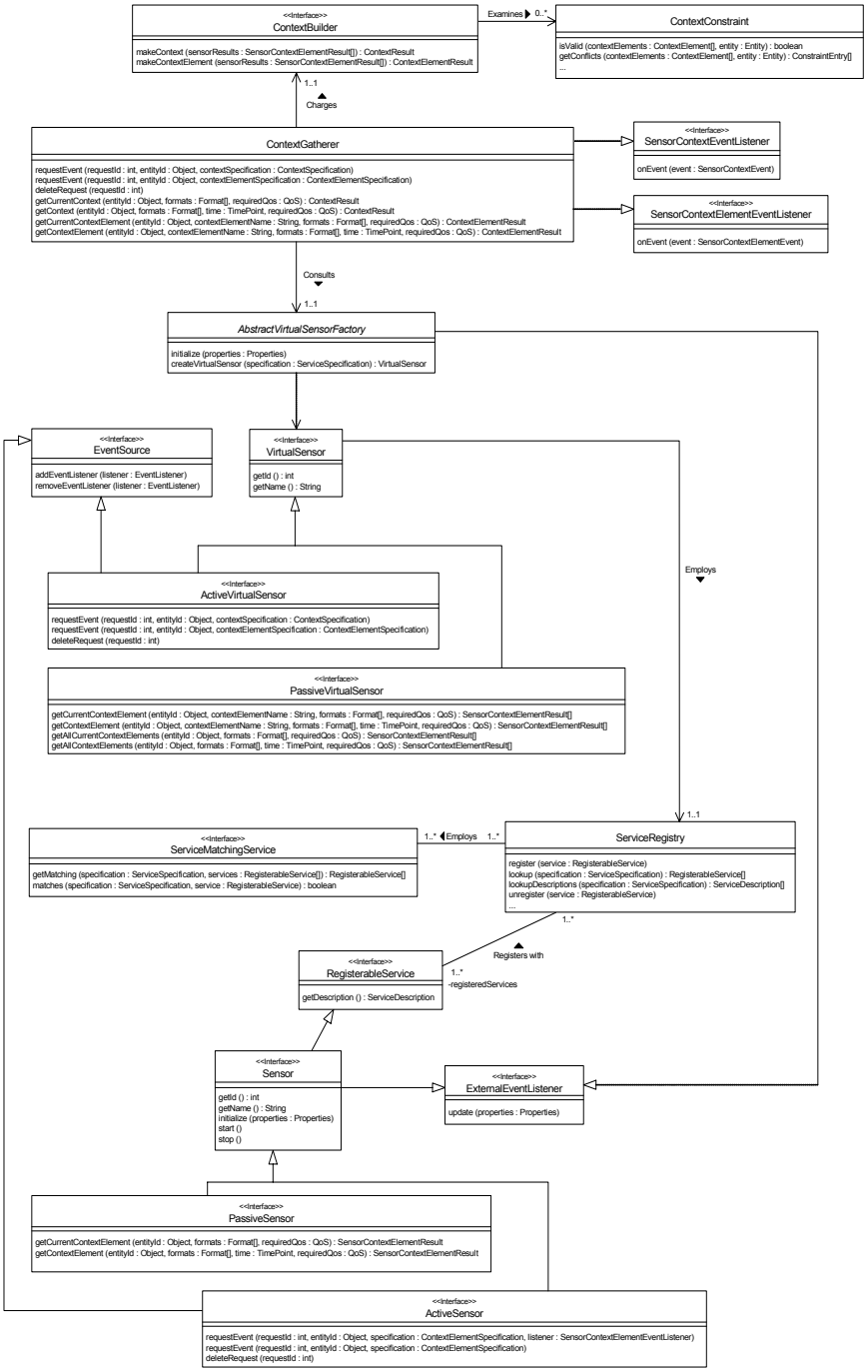

Figure 48: Overall model for context gathering structures

The main advantages resulting from the separation of the tasks carried out by virtual sensors from the acquisition of context data done by sensor adaptors and the context data filtering and final aggregation carried out on the next higher layer are:

- Higher layers of the Context Component can obtain context data from virtual sensors without having to possess any knowledge of the existence of sensor adaptors. As a result, a separation of concerns is achieved which reduces the complexity of the individual subcomponents involved in the context gathering process.
- Virtual sensors' clients have direct access to pre-aggregated and combined context data without having to collect and prepare individual pieces of information from sensor adaptors themselves. Therefore, the concept of virtual sensors contributes to an increased maintainability of the Context Component by ensuring a loose coupling of its subcomponents.
- Rules for context data combination and derivation specified by means of Context SRML provide a powerful and flexible instrument for the definition of application-specific procedures. These rules are interpreted and executed at run-time which furthermore allows for a rapid change of the mechanisms employed in response to varying conditions in the application domain and a definition of rules by those people that possess the most detailed knowledge of the application area. Since even ordinary users may be given the opportunity to define their own rules, the acceptance of an information logistic application is increased by making the context gathering process more transparent.

Finally, we have explained the third building block for context gathering, the so-called context builders. Context builders are responsible for the filtering and final aggregation of the context data supplied by virtual sensors. They remove redundant data, resolve existing contradictions, and generate the context data that are returned to the Context Component's clients. After explaining the tasks of context builders in greater detail the respective section has also contained an object model for this building block. In conjunction with the explanations concerning this model context gatherers representing an intermediate layer for the interaction with virtual sensors and the forwarding of the results obtained from them have been introduced. We have furthermore proposed the usage of Context SRML for the definition of context data filtering mechanisms.

Using context builders as building blocks for the final preparation of context data results in the following benefits:

- The data the Context Component supplies to its clients are optimized to the greatest possible extent as context builders filter out redundant or contradictory data determined by sensor adaptors and virtual sensors.
- Various policies concerning the filtering of context data can be implemented which allows for a further customization of the context gathering process.
- Context data filtering and aggregation are carried out at a centralized point of the context gathering process. As a result, lower layers do not have to concern themselves with assessing the data they supply which again is beneficial to the stability and maintainability of the Context Component.

Unlike other approaches such as Affective Computing or Sentient Computing, for instance, the context gathering techniques provided by us allow to integrate any possible context sensor into the Context Component. We have identified an extensive set of criteria which serve to assess context sensors and to facilitate their integration. Both these criteria and the procedure of sensor data transformation we have described are unique to our solution and are not provided by other existing approaches. Only the Context Toolkit covers a design process for context-aware applications, but it does not address the process of transforming sensor data into a uniform representation. Since sensor adaptors are managed in a service registry, our approach comprises a dynamic discovery mechanism which is not available in many other systems such as those proposed by the TEA or the Sentient Computing project, for example. We furthermore allow to associate gathered context data with an extensible set of quality characteristics. Apart from Henricksen et al.'s approach which aims at a similar goal other proposed systems ignore the aspect of context data quality or support only a limited number of quality attributes. Another distinguishing feature of the mechanisms for context gathering we have presented is the comprehensive consideration of context data augmentation. Our solution not only supports complex aggregation, combination, derivation, and filtering processes, but also includes means of declaratively defining rules that specify how these tasks are to be carried out. These rules can be created, deleted, or modified at run-time and thus allow for an application-specific adaptation of the Context Component. Although many existing systems cover context data augmentation to some extent, none of them provides a similarly comprehensive solution. Due to the hierarchical arrangement of the components involved in the context gathering process and their loose coupling, the encapsulation of sensor-specific tasks, the usage of Context SRML, and the modular design our context gathering techniques are universally applicable, extensible, and – in contrast to approaches such as the Coordinated Adaptation Platform, SOLAR, and others – ensure a separation of concerns between the gathering of context and its use.

This summary shows that the requirements onto the context gathering process identified in Section 2.4.2 are fully met by the solution we have developed. However, we do not leave unmentioned that the flexibility and power of the context gathering mechanisms described in this thesis come at the expense of the following drawbacks:

- The three-tiered hierarchy the context gathering process is composed of may result in an increased amount of computations and may impose a high amount of complexity, in particular on small-scale applications.
- A separate transformation of sensor data into context elements is required.
- Additional services and data sources for the transformation of sensor data are needed and may have to be created and managed manually.
- Context SRML as a language for the definition of context data combination and derivation and possibly context data filtering mechanisms as well is not easily understood by ordinary users. The expressive power gained by employing this language therefore may have to be paid for by the need to create a separate interface for rule definition suitable for inexpert users.

6 Architecture of the Context Component

The concepts, models, and techniques for the representation and gathering of context are incorporated in the architecture of the Context Component. This architecture serves as a guideline covering all tasks required to put context awareness in information logistic applications into practice. In this chapter the reference architecture for the Context Component we have developed in consideration of the requirements specified in Section 2.4.3 is presented.

Apart from an overall framework which determines the universal application structure and mechanisms required by all components separate reference architectures for individual components of information logistic applications exist. The reference architecture for the Context Component accordingly has to adapt itself to the information logistics framework by complying with the mechanisms and structures it defines and by ensuring interoperability with the existing components. In the first section of this chapter we therefore provide an overview of the information logistics framework and show how the Context Component fits into it.

Designing a reference architecture for the Context Component requires to take several aspects into consideration which include:

- Methods and concepts for the design, implementation, operation, and maintenance of the Context Component
- The integration of existing systems and services, e.g. context sensors and external data sources for data transformation
- Embedding the Context Component both into information logistic applications and into existing environments.

Due to the complexity the Context Component possesses in consideration of all these aspects it is advisable to distinguish between several views of the component. In doing so, each view describes the Context Component with regard to particular concerns and thus serves as a means of handling the respective aspects of the problem part it addresses. The concept of views as a principle of describing architectures is made use of in most architecting techniques (e.g. [HoNo99], [EmHi96]). Examples of proposed sets of views are Kruchten's »4+1« view model [Kruc95], the ISO Reference Model for Open Distributed Processing (RM-ODP) [ISO96], or the United States Department of Defense's C4ISR Architecture Framework [DoD97]. As can be seen from these examples, a variety of views exists. Architects are confronted with this hardly manageable number of views to choose from, and – since these approaches possess different degrees of precision and yet have similar objectives – it becomes extremely difficult for them to select the most suitable combination of views to make use of in a given situation.

An approach to overcoming this problematic heterogeneity is IEEE's 1471-2000 Recommended Practice for Architectural Description for Software-Intensive Systems [IEEE00]. This standard aims at providing a conceptual framework and vocabulary by means of which common terms and concepts are to be established on the basis of customary practices and consensus. The IEEE 1471 standard applies to architectural descriptions (ADs), sets of work results describing a system's architecture. The specifications made in the standard are notation-independent; they define the required minimal content of an AD, but not its format. An AD consists of one or more views which again may be composed of one or more architectural

models. The key contribution of IEEE 1471 is the recognition of viewpoints as the generic, reusable part of ADs. Each view belongs to exactly one viewpoint which can be regarded as a pattern for the construction of views. In contrast to other approaches IEEE 1471 does not provide a fixed set of viewpoints. Instead, it specifies a set of requirements every viewpoint has to fulfill and leaves it up to the architect to identify and define those viewpoints that are suitable for addressing the requirements made onto a particular application. Although IEEE 1471 also provides a definition of the term architecture, achieving a universal understanding of what a software architecture is is not its primary goal. It rather intends to specify how an architecture is to be described, irrespective of how the parties involved interpret this term.

With regard to the reference architecture for the Context Component we have chosen to conform to the IEEE 1471 standard and as a result do not add another definition of the term software architecture and another set of recommended views to the variety of propositions that already exist. Instead, we refer the reader to the existing definitions of the term software architecture and to their common elements we have illustrated in Section 2.3.3. As far as the views of the Context Component are concerned, we identify and describe – in compliance with IEEE 1471 – a set of appropriate viewpoints in consideration of the requirements made onto the component. These viewpoints are dealt with in the second section of this chapter.

6.1 Integration of the Context Component into the Information Logistics Framework

The goal of the information logistics framework has been to provide a unit construction system that allows for a quick and simplified implementation of information logistic applications. For this purpose the framework defines both a set of standard components and services along with a unified method of integrating further components as well as common mechanisms concerning the communication among the components, the management of the components' life cycle, quality control, and so on. In order to facilitate the understanding of the Context Component's architecture and to provide a closer insight into its boundary conditions this section outlines the information logistics framework and shows how the Context Component fits into it.

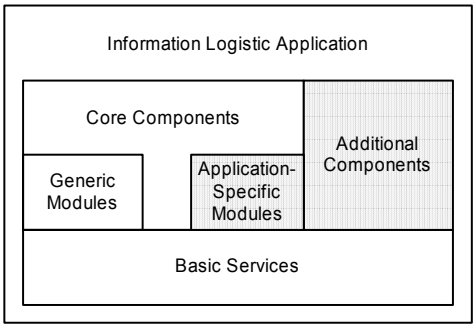

Figure 49: Elements of information logistic applications

An information logistic application contains both generic parts and application-specific logic. Its overall functionality is achieved by components, modules encapsulated by these components, and a number of basic services accessible to all components and modules. As depicted in Figure 49, a distinction is made between core components and additional components. Core components constitute the principal elements of every information logistic application. They provide a well-defined and cohesive functionality which is independent of any particular application domain. Core components encapsulate a number of modules each of which implements a fixed part of a component's overall functionality. Application-independent modules are called generic modules, while modules providing a functionality particular to a specific application or application type are referred to as application-specific modules. In contrast to core components additional components are only required for particular applications or types of applications. Therefore, neither additional components nor application-specific modules are covered by the information logistics framework.

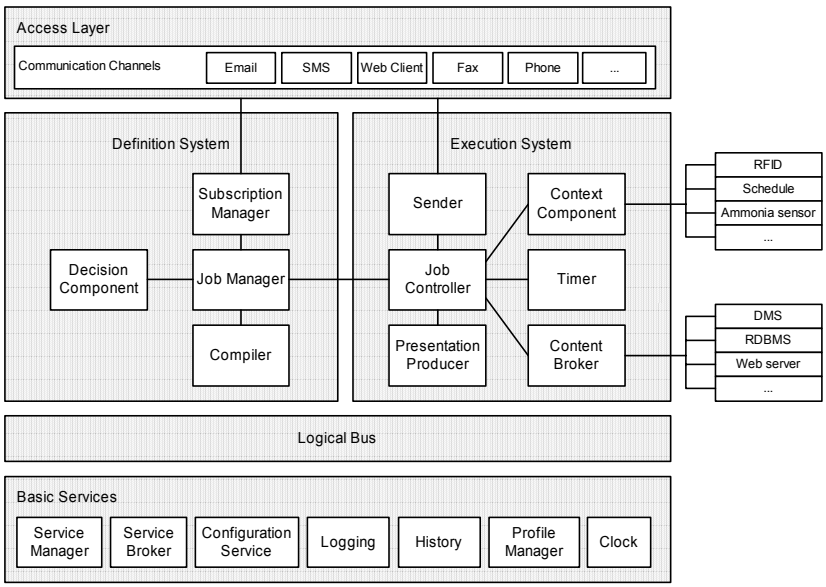

Figure 50: Subsystems and components of the information logistics framework

In summary, the information logistics framework includes core components, generic modules, basic services, and a communication infrastructure. The framework is subdivided into four logical subsystems:

- An access layer which enables the communication between information logistic applications and their users
- A definition system deciding upon what information is to be provided to users and how information supply is to be carried out. This part of the framework is concerned with users' information demands and determines the processes required to implement an information supply that meets these demands
- An execution system responsible for executing the processes determined by the definition system

- Basic services which are required by and accessible to all other components of the information logistics framework.

Each of the components the framework defines belongs to one of these subsystems. Figure 50 shows the subsystems of the information logistics framework along with the components belonging to each of them, including the newly added Context Component.

The components on the access layer each provide mechanisms for communicating via a particular channel. They encapsulate technical details such as the interfaces and data formats a channel possesses and provide a uniform interface to other components by means of which data can be sent and received. The access layer's components carry out the communication between an information logistic application and its users. The purpose of these components is to enable users to access profile data and define information demands and to enable the information logistic application to supply users with information.

The definition system of the information logistics framework covers the following components:

- Subscription Manager

  This component is passed users' information demands, so-called subscriptions, by the components on the access layer. It assigns identifiers to subscriptions, persistently stores them, and manages the access to the subscription repository. After the Subscription Manager has processed the received subscriptions it forwards them to the Job Manager.

- Job Manager

  The Job Manager receives subscriptions from the Subscription Manager. It charges the Job Compiler with the translation of subscriptions into so-called jobs, assigns identifiers to jobs, and manages the relation between subscriptions and jobs. The Job Manager represents the connecting link between the definition system and the execution system by forwarding the generated jobs to the Job Controller.

- Job Compiler

  The Job Compiler is responsible for the translation of subscriptions into jobs. A job represents the information demand of a user in an executable format, whereas a subscription contains a human-readable representation of this demand.

- Decision Component

  This component is responsible for the overall optimization of information supply by coordinating the individual optimizations carried out within the individual components. It serves as a final instance for determining the parameters concerning the execution of jobs.

The following components belong to the execution system of the information logistics framework:

- Job Controller

  The Job Controller is passed jobs from the Job Manager and is responsible for their correct and optimized execution. For this purpose it calls upon the other components of the execu-

tion system to perform their specific tasks with respect to the parameters defined in a job. Events fired by these components are received by the Job Controller and are forwarded to the appropriate jobs. The Job Controller consults the Decision Component if processes and optimizations have to be coordinated and is in charge of the enforcement of its decisions.

- Timer

  This component manages all aspects related to the dimension of time and is responsible for the optimization of information supply with regard to this specific dimension. It notifies the Job Controller whenever particular points in time, time intervals, cycles, etc. related to users' information demands occur.

- Content Broker

  The Content Broker provides the Job Controller with a uniform interface to the application-specific content services of an information logistic application. A content service represents a data source containing information that is to be supplied to users. The Content Broker assesses, selects, and retrieves the pieces of information users are to be provided with and therefore performs optimizations with regard to the dimension of content.

- Context Component

  The Context Component is concerned with the representation, gathering, management, and supply of entities' contexts. It provides models for the description of context, determines the contexts of entities, and makes these contexts available to the Job Controller. As a result, an optimization of information supply with respect to the dimension of context is achieved.

- Presentation Producer

  This component transforms the information made available by the Content Broker into different formats and media types. Its main responsibilities are the assessment of different transformations on the basis of criteria such as time, cost, information loss, and others and the execution of selected transformations to generate documents which are to be supplied to users. The Presentation Producer is dealt with in detail in [Wojc03].

- Sender

  The Sender interacts with the components on the access layer in order to supply users with information. It is passed several parameters by the Job Controller which affect how information supply is carried out such as the communication channel that is to be used, the maximum allowed time and cost for information delivery, specifications concerning security, etc.

The basic services defined by the framework include a Service Manager responsible for the management of the components and their life cycle, a Service Broker which provides distributed computing and availability mechanisms, a Configuration Service supplying the components with initialization data, a Profile Manager administering user profiles, a History which archives past interactions between users and the application as well as a service for the logging of status and error messages and an internal clock.

Since the information logistics framework itself is a complex architecture, this thesis can only give a brief introduction into its basic contents. More detailed information concerning the framework is given in [Sand01], [LoPf01], and [DeLo03].

Apart from defining the structure of information logistic applications the information logistics framework also contains general instructions regarding the following aspects:

- Communication protocol independence

  Apart from interfaces by means of which the functionality of a component can be accessed each component is required to define separate protocol-specific interfaces. As a result, components become independent of the communication protocol employed in an application and are not affected by protocol changes.

- Component structure

  Each component is subdivided into a functional and an administrative part. The actual functionality of a component is implemented in its so-called back-end, while the component's front-end is responsible for obtaining references to other components and services. This subdivision leads to a decoupling of components and facilitates component tests and maintenance.

- Component life cycle

  The information logistics framework defines the states a component may be in during its life cycle along with the operations that cause a transition from one state to another and the resulting states of transitions.

- Component configuration

  Each component is required to implement a specific `initialize()` method by means of which it is passed configuration data by the Configuration Service mentioned above.

- Programming language

  All components and modules belonging to the information logistics framework as well as the basic services are required to be written in the Java programming language.

Again, in the scope of this thesis these specifications can only be touched on; for further details we refer the reader to [KoPf01].

The explanations given in this section provide an insight into the information logistics framework and the components and mechanisms it defines. We have shown how the Context Component fits into the overall architecture of information logistic applications and which boundary conditions it has to comply with. Therefore, the specifications made in the framework have affected parts of the Context Component's architecture which is described in detail in the following section.

6.2 Context Component Viewpoints

In this section the reference architecture for the Context Component is presented by distinguishing between several viewpoints each of which describes a particular aspect of the component's architecture in detail. Before examining the individual viewpoints in the following subsections we first of all identify the relevant viewpoints by means of which an accurate and comprehensive description of the Context Component's architecture can be given and outline their basic contents.

The selection of viewpoints is driven by concerns. Each viewpoint addresses particular concerns that are important to the stakeholders of the Context Component. With regard to these concerns we have identified the following viewpoints used to describe the Context Component's architecture:

- External viewpoint

  The external viewpoint deals with the functional requirements made onto the Context Component and the fulfillment of these requirements with regard to the component's clients. It defines the functionality the Context Component makes available to clients and the way this functionality can be accessed.

- Logical viewpoint

  The logical viewpoint also addresses the functionality of the Context Component by decomposing the component into objects or object classes. The computational elements required within the Context Component along with their relationships to each other and to potential external elements are defined. In contrast to the external viewpoint the logical viewpoint deals with the internal structures of the Context Component.

- Dynamic viewpoint

  The dynamic viewpoint focuses on the processes executed within the Context Component along with their temporal interdependence and the flows of information that take place. It defines the behaviour of the computational elements the Context Component consists of and additionally addresses non-functional issues such as concurrency and fault-tolerance.

- Structural viewpoint

  The structural viewpoint is concerned with aspects related to the development of the Context Component such as maintainability, extensibility, etc. For this purpose a decomposition of the Context Component into subsystems is carried out. This viewpoint defines which modules and packages the component is composed of and how they are organized.

- Physical viewpoint

  The physical viewpoint defines how the Context Component's computational elements are mapped to the available hardware. It deals with aspects of distribution and of communication between the elements of the component software. Therefore, this viewpoint primarily addresses aspects of performance, scalability, fault-tolerance, and availability.

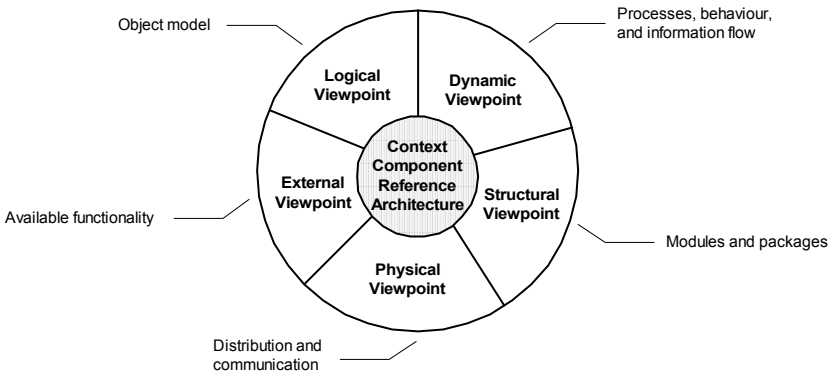

Figure 51: Context Component architecture's viewpoints

The viewpoints we have identified along with their primary concerns are illustrated in Figure 51. The following subsections each deal with one of them in detail. Since our viewpoint definition is to comply with the IEEE 1471 standard, each viewpoint specifies those characteristics that are defined by IEEE 1471 as the required part of every viewpoint, namely:

- a viewpoint name
- the stakeholders whose concerns the viewpoint addresses
- the stakeholders' concerns addressed by the viewpoint
- the viewpoint language
- the source of the viewpoint.

6.2.1 External viewpoint

The external viewpoint of the Context Component describes the component's functionality in terms of services made available to clients. It furthermore provides information regarding the way of accessing that functionality by describing the interfaces the Context Component puts at the disposal of other components. The external viewpoint therefore is addressed to the Context Component's clients – in particular the Job Controller defined in the information logistics framework –, the component developers and testers as well as integrators responsible for embedding the Context Component into information logistic applications. The component's interface is described in the form of UML class diagrams. The external viewpoint covers parts of what is known as static or application viewpoints in other approaches. Yet, this viewpoint is concerned with the externally available services of the Context Component only and does not deal with its internal structure. The latter is provided by the logical viewpoint described in Section 6.2.2. These two viewpoints have been separated from each other, because we consider a distinct treatment of the externally available services the Context Component provides advisable as the component's clients thus do not have to concern themselves with the internal details of the component described in the logical viewpoint. The characteristics of the external viewpoint are summarized in Table 3.

Viewpoint name	External
Stakeholders	Clients, developers, testers, and integrators
Concerns	- What is the component's functionality? - How is the component's functionality accessed? - What interfaces are available?
Viewpoint language	Interfaces and their parameters (UML class diagrams)
Viewpoint source	Also known as static or application viewpoint

Table 3: Characteristics of the external viewpoint

The functionality provided by the Context Component is to determine, represent, manage, and supply the entire context or individual context elements of entities. The component allows clients to both synchronously and asynchronously query contexts and context elements. An asynchronous query specifies conditions a context or a context element has to fulfill with respect to a particular entity and specified quality of service parameters. As soon as the Context Component detects that these conditions are met it fires a corresponding event. Clients may also delete a previously made asynchronous request. Synchronous queries induce the Context Component to determine an entity's context or context element with respect to specified formats and quality of service parameters. They may either refer to the current point in time or to an explicitly specified future or past point in time. When making a synchronous query, the client waits for the Context Component to return a result before proceeding.

In addition to this, the Context Component is also able to provide clients with metadata concerning the available context sensors and their capabilities. This information is required by other components of information logistic applications, in particular by the Subscription Manager, to prevent contexts or context elements which cannot be detected by the available sensors from being specified. In summary, the functionality provided by the Context Component to clients comprises

- notifying clients when an entity's context or context element matches specified conditions
- deleting a previously made request for a notification
- determining an entity's context or context element with respect to particular formats, quality of service parameters, and a specified point in time
- providing metadata about the available context sensors and their capabilities.

The Context Component's functionality of representing, gathering, managing, and supplying context is made available by means of the interface `ContextComponent`. Synchronous requests may either refer to an entire context or to a particular context element. Moreover, the client is enabled to request a context or a context element with respect to the current or to a dedicated future or past point in time. Thus, the `ContextComponent` interface makes four methods available for synchronous queries, each representing a certain combination of these criteria. The `getContext()` and `getCurrentContext()` methods induce the Context Component to determine the entire context of the entity specified by the `entityId` parame-

ter of these methods. Further parameters are the formats the context elements a context consists of are to possess and quality of service parameters that are to be fulfilled during the determination of the context. In addition, the `getContext()` method is passed a `TimePoint` object specifying the point in time the context is to be determined for. In the same manner, the synchronous determination of context elements is requested by means of the `getContextElement()` and `getCurrentContextElement()` methods. Apart from the parameters passed to the synchronous methods for context determination as well these methods also contain a `contextElementName` parameter used to specify the requested context element. Return values of the synchronous methods are `ContextResult` and `ContextElementResult` objects, respectively which are described later on.

Asynchronous queries are enabled by the `requestEvent()` methods of the `ContextComponent` interface. They are passed a `ContextSpecification` or a `ContextElementSpecification` object representing the abovementioned conditions specified by the client. The `requestEvent()` methods are furthermore passed a request identifier and an identifier of the entity the request refers to. Since the Context Component fires events when the conditions specified in an asynchronous request are fulfilled, the interface `ContextComponent` extends the interface `EventSource`. By means of this interface's methods clients have to register themselves with the Context Component before making asynchronous requests to it. Clients may delete previously made requests by means of the `deleteRequest()` method containing the request identifier as a parameter. Since the parameters of the `ContextComponent` interface's methods have already been explained in detail in conjunction with context gathering, we do not describe them once again in this section. The interface `ContextComponent` is shown in Figure 52.

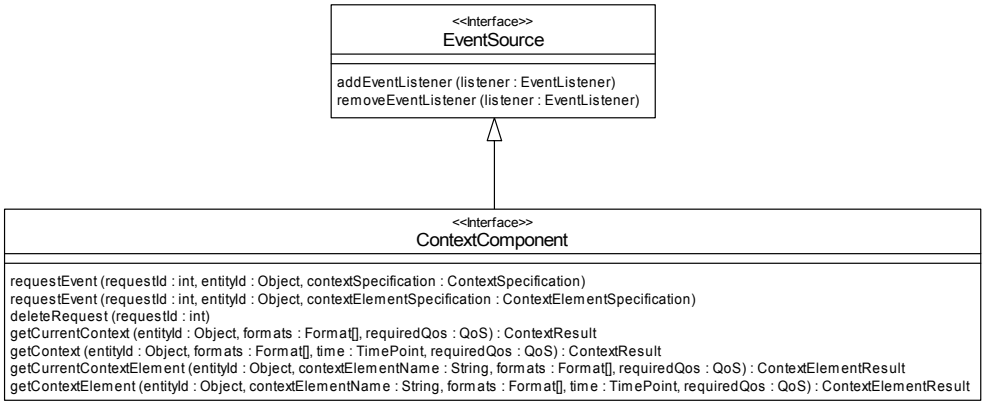

Figure 52: Interface **`ContextComponent`**

The values returned by the synchronous methods of the `ContextComponent` interface are `ContextResult` or `ContextElementResult` objects, depending on whether the corresponding request referred to a context or a context element. A `ContextElementResult` instance aggregates both a `ContextElement` object representing the determined context element and a `QoS` object containing the actual values for the quality of service parameters specified in the request. It provides methods to access these objects. Similarly, both a `Con-`

`text` object and one or more `QoS` objects, each representing the quality of service parameters measured for a particular context element the context consists of are contained in a `ContextResult` object.

The events fired by the Context Component are either `ContextElementEvent` or `ContextEvent` objects. Each of these objects aggregates a `ContextElementResult` or a `ContextResult` instance, respectively. To be able to receive events of these types clients are required to implement corresponding interfaces. For this purpose the Context Component furthermore defines the interfaces `ContextEventListener` and `ContextElementEventListener` the `onEvent()` methods of which serve an implementing class to receive events fired by the Context Component.

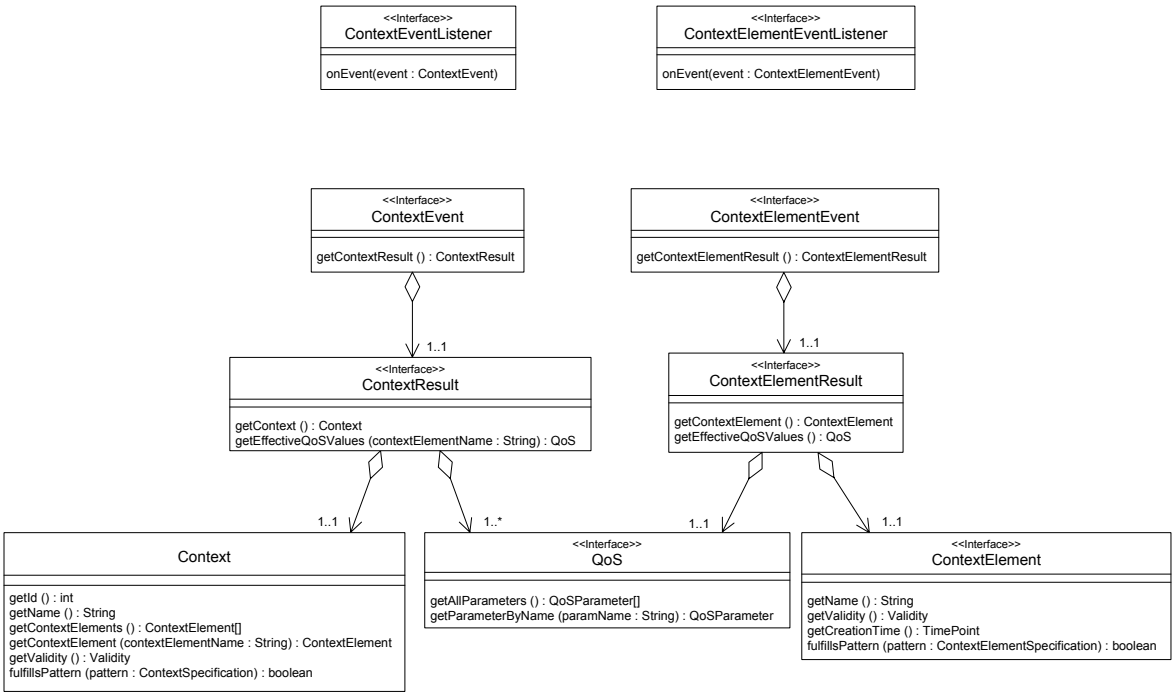

Figure 53: Context Component's results

As can be seen in Figure 53 which illustrates the results the Context Component provides its clients with, the interface `ContextElement` and the class `Context` contain further methods we have not dealt with so far. Two methods in particular, defined in both the `Context` class and the `ContextElement` interface, require more careful attention. First, the `getValidity()` method serves to enable clients receiving a `Context` or a `ContextElement` object as a result of a request to obtain information concerning the context's or context element's validity. Although the context information supplied by the Context Component initially is valid as a matter of principle, a supplied context or context element may become invalid over time. Therefore, the `getValidity()` method provides valuable information indicating whether previously received contextual information can still be made use of after a certain period of time. The return value of the `getValidity()` methods is a `Validity` object. As

illustrated in Figure 54, the type `Validity` is a general interface providing an `isValid()` method which returns true if the associated context data are valid and false if they are not. The interface `Validity` itself does not define what is considered valid context data. Rather, the model leaves it up to specific classes implementing this interface to define the conditions under which a context or a context element is considered valid.

In Figure 54 a `ValidityPeriod` class is shown which implements the `Validity` interface and represents validities depending on the dimension of time. Its attributes are a `TimeInterval` object containing the absolute period of validity of the associated context data and a `TimeSpan` object which represents the remaining time the context data are going to be valid, measured from the current point in time. By means of the `ValidityPeriod` class' methods these attributes can be accessed, and the expected point in time the context data cease to be valid at can be determined. A `Validity` object may also be passed to the Context Component as a `QoSValue` contained in the `QoS` parameter of a request which enables clients to specify conditions concerning the validity of the context or context element that is to be determined.

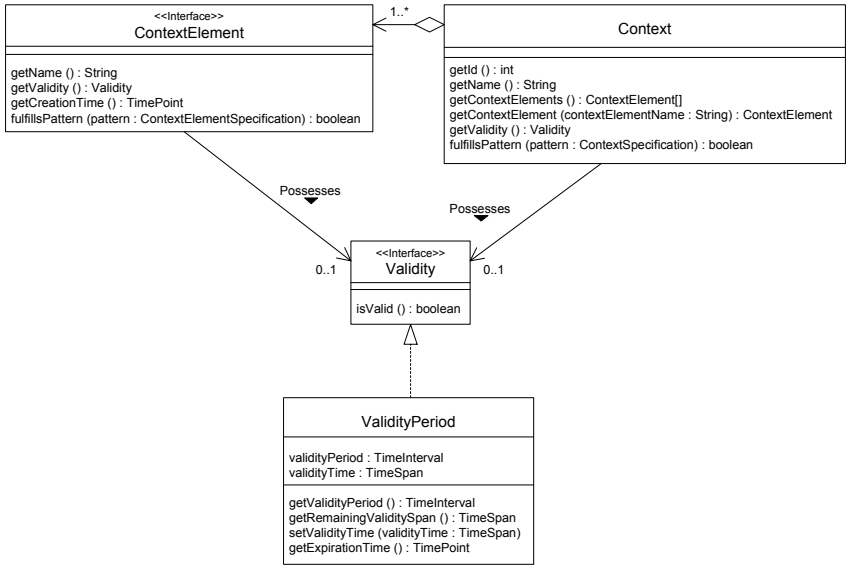

Figure 54: Representation of contexts' and context elements' validity

In addition to this, both `Context` and `ContextElement` objects possess a method called `fulfillsPattern()`. The purpose of this method is to facilitate the comparison between context data supplied by the Context Component and context data defined in information demands, i.e. in subscriptions or jobs. Upon receipt of context data from the Context Component in response to a synchronous request clients have to check whether the received data match particular conditions. These conditions are usually represented by a `ContextSpecification` or a `ContextElementSpecification` object stored in a job or a subscription. These objects define the attributes and values a context or a context element is required to possess. Instead of having to compare each of these attribute values with those of the object supplied by the Context Component themselves, clients may call the `fulfillsPattern()`

method on the received object and pass to it a `ContextSpecification` or `ContextElementSpecification` object that is required to be met. The `fulfillsPattern()` method returns true in case the object the method is called upon matches the specification passed to it. This means that all of the context elements and attributes contained in the specification have to be, either directly or indirectly, also present in the context or context element matched against it. In addition, the attribute values of the context or context element are required to conform to the corresponding operation defined in the specification or – if no operation is present – have to be equal to the context attribute values contained in it. Thus, the attributes' names and values of the context or context element the method is called upon are compared with those of the passed parameter, including all nested attributes, and besides the `fulfillsPattern()` method also determines indirect correspondence between context elements, meaning that it returns true if, for instance, an entity has been detected to be at a location that is contained in the location passed to the method.

We complete our explanations concerning the methods of the `Context` class and the `ContextElement` interface by mentioning that apart from methods to access the context elements a context consists of which have already been introduced in Chapter 4 the class `Context` furthermore provides methods by means of which a context's name and identifier can be obtained. A `ContextElement` object also possesses a name and a method to obtain it as well as a `getCreationTime()` method by means of which callers may determine the point in time the `ContextElement` object was created at.

A further functionality of the Context Component is to provide metadata concerning the available context sensors and their capabilities to clients. For this purpose the interface `ContextMetadata` has been defined. It contains a method called `lookupDescriptions()` which is passed a `ServiceSpecification` object and returns a vector of `ServiceDescription` objects belonging to sensors which match this specification. If passed a `null` value, the `lookupDescription()` method returns the descriptions of all available sensors. The `ServiceDescription` objects a client is enabled to obtain by means of this method allow it to access the names, service models, interfaces, and capabilities of the available context sensors. Thus, the client can come to know which context elements, context element attributes, and values for them are detectable in a particular application. The interface `ContextMetadata` is shown in Figure 55.

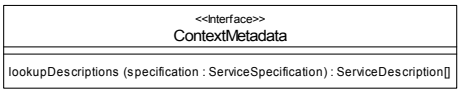

Figure 55: Interface **`ContextMetadata`**

As mentioned when introducing the basic contents of the information logistics framework, the Context Component is required to additionally define protocol-specific interfaces by means of which the component's functionality can be accessed using a particular communication protocol. Thus, the external viewpoint also defines the interfaces `ContextComponentRMI` and `ContextMetadataRMI` for communication on the basis of Java Remote Method Invocation (RMI) [Sun04] which currently is the standard method of communication among the framework's components. Further protocol-specific interfaces such as `ContextCompo-`

nentEJB and ContextMetadataEJB used to implement Enterprise Java Beans (EJB) [DeMi02], for example, may be defined on demand. Each protocol-specific interface extends both the corresponding functional interface – i.e. ContextComponent or ContextMetadata – and one or more protocol-specific interfaces required to implement the respective type of communication, in this case java.rmi.Remote as depicted in Figure 56.

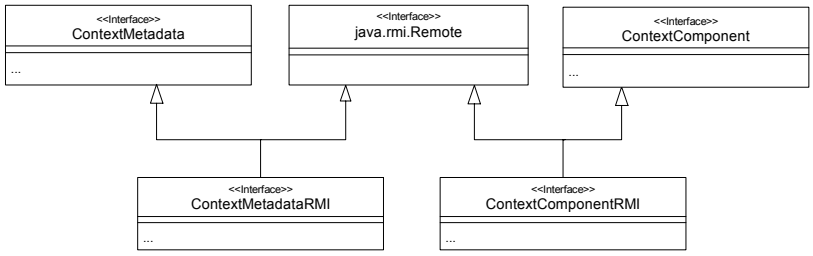

Figure 56: Communication protocol-specific interfaces of the Context Component

6.2.2 Logical viewpoint

Similar to the external viewpoint the logical viewpoint addresses the functional requirements made onto the Context Component. Unlike the former, however, the logical viewpoint deals with the question of how to achieve the component's functionality and therefore is primarily concerned with the internal structure of the Context Component. Although the logical viewpoint is also based on the services the component provides to clients, it deals with the implementation of these services rather than with the way they are made available.

In the logical viewpoint the computational elements required to implement the Context Component's functionality, their relationships to each other and to external elements, and their interfaces are identified and described. The component is decomposed into object classes representing these computational elements. The contribution of the logical viewpoint to the architecture of the Context Component therefore is an object model which is independent of any particular application domain. Since the logical viewpoint deals with the internal details of the Context Component, it addresses the concerns of developers, testers, and integrators responsible for embedding the Context Component into information logistic applications and for integrating external data sources and services into it. The object model provided by the logical viewpoint is represented by UML class diagrams. What is called logical viewpoint in our approach is also referred to as static, application, or information viewpoint in other architectures. Yet, in our approach the logical viewpoint is limited to the internal details of the Context Component. Table 4 gives a summary of the logical viewpoint's characteristics.

Viewpoint name	Logical
Stakeholders	Developers, testers, and integrators

Table 4: Characteristics of the logical viewpoint

Concerns	- What are the component's computational elements? - How are these elements structured? - How do they relate to each other? - How do they relate to external elements? - What interfaces do the component's elements possess?
Viewpoint language	Classes, attributes, methods, roles, interfaces and their parameters (UML class diagrams)
Viewpoint source	Also known as static, application, or information viewpoint

Table 4: Characteristics of the logical viewpoint

As mentioned before, the Context Component's tasks are the representation, determination, management, and supply of context data. The logical viewpoint provides an object model containing the computational elements required to carry out these tasks. Due to the complexity of the structures representing contexts and of those necessary for context gathering the individual object models for these structures have been devoted separate chapters. In Chapter 4 we have presented the object model for the representation of context. Subsequently, an object model for the entities involved in the context gathering process has been described in Chapter 5. The provision of context data to the Context Component's clients has been dealt with in the external viewpoint of the Context Component described in the previous section. Therefore, as far as these object models are concerned we refer the reader to the previous chapters and to Section 6.2.1 and restrict our explanations concerning the logical viewpoint to a further aspect of the component's internal structure that has not been dealt with so far.

In Section 6.1 we have already mentioned that the information logistics framework commits each component to being subdivided into a front-end responsible for obtaining references to other components and to basic services and a back-end implementing the actual functionality of the component. A component's front-end implements the communication protocol-specific component interfaces as well as the protocol-specific child of an interface called `Administerable`. The interface `Administerable` is defined by the information logistics framework and provides generally required mechanisms for managing a component's life cycle.

For the Context Component this means that a front-end, represented by the class `ContextComponentFrontEnd`, is required which has to implement the interfaces `ContextComponentRMI`, `ContextMetadataRMI`, and `AdministerableRMI`, on condition that the default communication mechanism RMI is employed. An instance of the class `ContextComponentFrontEnd` makes requests to the Service Broker as a result of which it is provided with references to other components the Context Component requires. The `ContextComponentFrontEnd` object forwards both these references as well as all requests made to itself to the component's back-end. The back-end is represented by the `ContextComponentBackEnd` class the `ContextComponentFrontEnd` class is associated with as depicted in Figure 57.

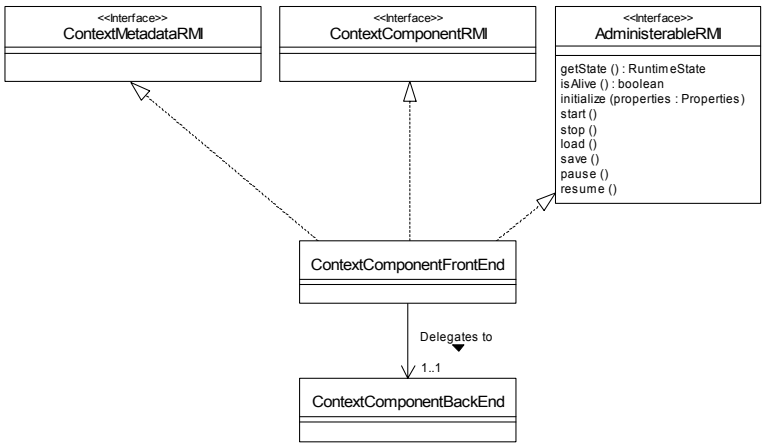

Figure 57: Context Component front-end

Since a component's back-end is responsible for providing the actual functionality of the component, it has to implement the generic component interfaces. In order to be able to change its state a component's back-end furthermore implements the `Administerable` interface. In addition to this, the information logistics framework defines so-called dependency interfaces for all components and basic services. By means of these interfaces' methods a component's back-end is passed and withdrawn from references to other components and services by its front-end. Each dependency interface possesses methods to add and remove a reference to the respective component or service. The dependency interface which is newly defined for the Context Component to enable other components to interact with it is shown in Figure 58.

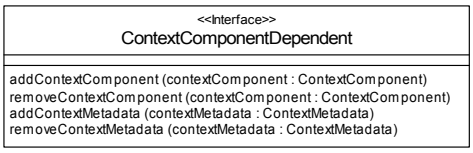

Figure 58: Dependency interface for the Context Component

Like all components of information logistic applications the Context Component needs to have access to basic services for logging, profile management, and error handling, and to the internal clock. Thus, apart from the interfaces `ContextComponent`, `ContextMetadata`, and `Administerable` the Context Component's back-end, represented by the class `ContextComponentBackEnd`, implements the respective dependency interfaces of these services. The class `ContextComponentBackEnd` is associated with the classes `ContextGatherer` and `ServiceRegistry` which effectively provide the Context Component's functionality made available by means of its interfaces as explained in Chapter 5. The back-end of the Context Component is illustrated in Figure 59.

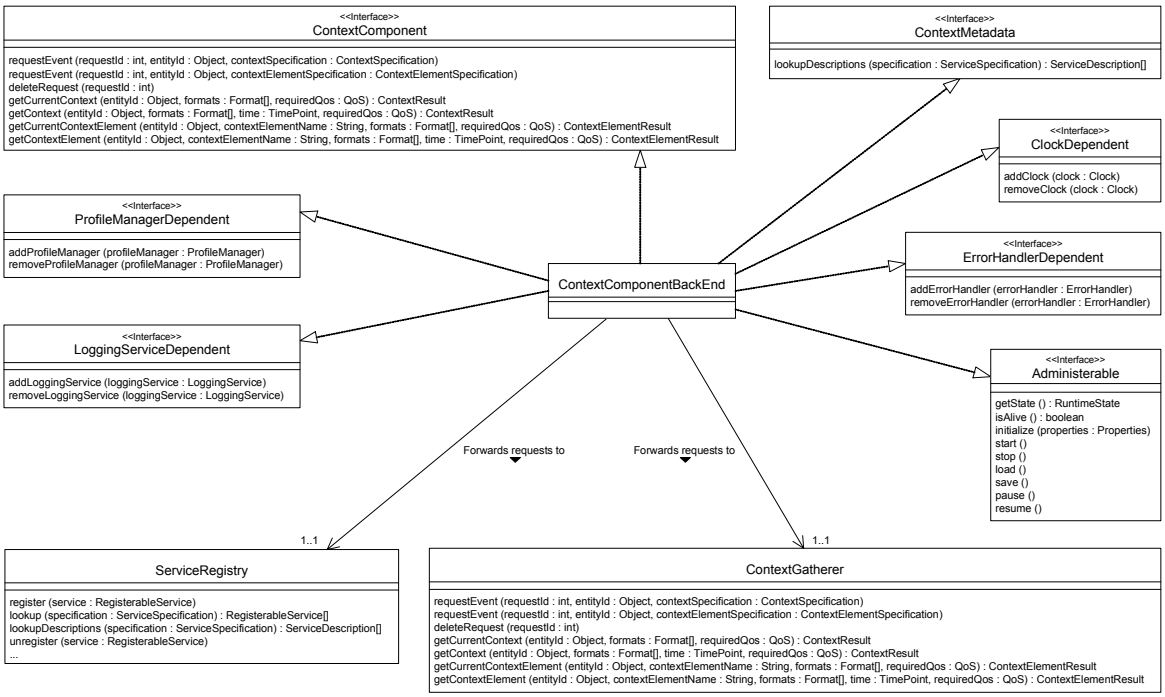

Figure 59: Context Component back-end

6.2.3 Dynamic viewpoint

The dynamic viewpoint of the Context Component identifies and describes the processes that are executed within the component. The exchange of messages between the computational elements the Context Component consists of and the sequence of these message exchanges are examined. In addition, the behaviour of the component's individual elements, their reaction to incoming messages and events, is defined. The dynamic viewpoint thus deals with the flow of information between the Context Component's computational elements and its chronology. It therefore not only describes the processes that have to be carried out to provide the component's functionality, but also is concerned with some non-functional requirements such as concurrency, availability, and fault-tolerance. Accordingly, this viewpoint addresses the concerns of component developers, testers, and integrators and of systems management. The behaviour of and interactions between the Context Component's computational elements is described by means of UML statechart diagrams and UML sequence diagrams. In other approaches the dynamic viewpoint is also referred to as process or computational viewpoint. The characteristics of this viewpoint are summarized in Table 5.

Viewpoint name	Dynamic
Stakeholders	Developers, testers, integrators, and systems management

Table 5: Characteristics of the dynamic viewpoint

Concerns	- Which processes are executed within the component? - What messages are exchanged between the component's computational elements? - In what sequence are these messages exchanged? - What is the behaviour of the component's computational elements?
Viewpoint language	Objects, messages, events, actions (UML statechart diagrams, UML sequence diagrams)
Viewpoint source	Also known as process or computational viewpoint

Table 5: Characteristics of the dynamic viewpoint

The dynamic viewpoint of the Context Component first describes the component's life cycle. This life cycle consists of the states the Context Component can be in, the events causing a transition from one state to another, and the actions performed in each state. The description of the Context Component's life cycle thus prescribes the overall operation of the component as a whole, its reaction to messages received from the Service Broker or the Service Manager, and the possible sequences of actions that may take place within the component. The life cycle of the Context Component is illustrated in Figure 60.

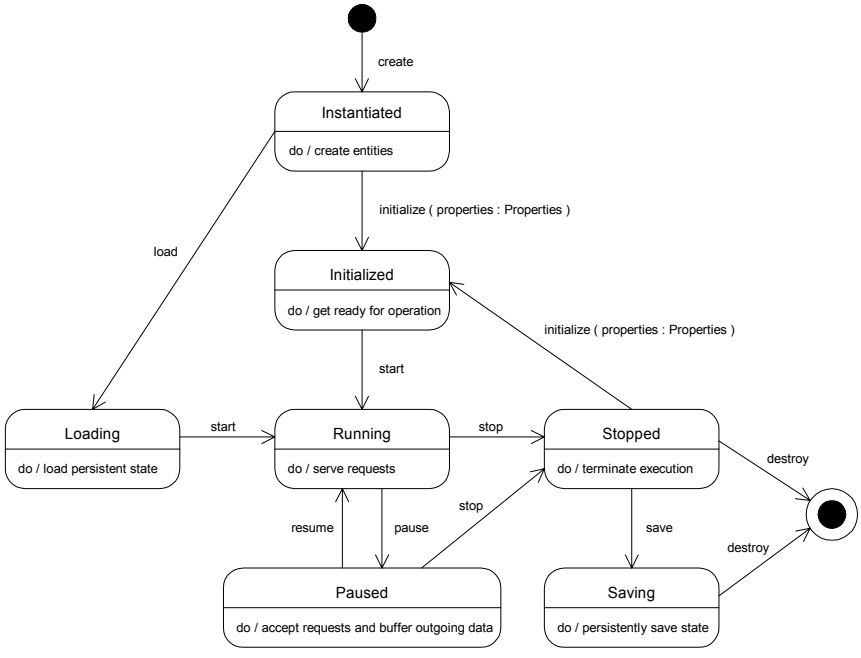

Figure 60: Life cycle of the Context Component

The Context Component is created by instantiating the class `ContextComponentFrontEnd`. This is caused by a `create` event received from the Service Broker. Upon receipt of this event the component is in the `Instantiated` state during which all other required instances

of the classes the component consists of are successively created and supplied with references to associated objects. While the Context Component is in the `Instantiated` state, two different events may occur. On the one hand, by means of a `load` event received from the Service Manager the component may be caused to restore a previously saved state from persistent storage. In this case a transition into the `Loading` state takes place. While in this state the Context Component retrieves information concerning pending requests from a storage device and forwards it to the corresponding entities responsible for the processing of these requests. By means of the `Loading` state and the corresponding `Saving` state explained below the component is able to be safely shutdown and restarted – possibly on a different computer – without information loss and without the need for clients to repeat previously made requests. On the other hand, while in the `Instantiated` state the Context Component may receive an `initialize` event containing appropriate initialization parameters which causes it to enter the state `Initialized`. During this state the component performs all operations necessary to become ready for operation. This includes the initialization of the available sensor adaptors and their registration with the service registry.

From both the `Loading` and the `Initialized` state a transition into the state `Running` takes place upon receipt of the `start` event. When in the `Running` state the Context Component provides its functionality to clients and serves the requests made to it. Details of the processing taking place within this state are explained later on in this section. When running the Context Component may be either paused or stopped. A `pause` event causes a transition into the state `Paused`. During this state the component still accepts requests, but does no longer perform any activities visible to entities outside the component. In particular, while being paused the Context Component does not fire any events in response to asynchronous requests. The component returns to the state `Running` upon receipt of the `resume` event. From the `Running` state the Context Component enters into the state `Stopped` as soon as the event `stop` is received. A transition into this state caused by the same event is also possible from the `Paused` state. While stopped the Context Component terminates its execution and thus no longer provides functionality to clients. In this state the component may be induced to reinitialize itself by means of the `initialize` event, causing it to again enter the state `Initialized`. Furthermore, if pending requests are to be persistently stored, the component is sent a `save` event in reaction to which it enters the state `Saving`. Both the `Saving` and the `Stopped` state may be followed by the destruction of the Context Component triggered by the event `destroy` which, like the `create` event, is sent by the Service Broker.

Of these states the Context Component can be in during its life cycle the state `Running` is particularly important as by means of the actions performed in this state the actual functionality of the component is provided to clients. Therefore, the processes that are executed within this state and the behaviour of the computational elements involved in these processes are dealt with in greater detail in the following paragraphs.

First we illustrate the overall interactions between the Context Component's objects, i.e. the exchange of messages between them along with their sequence, taking place when a request is made to the component. These interactions have already been touched on in Chapter 5 when explaining context gathering mechanisms. Yet, the dynamic viewpoint of the Context Component exclusively deals with this issue and provides an in-depth study of the processes that are executed. In Figure 61 the interactions for a synchronous request made to the Con-

text Component referring to the entire current context of an entity are shown. In case of asynchronous requests or requests referring to a single context element or to a dedicated point in time the message exchanges differ only slightly as explained shortly. Since the objects involved in these interactions have already been described in detail, neither these objects nor the parameters of the messages they exchange are explained again in this section.

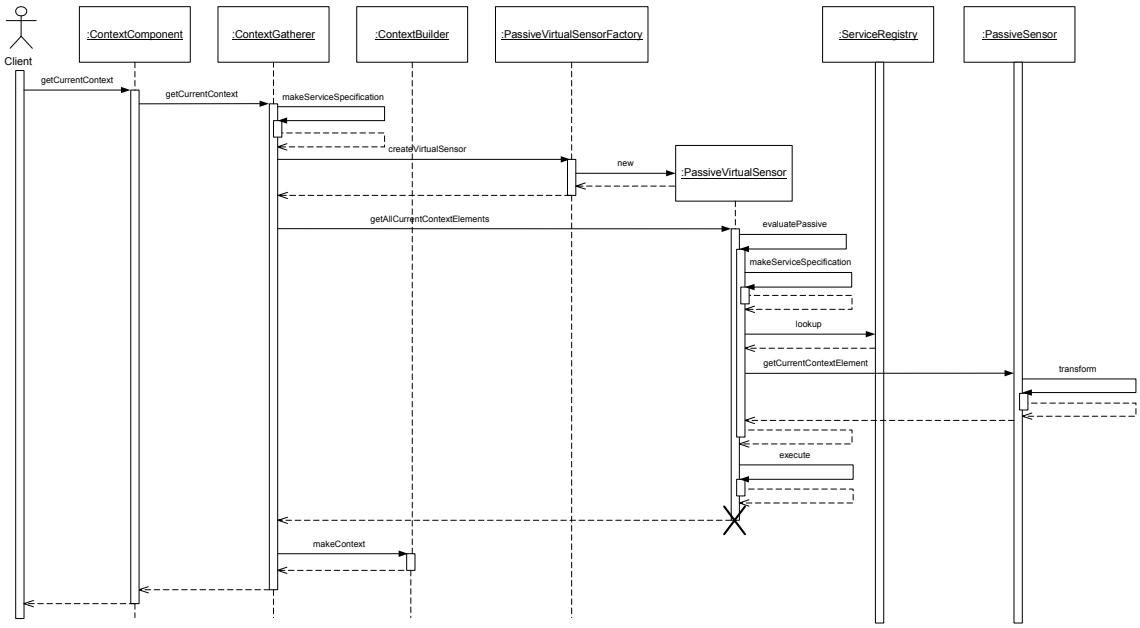

Figure 61: Message exchange within the Context Component

The Context Component starts its processing upon receipt of a client's request. The request is made to the front-end of the component which forwards it to the component's back-end that implements the interface `ContextComponent`. The back-end again forwards the request to an instance of the `ContextGatherer` class. This instance creates a `ServiceSpecification` object on the basis of this request and its parameters and passes it as a parameter of the `createVirtualSensor` message to a `VirtualSensorFactory` instance. In the example shown in the figure this is a `PassiveVirtualSensorFactory` object. In case of asynchronous requests the `ContextGatherer` instance interacts with an instance of the `ActiveVirtualSensorFactory` class instead. Upon receipt of the `createVirtualSensor` message the `PassiveVirtualSensorFactory` object creates a new instance of the `PassiveVirtualSensor` class and returns it to the `ContextGatherer` object. At this point an `ActiveVirtualSensor` instance is created if the request is an asynchronous one.

After that the `ContextGatherer` object makes a request to the `PassiveVirtualSensor` instance; in our example this is done by a `getAllCurrentContextElements` message. In order to process this request the `PassiveVirtualSensor` sends an `evaluatePassive` message to itself or, more precisely, to the `Condition` object it is associated with. The structures virtual sensors consist of, however, are not explicitly shown in Figure 61 for reasons of clarity; the interactions between these structures is instead dealt with separately later on. Dur-

ing the evaluation of the condition the request parameters are again converted into a `ServiceSpecification` object which is used as a parameter of the `lookup` message sent to the `ServiceRegistry` object managing the available sensor adaptors. After being returned appropriate sensor adaptors the `PassiveVirtualSensor` object interacts with these adaptors which are objects of the type `PassiveSensor`. It sends a request for the determination of a particular context element to each of them; in our example this is a `getCurrentContextElement` message. The `PassiveSensor` objects thereupon query the underlying context sensors and transform their data into `ContextElement` objects for the purpose of which they send a `transform` message to themselves. After that the `PassiveSensor` objects return their results to the `PassiveVirtualSensor` instance which performs necessary pre-aggregation, combination, and derivation operations upon them. For this purpose it sends an `execute` message to itself which again is actually a message to associated `Action` objects. After having generated its results the `PassiveVirtualSensor` instance returns them to the `ContextGatherer` object and is destroyed. The `ContextGatherer` object then charges a `ContextBuilder` instance with the filtering and aggregation of the received data by means of a `makeContext` message. In case of a synchronous request referring to a single context element a `makeContextElement` message is sent. The results returned from the `ContextBuilder` object are forwarded from the `ContextGatherer` instance to the `ContextComponent` object which in turn returns them to the client. Since in our example a synchronous request is made, the client waits for the receipt of this result before proceeding.

Please note that the Java programming language used to implement information logistic applications inherently supports multitasking. Therefore, several concurrent requests can be made to the Context Component and to the computational elements it consists of. As can be seen in Figure 61, the objects within the Context Component do not simultaneously access shared resources. As a result, synchronization mechanisms to coordinate the interactions between these objects are not required in the dynamic viewpoint.

Since requests made to the Context Component may be either synchronous or asynchronous, may refer to a single context element or an entire context, to the current or a dedicated point in time, and may contain various different combinations of parameters, the dynamic viewpoint cannot provide a detailed description of any possible interaction between the component's entities. In the example given above the differences between the individual types of requests have been pointed out. Therefore, the sequence of actions and message exchanges for different types of requests made to the Context Component are assumed to be easily accessible.

We now examine the principal elements involved in serving requests made to the Context Component in greater detail. In doing so, we begin with the lowest layer of context gathering components, the sensor adaptors. Again, since not any possible combination of interactions can be described explicitly, we continue the abovementioned example by explaining the interactions that take place upon receipt of a synchronous request. Possible variations of interactions are indicated where necessary.

The creation and initialization of a sensor adaptor are caused by corresponding messages a `Sensor` instance – in our example a `PassiveSensor` object – receives from an object implementing the `Administerable` interface, i.e. by the `ContextComponentBackEnd` instance. After having received an `initialize` message the `Sensor` object creates a

`ServiceDescription` object containing information about its interfaces and capabilities. This object is passed to an instance of the `ServiceRegistry` class when the `Sensor` object registers with it by means of a `register` message. Upon receipt of a `start` message from an `Administerable` instance the `Sensor` object establishes a connection to the context sensor it encapsulates. After these steps a sensor adaptor is ready to serve requests made to it by virtual sensors. The interactions sensor adaptors are involved in are shown in Figure 62.

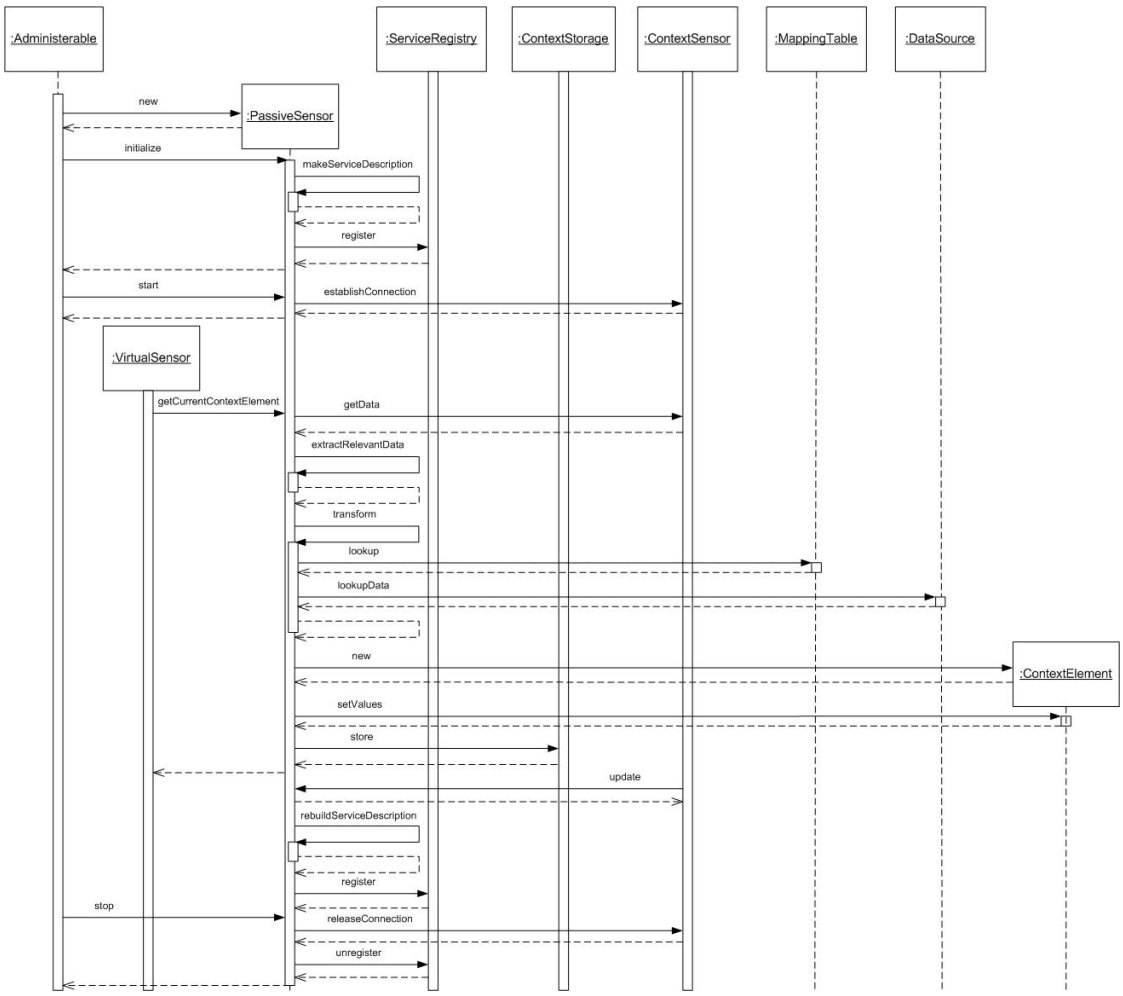

Figure 62: Interactions with the participation of sensor adaptors

Upon receipt of a request for the determination of a context element from a virtual sensor a `Sensor` object obtains the requested data from the context sensor. The `Sensor` instance may have to remove unnecessary information from these data, depending on the data provided by the respective context sensor. If this step is required, the `Sensor` object sends an `extract-RelevantData` message to itself. Afterwards, when the relevant sensor data are available, the `Sensor` object sends a `transform` message to itself which triggers the transformation of the sensor data into context data. During this transformation the `Sensor` object first consults the mapping table that contains instructions regarding the mapping of sensor data onto a con-

text element, its attributes and values. If necessary, the Sensor object sends messages to data sources containing additional data required for the transformation of sensor data to get provided with these data. After that a ContextElement instance is created by the Sensor object and is assigned the appropriate values by means of corresponding setValue messages. If the created context data are to be made persistent, the Sensor object sends a store message to a context storage, causing it to persistently store the ContextElement object it is passed in this message. The store message may also be sent by the Sensor object before the transformation takes place if sensor data instead of context data are to be stored. It may also be omitted if no persistent storage of either sensor or context data is desired. After these steps the Sensor object provides its client, the VirtualSensor instance, with the results of its processing.

Whilst started a sensor adaptor may be notified that changes have been made to the context sensor it interacts with which affect the sensor adaptor's service description. For this purpose an update message is sent from the context sensor to the belonging Sensor object. Upon receipt of this message the Sensor object sends a rebuildServiceDescription message to itself and updates its associated ServiceDescription object. After that it reregisters with the ServiceRegistry object by means of another register message.

A sensor adaptor is induced to terminate its execution by means of a stop message it receives from an Administerable object. In reaction to this message it disconnects from the context sensor and unregisters with the ServiceRegistry object by sending it an unregister message.

We have already mentioned above that the structure of virtual sensors is too complex to be dealt with completely in conjunction with the overall sequence of messages within the Context Component. Thus, the interactions between the elements on the virtual sensor layer are now examined in detail. A VirtualSensor object is created by a VirtualSensorFactory instance and is responsible for serving exactly one request made by a ContextGatherer object. To continue our example Figure 63 shows an instance of the PassiveVirtualSensorFactory class which upon receipt of the createVirtualSensor message first searches for Context SRML rules suitable for the request. If appropriate rules are available, they are passed to the new instance of the class VirtualSensorImpl the PassiveVirtualSensorFactory object creates. In our example this instance implements the interface PassiveVirtualSensor. The ContextGatherer object then makes a request to it. If the VirtualSensorImpl instance has been passed Context SRML rule files upon creation, it first evaluates these rules in order to determine which types of conditions and actions are defined in them and which Condition and Action objects correspondingly have to be created. After that the VirtualSensorImpl object creates the required objects on the basis of the existing rules and the parameters of the request message. In our example a Hashtable object, an OrCondition object, two or more SimpleCondition objects as well as a CompoundAction object and two or more SimpleAction objects are created. Depending on the type of request made to the virtual sensor and on the relevant rules the Hashtable, CompoundCondition, and CompoundAction objects may not be needed and different subtypes of the SimpleCondition and SimpleAction classes need to be created.

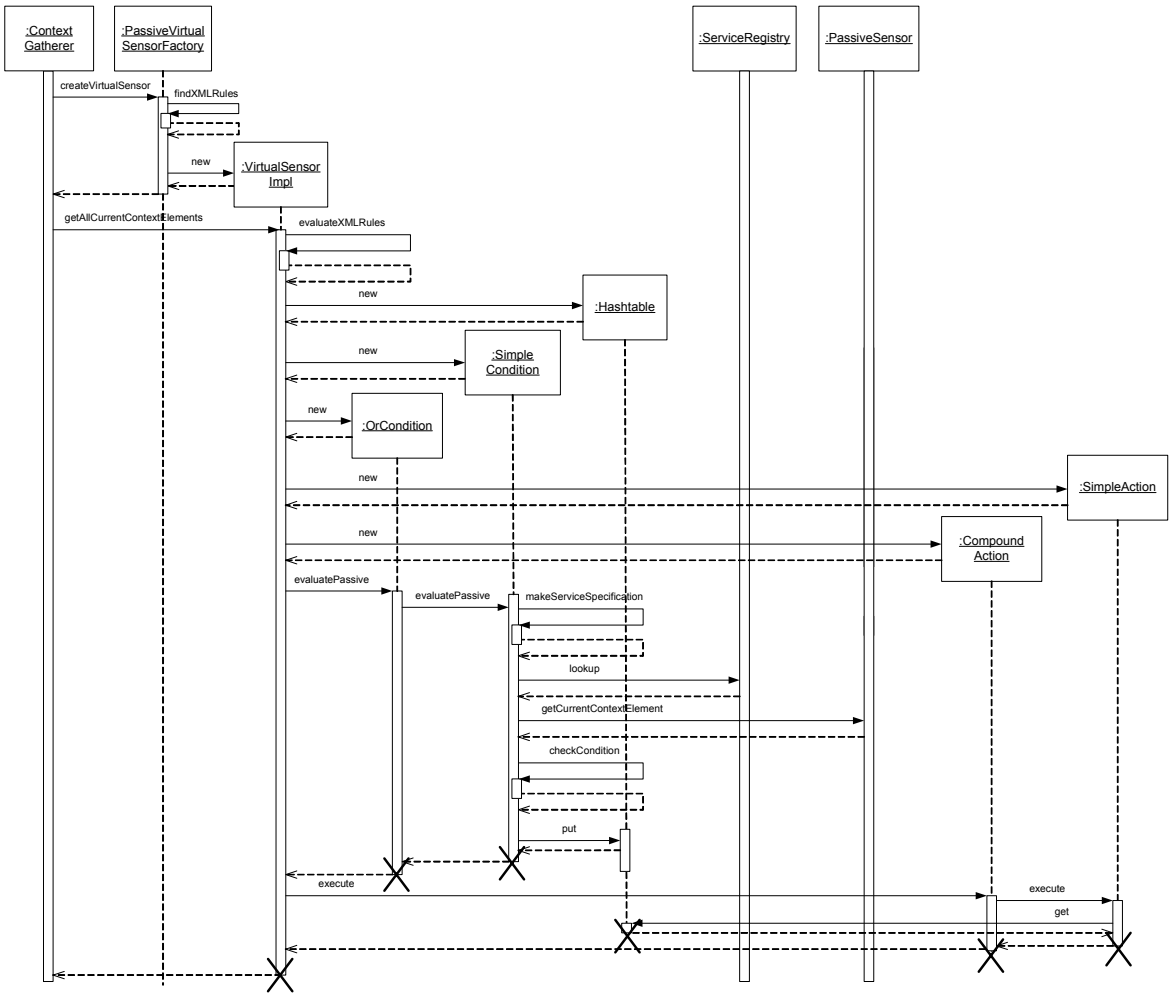

Figure 63: Interactions with the participation of virtual sensors

After these objects have been created the conditions are evaluated. This is done by an `evaluatePassive` message like in our example or an `evaluateActive` message if the request made by the `ContextGatherer` object is asynchronous. This message contains the received request parameters. It is sent to a `Condition` object which can be either a `CompoundCondition` or, if only a single condition exists, a `SimpleCondition` object. In case of a synchronous request referring to more than one context element the message is sent to an instance of the `OrCondition` class as shown in the figure. Upon receipt of this message a `CompoundCondition` instance forwards the message to the corresponding `SimpleCondition` objects after having related the `ContextElementSpecification` objects contained in it in case of an asynchronous request referring to more than one context element to these objects. The `SimpleCondition` objects thereupon send a `makeServiceSpecification` message to themselves, resulting in the creation of an appropriate `ServiceSpecification` object. This object is then passed to the `ServiceRegistry` instance managing the sensor adaptors as a parameter of the `lookup` message. After being returned suitable `Sensor`

objects by the registry the `SimpleCondition` instances query these objects by means of an appropriate message, in our example `getCurrentContextElement`, and are supplied with results. The `SimpleCondition` objects may have to examine the received results for compliance with specified rules; in this case they send a `checkCondition` message to themselves for this purpose. Moreover, if the obtained results are to be made available to the `Action` objects for further processing, they are written into the `Hashtable` object by means of the `put` message sent to it. The `VirtualSensorImpl` object is then returned the result of the evaluation, and the `Condition` objects are destroyed.

If the conditions indicate that context data are available, i.e. if they return `true`, the `VirtualSensorImpl` instance sends `execute` messages to the `Action` objects it has created. In our example the message is sent to a `CompoundAction` instance which forwards it to the `SimpleAction` objects it aggregates. In order to carry out its task a `SimpleAction` instance may have to obtain data from the `Hashtable` object that has been filled by the `Condition` objects before. If this is the case, it sends a `get` message to the `Hashtable` object and is provided with the requested data. Afterwards the `Hashtable` object is destroyed. The `SimpleAction` instances then return their results to the caller, either to the `CompoundAction` object which aggregates these results or to the `VirtualSensorImpl` instance if no `CompoundAction` object exists. Finally, after the `VirtualSensorImpl` instance has received the results, the `Action` objects are destroyed, and the results are returned by the `VirtualSensorImpl` instance to the `ContextGatherer` object.

The top layer involved in providing the Context Component's functionality contains context builders. Again we describe the sequence of message exchanges context builders participate in with the help of the same example as above. The interactions context builders are involved in are illustrated in Figure 64.

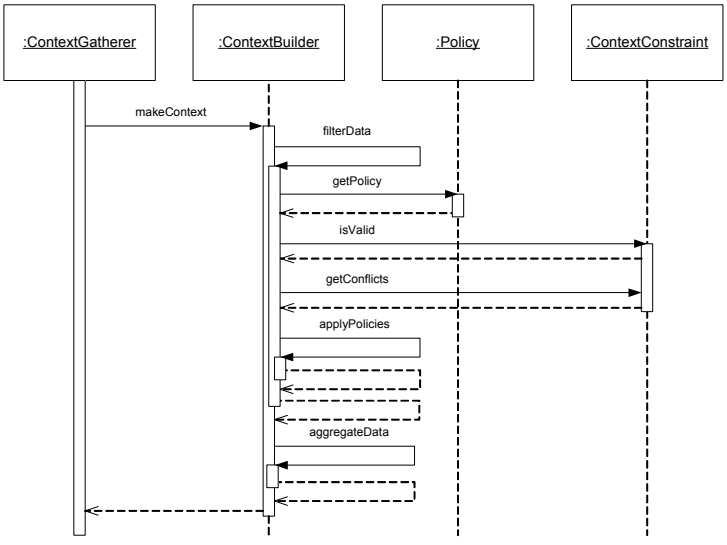

Figure 64: Interactions with the participation of context builders

Context builders are charged by `ContextGatherer` objects with the filtering and final aggregation of context data into a `Context` or a `ContextElement` object. For this purpose they are sent a `makeContext` or a `makeContextElement` message by the `ContextGatherer` instance. The received data first have to be filtered. Thus, a `ContextBuilder` instance sends a `filterData` message to itself. The filtering of context data is carried out on the basis of constraints or policies regarding permissible combinations of data, quality of service criteria, etc. The `ContextBuilder` instance therefore first retrieves policies that are relevant for the respective context data and consults the available `ContextConstraint` instances by means of the `isValid` and `getConflicts` messages to determine if there are conflicts in the context data. It then performs the filtering by applying these policies and constraints. After that, the `ContextBuilder` instance sends an `aggregateData` message to itself whereupon the remaining context data are aggregated into one or more `ContextElement` objects and possibly into a `Context` object as well. The result of the `ContextBuilder` instance's processing is then returned to the `ContextGatherer` object.

Finally, Figure 65 shows the sequence of actions taking place when the `lookupDescriptions()` method of the `ContextMetadata` interface is called by a client. This method may be passed a `ServiceSpecification` object as a parameter. In this case the `ServiceRegistry` instance receiving the `lookupDescriptions` message from a `ContextMetadata` object consults a `ServiceMatchingService` object to obtain the descriptions of registered sensor adaptors that fulfill this specification. For this purpose it sends a `getMatching` message to the `ServiceMatchingService` instance as shown in the figure and is returned `ServiceDescription` objects of suitable sensor adaptors which it in turn returns to the client. If no parameter is passed to the `lookupDescriptions()` method, the `ServiceRegistry` object simply returns the descriptions of all sensor adaptors registered with it.

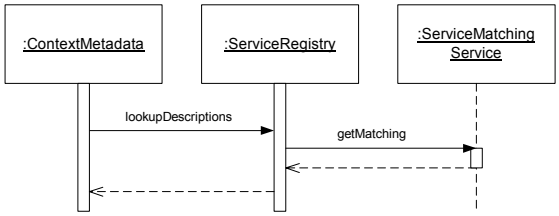

Figure 65: Message exchange for the provision of metadata

6.2.4 Structural viewpoint

The structural viewpoint of the Context Component decomposes the component software into small units, i.e. into packages and modules. The viewpoint defines which packages and modules exist and how they are organized. Dependencies among these software units are identified and described. The structural viewpoint addresses concerns related to software development such as the structure and size of compilation units, reuse, or software management. It serves as a basis for the management of the component development process, the allocation of work to teams, cost and resource planning, the monitoring of progress, the reuse of packages and modules, etc. Thus, apart from developers, testers, and integrators the struc-

tural viewpoint also addresses managers responsible for controlling the software development process. The software units the Context Component is composed of are described in the form of packages along with the relationships among them as well as in the form of UML component diagrams. The structural viewpoint is similar to the development viewpoint proposed by Kruchten [Kruc95]. Table 6 shows a summary of this viewpoint's characteristics.

Viewpoint name	Structural
Stakeholders	Developers, testers, integrators, and software engineering management
Concerns	- How is the software organized, how are the elements of the software partitioned and grouped? - Which compilation units exist? - Which dependencies exist among the packages and modules of the software? - How is software reuse supported?
Viewpoint language	Packages and their access and import relationships, modules and their dependencies (UML component diagrams)
Viewpoint source	Also known as development viewpoint

Table 6: Characteristics of the structural viewpoint

In the structural viewpoint we first describe the organization of the Context Component software into packages. The information logistics framework defines the basic package structure components belonging to it have to comply with; in particular this means that all packages defined by core components must be nested within the package `de.fhg.isst.ilog`. Correspondingly, the top-level package the Context Component software modules belong to is `de.fhg.isst.ilog.context`. In this package several subordinate packages are nested either directly or indirectly, resulting in a package hierarchy which is a tree structure.

The `de.fhg.isst.ilog.context` package contains the basic software elements required for the representation of contexts and for the interaction of the Context Component with its clients. Within this package two other packages are nested, one of which contains those parts of the software that implement the representation of context elements, while in the other one the software elements required in conjunction with the determination and supply of context are contained. In both of these packages, called `elements` and `services`, respectively[1], again several other packages are nested. The packages within the `elements` package contain that parts of the software which represent the individual context elements and the attributes required in conjunction with some of them. Since the context elements of location and state both possess a complex structure involving a number of classes, the corresponding packages `location` and `state` are again subdivided. The software elements which represent the different aspects of locations, their structure, coordinates, prepositions, and the containers for

1. The prefixes of the packages nested within the `de.fhg.isst.ilog.context` package are omitted in the following in order to make the text easier to read.

them, are each organized in an individual package. Similarly, separate packages corresponding to the elements of state exist within the `state` package. The `services` package contains those parts of the component software which carry out the determination and supply of context data. It is subdivided into three packages that correspond to the hierarchy of context gathering components introduced in Chapter 5. Since context gatherers are implemented by a single class, they are grouped with context builders in a single package called `core`. Figure 66 shows the packages the Context Component software is organized into along with the packages they interact with and the type of these relationships.

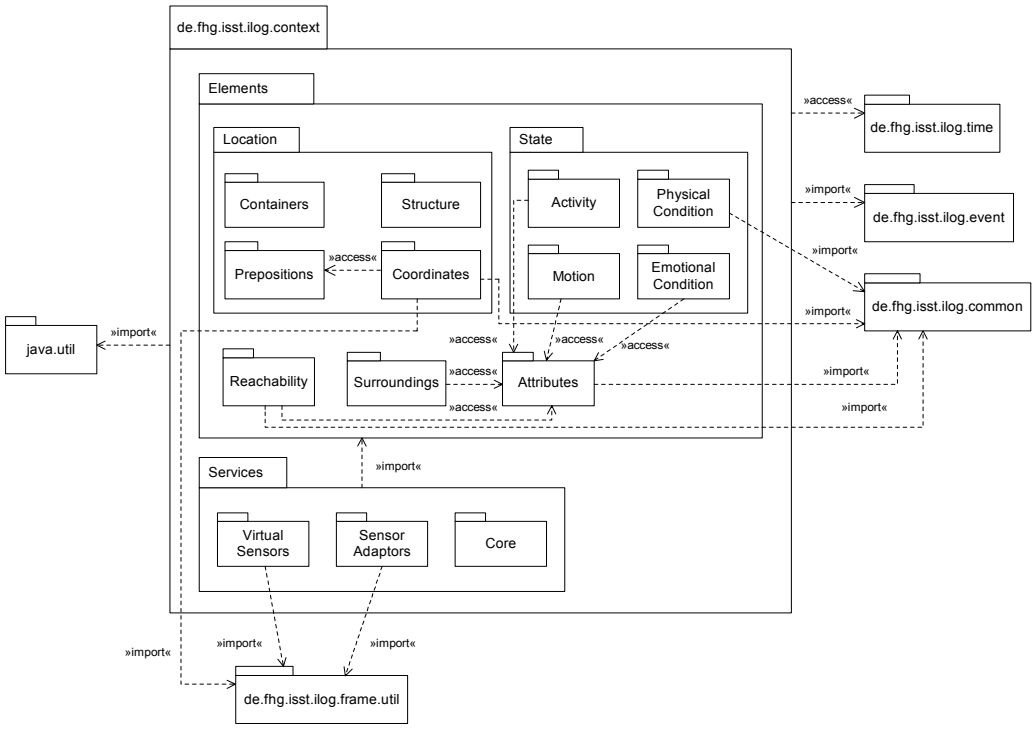

Figure 66: Context Component packages and their relationships

Several classes within packages nested in the `de.fhg.isst.ilog.context` package need to have access to the contents of other nested packages. The state elements of activity, motion, and emotional condition as well as the context elements of reachability and surroundings are represented using `Attribute` objects. Thus, the `attributes` package is accessed by all packages containing the software elements for the former structures. In addition, coordinate systems define the permissible prepositions for locations the coordinates of which refer to them. Therefore, the `prepositions` package is accessed by the `coordinates` package. Since during the context gathering process `ContextElement` objects are created within the `services` package, this package needs to have access to the package `elements`.

For its compilation and execution the Context Component software furthermore needs to have access to several other software elements which either are defined in other components and services of the information logistics framework or are part of the Java library. The event han-

dling mechanism of the framework and different representations of time such as the `TimePoint` class mentioned before are implemented by classes defined in the packages `de.fhg.isst.ilog.event` and `de.fhg.isst.ilog.time`, respectively. The corresponding structures are made use of throughout the Context Component software. Therefore, the package `de.fhg.isst.ilog.context` imports both of these external packages. Moreover, structures that are frequently used within several components of the information logistics framework such as classes representing values and value dimensions are also defined outside the Context Component in the `de.fhg.isst.ilog.common` package. Since representations of values are required in various parts of the context model, this package is imported by the respective packages nested within the `de.fhg.isst.ilog.context` package. The `ServiceRegistry` class made use of within the Context Component is a basic utility class belonging to a package called `de.fhg.isst.ilog.frame.util` which correspondingly is imported by the packages `virtualsensors`, `sensoradaptors`, and `coordinates`. In addition, apart from the basic `java.lang` package the package `java.util` defining structures for the representation of sets and lists and containing the `Hashtable` class also needs to be imported by the Context Component software. In order to support specific communication protocols or to make use of further features of the Java language other packages defined in the Java library which are not shown in Figure 66 may have to be imported as well.

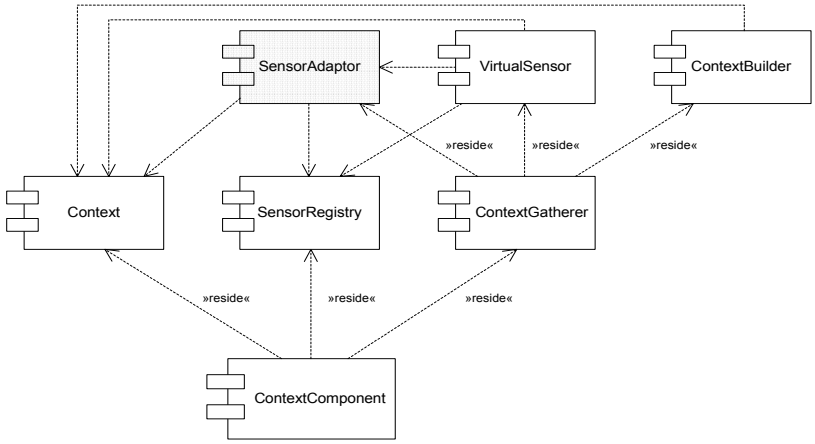

Figure 67: Context Component modules and their relationships

Each of the packages the Context Component software is subdivided into represents a separate compilation unit the development of which can be assigned to an individual or a team. As a result, the component software is organized in a modular way which facilitates the component development and the management of the development process. As mentioned in Section 6.1, a core component of the information logistics framework is composed of several modules each of which implements a specific part of the component's overall functionality. While packages represent compilation units the purpose of which is to facilitate software development and its management, a component's modules are more coarse-grained as they serve to implement a coherent functionality. A module may be composed of several packages.

Modules may be regarded as subcomponents of the Context Component. Accordingly, Figure 67 shows the modules the Context Component consists of and their relationships to each other by means of a UML component diagram.

The basic functionalities of the Context Component are the representation of context, its determination, management, and supply, and the provision of metadata concerning the available context sensors. Each of these functionalities is implemented in a specific module or, in other words, subcomponent. The module `ContextGatherer` which serves to implement the determination and supply of contexts is again subdivided into three smaller modules corresponding to the layered structure of the entities involved in the context gathering process. Among them the module `SensorAdaptor` is made stand out by means of a dotted pattern. This is due to the fact that in contrast to all other modules of the Context Component this module is application-specific in terms of the information logistics framework. Although possessing a universal interface, sensor adaptors implement sensor-specific mechanisms to access context sensors and to obtain data from them. The programme code implemented in sensor adaptors varies in every information logistic application, depending on the number and type of context sensors employed. Thus, the `SensorAdaptor` module contains application-specific logic that is likely to require modifications in any new installation of an information logistic application and is therefore not within the scope of the information logistics framework.

For registration and lookup purposes both sensor adaptors and virtual sensors call methods defined in the `ServiceRegistry` interface which is made available by the `SensorRegistry` module. Since virtual sensors furthermore are the clients of sensor adaptors, they access the services provided by the `SensorAdaptor` module. In addition, sensor adaptors, virtual sensors, and context builders create `ContextElement` and `Context` objects, respectively. In doing so, they call methods defined in the `Context` module's interfaces. Corresponding dependency relationships between these modules are therefore drawn in Figure 67.

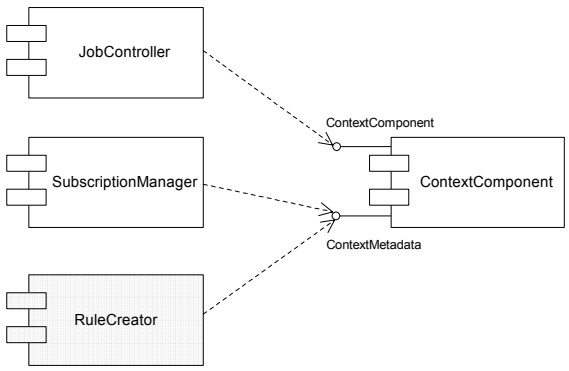

Figure 68: Clients' relationships with the Context Component

We conclude our explanations concerning the structural viewpoint of the Context Component by briefly illustrating the dependencies between the Context Component and its clients. As can be seen in Figure 68, the services the Context Component makes available are mainly provided to the Job Controller and the Subscription Manager of information logistic applications. The Job Controller accesses the `ContextComponent` interface in order to be supplied with con-

text data required for the execution and optimization of information supply. In contrast to this, the Subscription Manager requires information concerning the available context sensors and their capabilities. It thus accesses the `ContextMetadata` interface in order to prevent the definition of subscriptions that refer to context data which cannot be determined in an application. Apart from these core components of information logistic applications further additional components may exist which also require the metadata the Context Component supplies. In Figure 68 a Rule Creator component is exemplarily shown which serves to provide users with the opportunity to define rules for context data combination and derivation (see Section 5.2.4) and to translate these rules into Context SRML files.

6.2.5 Physical viewpoint

The physical viewpoint of the Context Component deals with the configuration of the component at run-time. The modules the component software is composed of are distributed over a set of processing resources, i.e. nodes. The physical viewpoint describes how modules are mapped to nodes and which communication associations exist between the Context Component's modules. Therefore, this viewpoint primarily takes into consideration non-functional requirements made onto the Context Component such as performance, fault-tolerance, scalability, and so on. The physical viewpoint accordingly addresses the concerns of developers, integrators, testers, and of systems management. The distribution of modules – which can also be regarded as subcomponents as mentioned previously – over nodes and the existing communication associations are described by means of UML deployment diagrams. The physical viewpoint of the Context Component is based on the viewpoint of the same name introduced by Kruchten [Kruc95] and is similar to what is called engineering viewpoint in other approaches. Table 7 shows a summary of the physical viewpoint's characteristics.

Viewpoint name	Physical
Stakeholders	Developers, integrators, testers, and systems management
Concerns	- How are the component's computational elements mapped onto the available hardware? - How do these elements communicate with each other at run-time? - How is the fulfillment of non-functional requirements such as reliability, availability, scalability, etc. supported?
Viewpoint language	Modules and their distribution over nodes, communication associations between modules (UML deployment diagrams)
Viewpoint source	Similar to Kruchten's physical viewpoint [Kruc95]; also known as engineering viewpoint

Table 7: Characteristics of the physical viewpoint

The way the Context Component's modules are mapped to nodes in an information logistic application is dependent on various factors such as the number of users, the number of context sensors employed, the variety of context data supplied by these sensors, the number of

rules for context data combination and derivation, and so on. A large-scale application with a high amount of computations usually requires a greater degree of software distribution, because the computational load needs to be shared among several nodes. The mapping of software onto hardware is furthermore influenced by the non-functional requirements made onto a specific application, for example the maximum tolerable down time or response time. Thus, the exact mapping of software elements onto hardware is application-specific. In addition, the physical configurations used for development and testing usually differ from those employed during the operation of an application. In order to take into account this wide variety of requirements and conditions the physical viewpoint provides a number of standard configurations. Each of these configurations is targeted at a specific application scale and provides a blueprint for the mapping of modules to nodes under given circumstances. The physical viewpoint furthermore ensures that the mapping of the Context Component's modules to nodes is flexible and can be adjusted to changing requirements and conditions.

In small or medium-sized applications and during development and test the Context Component's modules are distributed over a small number of nodes. If very few context sensors are employed, all modules may even execute on a single node. However, this configuration quickly reaches its limits, and its usage therefore needs to be considered very carefully. Sensor adaptors have to access context sensors by means of different hardware or software interfaces. They may also have to authenticate themselves with a context sensor and may have to access additional data sources in order to transform the data received from context sensors. Due to the amount and the heterogeneity of resources required by sensor adaptors – such as protocols, drivers, certificates, pieces of software, etc. – it is neither feasible nor desirable to allocate all sensor adaptors to a single node in most cases. Therefore, in small or medium-sized applications various instances of the `SensorAdaptor` module are created and deployed on several nodes. Each of these instances is allocated to an individual node. The number of context sensors a `SensorAdaptor` module instance interacts with should be restricted to a manageable amount; it is recommended to limit this number to a maximum of two in order to be able to adjust sensor adaptors to context sensor changes and to avoid bottlenecks in the usage of resources.

In Figure 69 a standard configuration for a fault-tolerant medium-sized application is shown. Except from the `SensorAdaptor` module all modules of the Context Component execute on a single node the type of which is `ComponentServer`. As explained above, several instances of the `SensorAdaptor` module exist, each of which is allocated to an individual `SensorServer` node. Since sensor adaptors create `ContextElement` objects, the `Context` module is replicated on each `SensorServer` node in order to reduce communication overhead. A further instance of this module also executes on the `ComponentServer` node where it is accessed by the `ContextBuilder` and `VirtualSensor` modules responsible for the creation of `Context` and `ContextElement` objects. Since the interfaces of the structures employed to represent contexts and to gather and supply context data as well as their usage have already been explained in detail in the previous chapters, we do not describe them once again in this section and refer the reader to Chapter 4 and Chapter 5 instead. In Figure 69 a second `ComponentServer` node instance is shown. In case the initial node fails this node serves as a back-up which ensures the operability of the Context Component by executing the module instances allocated to this type of node on the primary node's behalf. By means of this back-up node the stability and reliability of the Context Component is thus increased to a great

extent. For `SensorServer` nodes, too, the usage of back-up nodes is advisable, in particular for those nodes `SensorAdaptor` modules for frequently used context sensors execute on. These back-up nodes are not shown in the figure for reasons of clarity.

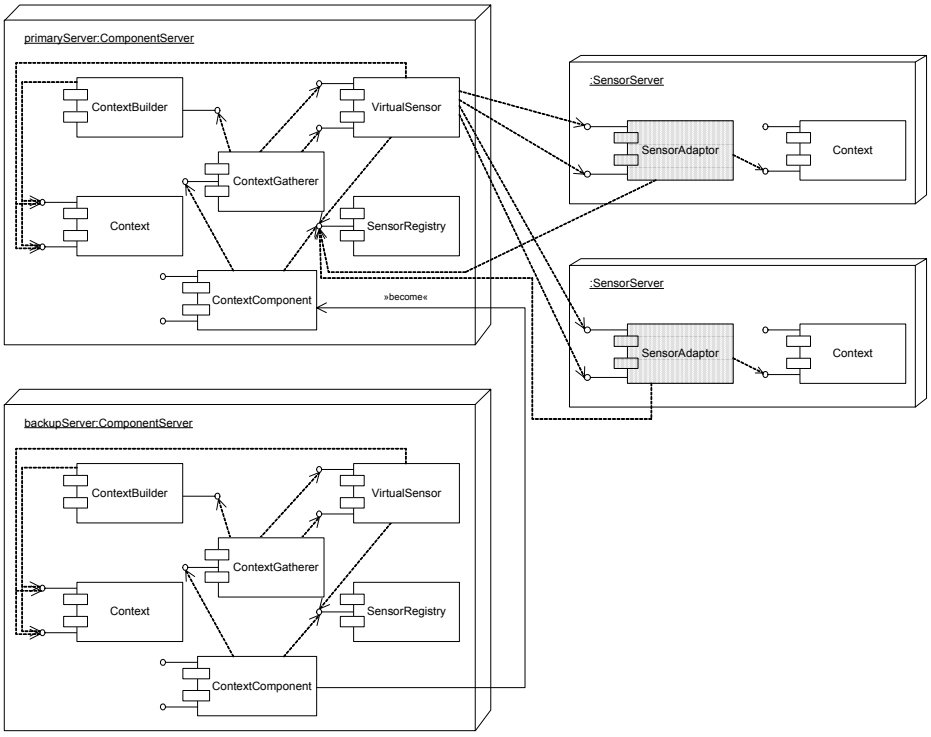

Figure 69: Physical configuration for medium-sized applications

In large-scale applications the allocation of a great number of modules to a single `ComponentServer` node is likely to lead to an overburdening of this node's computational power. Apart from sensor adaptors several other modules of the Context Component may need to perform complex calculations as well. This includes the evaluation and execution of rules for context data combination and derivation, the discovery of appropriate sensor adaptors with regard to a given specification, or the filtering and aggregation of context data. The execution of these tasks may frequently consume a large amount of computational resources; in particular the memory usage of these tasks may exceed the capacity of a single processing resource. As a result, if in a large-scale application the corresponding modules are all executed on the same node, the performance of the Context Component and thus of the entire information logistic application deteriorates heavily.

Large-scale applications therefore need to be supported by a different physical configuration than that employed for smaller information logistic systems and for development and test. Since the software elements contained in the `VirtualSensor`, `SensorRegistry`, and `ContextBuilder` modules all perform complex tasks, each of these modules is allocated to a separate node in an application with a heavy load. In addition, the `ContextComponent` and `ContextGatherer` modules are put together and are both executed on a dedicated

node as well. Availability and fault-tolerance are again achieved by providing back-up nodes for each of the primary node instances. The physical configuration for large-scale applications is depicted in Figure 70; the back-up nodes that need to exist are not shown in this figure in order to make it clearer.

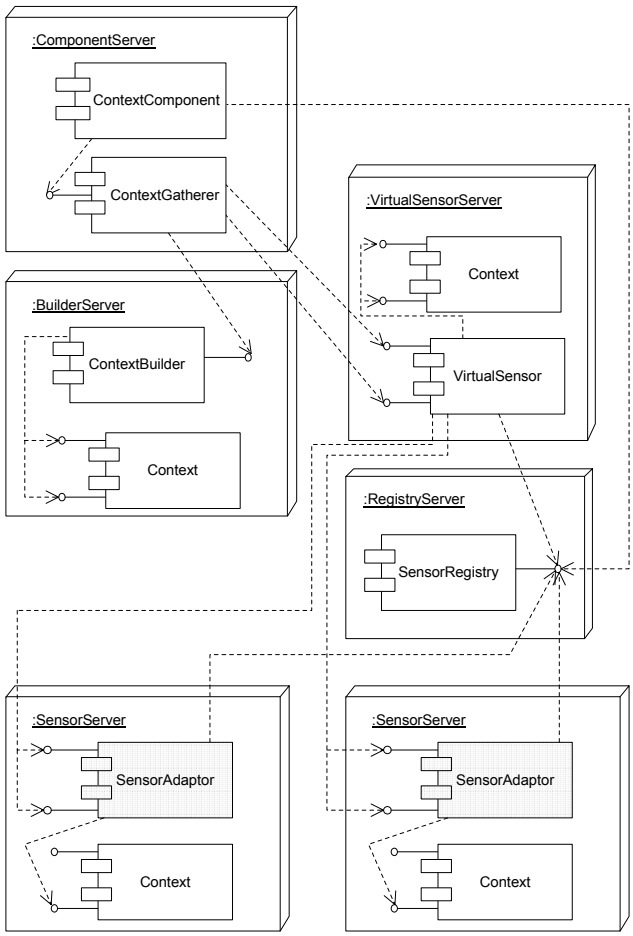

Figure 70: Physical configuration for large-scale applications

Variations of the physical configurations described above are also possible. Particular modules of the Context Component may be subject to heavy load in an application – for example if a large number of Context SRML rules exists that have to be interpreted and executed by virtual sensors –, while the complexity of other modules' computations is limited. This may, for instance, be the case when the filtering carried out by context builders is simple, because the context data provided by context sensors are mostly disjoint or when the discovery of sensor adaptors done by the service registry can be restricted to few comparisons between the required and the provided capabilities of sensor adaptors. In these cases those modules that make high demands on the available processing resources are each allocated to individual nodes, while the other modules reside together on a different shared node.

We have already mentioned that the information logistics framework provides mechanisms for the decoupling of a component's core functionality from the communication protocol employed in an application and from the process of obtaining references to other, potentially remote, components and services. In conjunction with the external viewpoint and the logical viewpoint we have explained how these mechanisms are applied to the Context Component as a whole. Yet, they are also made use of whenever the modules a component is composed of are distributed over several nodes. The impacts the physical distribution of the Context Component's modules over nodes has on the source code are therefore minimal, and the component's physical configuration can thus be flexibly adjusted to changes.

6.2.6 Interrelation of the viewpoints

The viewpoints we have described in the previous sections are not completely independent of each other. Since they describe different aspects of the same subject, the architecture of the Context Component, they may overlap to a certain extent. However, the individual viewpoints have to be consistent and must not contradict themselves. In this section we describe how the viewpoints of the Context Component are interrelated and in what way elements of one viewpoint are connected to elements in other viewpoints.

As we have already mentioned, the external viewpoint and the logical viewpoint are closely related to each other by both addressing the functional requirements made onto the Context Component. While the external viewpoint describes the way the Context Component's services are made available to clients, the logical viewpoint provides an object model for the computational elements required to implement these services. As a result, the interfaces, methods, and method parameters defined in the external viewpoint need to correspond to the logical viewpoint's structures and vice versa. These two viewpoints therefore cannot be developed fully independently of one another, but rather require knowledge of the structures described in the respective other viewpoint.

The dynamic viewpoint deals with aspects that are not taken into account in the external and the logical viewpoint such as the behaviour of objects or concurrency of processes and therefore extends the Context Component's architecture to the consideration of some non-functional requirements as well. For this purpose the dynamic viewpoint is required to possess a certain amount of knowledge of the static structures the Context Component consists of. Yet, when developing this viewpoint two different approaches can be pursued. On the one hand, based upon the structures identified in the external and the logical viewpoint, the behaviour of these structures, the processes that are executed, and the message exchanges between them can be defined. This approach implies that the dynamic viewpoint is developed after the external and the logical viewpoint. On the other hand, the dynamic viewpoint may take the component's services described by the external viewpoint and thus the requests made by clients as its starting point and define corresponding processes and interactions that have to take place in order to fulfill these requests. In this case the object model for the Context Component's structures – i.e. the logical viewpoint – is developed in detail after the dynamic viewpoint and has to correspond to the dynamic viewpoint's contents.

Each of the structures defined in the external and the logical viewpoint belongs to exactly one package. Packages are defined in the structural viewpoint and group related classes and interfaces. The definition of packages has to be carried out in consideration of additional aspects such as the expected amount of a package's programme code, the reuse of structures, the size and organization of development teams, or configuration management. These aspects also have to be considered when packages are grouped into modules representing a particular part of the component's overall functionality. Therefore, the structural viewpoint is based on the external and the logical viewpoint, but addresses different concerns by examining the Context Component's architecture from a more coarse-grained point of view and describing the organization and grouping of the computational elements identified before.

The physical viewpoint maps the modules defined in the structural viewpoint to the available processing resources and provides physical configurations for different application environments and sizes. This viewpoint thus depends on the definitions made by the structural viewpoint, but extends the other viewpoints' descriptions by explicitly addressing aspects of software distribution.

6.3 Summary and Assessment

In this chapter the reference architecture for the Context Component has been presented. This architecture incorporates and enhances the models and mechanisms for the representation and gathering of context described previously. We conclude our explanations concerning the Context Component's architecture in this section by giving a brief summary of the architecture's main features and comparing our results with the existing approaches introduced in Chapter 3.

The Context Component has to fit into and cooperate with other components and services information logistic applications are composed of. Thus, the Context Component's architecture must comply with the mechanisms and structures defined in the information logistics framework to ensure interoperability. In the first section of this chapter we have therefore introduced the information logistics framework and the basic topics it deals with in order to illustrate the technical environment the Context Component has to be integrated into.

The design of the Context Component's architecture has to take different aspects into consideration. These include concepts for the design, implementation, operation, and maintenance of the component, the integration of external data sources and services into it, and the integration of the component itself into information logistic applications and into existing environments. In order to reduce the complexity resulting from this variety of concerns that have to be dealt with the Context Component's architecture is examined from different viewpoints. Each viewpoint describes a particular part of the architecture and addresses a specific subset of the entirety of concerns. The concept of viewpoints has proven reasonable in various approaches. We have outlined some of the most prominent suggested sets of viewpoints and have argued that the variety of existing proposals hampers the selection of those viewpoints that are relevant in a given situation. Therefore, our approach has been to adopt the IEEE 1471-2000 Rec-

ommended Practice for Architectural Description for Software-Intensive Systems. In contrast to other approaches this standard does not prescribe a fixed set of viewpoints, but instead specifies requirements each viewpoint has to fulfill.

Correspondingly, the Context Component's architecture has been described by means of five viewpoints each of which complies to IEEE 1471's specification concerning the definition of viewpoints. The viewpoints employed to describe the Context Component's architecture have been chosen on the basis of the concerns identified in conjunction with the requirements onto the component's architecture. They comprise

- an external viewpoint describing which services the Context Component makes available to clients and how these services can be accessed and used
- a logical viewpoint providing an object model for the elements that implement the Context Component's services
- a dynamic viewpoint focusing on the processes, interactions, and message exchanges between these elements
- a structural viewpoint dealing with the organization of the Context Component's computational elements
- a physical viewpoint describing the mapping of these computational elements onto processing resources.

These viewpoints together provide a comprehensive description of the Context Component's architecture and thus constitute a reference architecture for this component. The Context Component is newly added to the core components of the information logistics framework, and as a consequence the architecture described in this chapter is binding for any implementation. In addition to providing a reference architecture for the Context Component, we have also developed a standard implementation of it. A prototype application demonstrating the Context Component's use and its benefits is described in the following chapter. For this reason further implementations are expected to be restricted to extending and customizing the Context Component – for example, new sensor adaptors or new coordinate systems for locations may be added – instead of implementing all the structures defined by us all over again. Nevertheless, developing extensions of the Context Component also passes through different phases of the software development process as, for example, those defined in the Rational Unified Process [Kruc00]. Thus, the process of applying and possibly extending the viewpoints described above is an iterative one.

The reference architecture we have designed for the Context Component describes all structures and interfaces the component consists of along with the relationships between them as well as the processes executed within the component. It thus covers all elements that are needed in order to efficiently represent, gather, handle, and supply context data. In addition, unlike the majority of other existing approaches our architecture takes both the modelling and the gathering of context into consideration to the same extent. Moreover, it allows clients to obtain metadata concerning the detectable context elements, their attributes and values. We are not aware of any other system that provides a similar functionality. Context data can be queried both synchronously and asynchronously as a result of which clients are able to access them the way that is most suitable for their needs. In contrast to this, many other approaches

such as Sentient Computing are restricted to supplying context in an event-driven manner. Clients are furthermore enabled to obtain information both about the entire contexts of entities and about specific context elements, attributes, or values. While many other systems, for example those developed in the course of the TEA project or the Context Service, fail to provide adequate means that allow clients to specify which data they require, the architecture of the Context Component comprises sophisticated mechanisms for this purpose. In the SOLAR system clients have to explicitly specify the context sensors data are to be gathered from and the operators that are to be used to augment context data. In contrast to this, specifications concerning required context data in our approach refer to the type or contents of the data rather than to their source. In addition, since our context model is sensor-independent and the mechanisms to acquire and augment sensor data are encapsulated in the Context Component, a separation of concerns is achieved. Other components of information logistic applications therefore do not have to concern themselves with the way context is determined and managed. As a consequence, the architecture of the Context Component we have presented possesses a number of features that are not covered by other proposed architectures and thus overcomes the shortcomings existing approaches suffer from. The architecture furthermore takes into account aspects of transparent distributed communication, availability, scalability, performance, and resource discovery. Accordingly, both the functional and the non-functional requirements made onto the Context Component's architecture are met. In particular, the major benefits resulting from the reference architecture we have developed are as follows:

- Since the Context Component's architecture is described by means of several viewpoints each of which addresses specific concerns of different stakeholders, all concerns are dealt with explicitly and no particular aspect is under-represented or overstressed. In addition, stakeholders can easily find those aspects of the Context Component's architecture they are interested in and do not have to concern themselves with viewpoints of minor relevance to them.
- The Context Component's architecture can be tailored to different application domains and sizes as well as to different environments without difficulty. If need be, other notations or design methods may be used for the individual viewpoints. Thus, the Context Component's architecture is generic and is able to evolve over time and to be adapted to changing conditions and requirements.
- The reference architecture for the Context Component defines sophisticated features for the representation, gathering, management, and supply of context data. Apart from thus supporting the fulfillment of all functional requirements it furthermore facilitates the implementation and customization of the component and allows for a shorter period of development and consequently a reduced time to market for information logistic applications.
- The Context Component's architecture allows for deploying this component in different, heterogeneous environments and enables its utilization as a means of prototyping information logistic applications that address new problems in the area of context-aware computing and as a research testbed.

7 Implementing Context-Aware Information Logistic Applications

In order to prove the validity and usability of the concepts, models, and the architecture presented in this thesis both the Context Component and an information logistic application using it have been implemented. After briefly explaining the implementation of the Context Component the main stress of this chapter is put on the context-aware information logistic application we have developed on the basis of this component. Considering this application as an example the chapter demonstrates the benefits of a context-aware information supply and the usage of the Context Component in connection with the conceptual design, development, and operation of context-aware information logistic applications. The functionality of the application we have developed as well as details concerning its design and implementation are presented. In addition, we point out the application-specific extensions that have been made to the Context Component to fulfill the application's requirements.

7.1 Implementation of the Context Component

The Context Component has been implemented according to the reference architecture presented in Chapter 6. The implementation complies with the information logistics framework by making use of the mechanisms concerning aspects such as component configuration, life cycle management, or component structure the framework defines. Correspondingly, the classes and interfaces belonging to the Context Component have been implemented according to the package structure defined in Section 6.2.4. The programming language made use of is Java, and the communication mechanism employed is Java RMI.

The implementation of the Context Component is directed towards the component's generic parts only. It provides modules that can be reused in any context-aware information logistic application. Since neither the requirements of future applications nor the application-specific extensions that may accordingly become necessary can be foreseen, the implementation of the Context Component does not include application-specific functionality. This functionality is rather to be programmed and integrated during the development of the individual applications using the component. Besides, the information logistics framework stipulates that a distinction between the generic modules and the application-specific modules a component consists of – the latter of which it does not cover – is made. This distinction significantly increases the maintainability of the component software and facilitates the implementation and integration of application-specific programme code.

As defined by the reference architecture for the Context Component, our implementation provides programme code for the component's interfaces `ContextComponent` and `ContextMetadata` as well as for corresponding RMI interfaces. The component is subdivided into a front-end and a back-end, the former implementing the communication protocol-specific component interfaces and containing programme code to obtain references to other components, to the Profile Manager, the Logging Service, the Error Handler, and the internal clock, to be precise. The component's back-end provides the actual functionality of the Context Component and implements the general component interfaces. In addition, the compo-

nent's back-end also implements corresponding dependency interfaces to be provided with references to the abovementioned components and services. The implementation of the Context Component's back-end includes code for the management of the component's life cycle. Moreover, a dependency interface for the Context Component has been coded.

The implementation of the Context Component furthermore comprises programme code for the computational elements specified in the context model. The code complies with the model and thus covers the context elements location, state, reachability, and surroundings. A number of coordinate systems that are expected to be made use of in many applications as well as common dimensions for them have been implemented. Since a complete set of potentially relevant coordinate systems and dimensions cannot be foreseen, further coordinate systems and/or additional dimensions that may be needed by specific applications in the future will have to be implemented on demand. In addition, within the scope of the Context Component's implementation no operation execution services have been integrated into the component. This is due to the fact that operation execution services are application-specific and therefore do not belong to the scope of the context model or the information logistics framework. The implementation of the Context Component rather leaves it up to individual applications to develop the specifically required services if need be and to integrate them into the component.

Similarly, our implementation of context gathering mechanisms restricts itself to the generic parts of the corresponding object models. This means that in particular all interfaces defined in these models and in addition the classes related to the representation of virtual sensors, context gatherers, context gathering results, and specifications concerning contexts and context elements have been implemented. In addition, some common rules for context data combination and derivation have been defined. However, we have not implemented any sensor adaptor classes. Sensor adaptors are specific to both the context sensors employed and the application that makes use of them. Thus, since the implementation of the Context Component covers generic modules only, sensor adaptors have to be implemented together with the information logistic applications requiring them.

The context model, the context gathering techniques, and the reference architecture for the Context Component described previously are very fine-grained and already contain detailed guidelines concerning the implementation of the component. We have gained from this high degree of meticulousness when implementing the component software. The implementation of the Context Component mainly involves the unaltered transformation of the concepts and models into programme code. Where needed, some additions in the form of methods to access and modify the attributes of an object or temporary data structures and the like have been added which, however, do not affect the overall validity of the underlying models. Due to the correspondence between the implementation and the models and architecture we do not consider further details concerning the Context Component's implementation necessary.

7.2 A Context-Aware Information Logistic Portal

This section presents a context-aware information logistic application we have designed and implemented. This application uses the Context Component to optimize information supply with regard to the dimension of context. Its development has been driven by the goal to dem-

onstrate the benefits of a context-aware information supply and the usability of our solutions. In this section we first motivate the development of this application and give an overview of its functionality. We furthermore point out the application's benefit to users and the way it contributes to putting the goal of information logistics into practice. Subsequently, this section describes the application's overall architecture and presents details concerning its design and implementation. In doing so, explanations regarding the way the application uses the Context Component and the application-specific extensions that have been made to it are given.

7.2.1 Application description

The application we have developed is targeted on office environments. The effects of the recent advancements in information and communication technology mentioned in Chapter 1 are particularly evident in this application domain. Office workers have a multitude of different information systems at their disposal and access them by means of complex user interfaces. The way office work is organized has undergone considerable changes. The dividing line between work and spare time has become blurred which is reflected in modern forms of work such as teleworking or flexible working hours. Office workers furthermore are expected to be extremely flexible; many of them work in varying teams, are frequently assigned to different projects or tasks making different demands on their skills, or are in charge of varying customers. In addition, the dynamics of today's market situation require companies to respond to changes very quickly, as a consequence of which office workers becoming increasingly pressed for time. Non-territorial concepts of work as well as a growing mobility of people furthermore characterize the office work of today.

Thus, since office workers are confronted with rapidly changing situations, the ability to access relevant information in a quick and purpose-oriented manner is indispensable. The existing information overflow, however, often makes it difficult for office workers to find the information they need in time. In addition, the information demand of office workers to a high extent depends on context. In particular the location, motion, activity, and reachability of office workers determine the information they need. Consider, for example, a person who is writing a report within the scope of a particular project. In this case documents related to the project such as the project plan, milestone documents, or correspondence with the project partners are of particular importance to her. Similarly, while having a telephone conversation with a customer, an office worker wishes to have access to information relevant to this incident and at the same time does not want to be disturbed by any other incoming messages.

In summary, we have recognized the need for an information supply according to information logistic principles in office environments. This demand has motivated the development of the context-aware information logistic application we present in this section. This application combines the concepts of information logistics – including a consideration of the dimension of context – and portal technology. Portal systems serve to integrate several, potentially heterogeneous data sources and enable users to access information from any of these sources via a uniform interface [Koen00]. In office environments in particular portal technology can contribute to facilitating the access to and the retrieval of information. However, we believe that integrating various data sources in a portal is not sufficient, because a portal alone is not capable of assessing, selecting, and timely supplying exactly those pieces of information users need in

given situations. Therefore, we have developed a context-aware information logistic portal system that allows for a uniform access to different information sources and at the same time provides users with information that is particularly relevant in their current context.

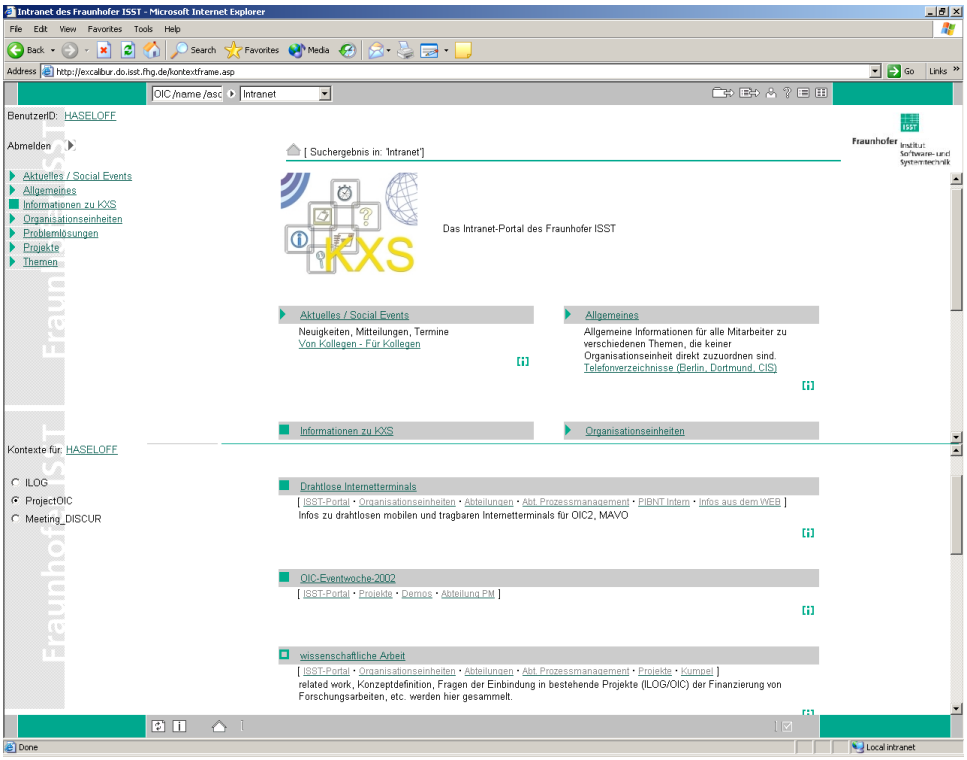

Figure 71: Modified user interface of the KXS

Our context-aware information logistic portal complies with the information logistics framework. Both as a content service and at the access layer of the application the portal system KXS (Knowledge eXchange System) is used. The KXS is a joint development of the Württembergische Versicherung, a German insurance company, and the Fraunhofer ISST. The standard user interface of the KXS is subdivided into a frame showing the available document categories and a main page displaying the contents of the currently selected category or document. In order to enable an unobtrusive context-aware information supply while at the same time ensuring that the KXS' traditional functionality remains available in the usual manner we have modified this layout. As shown in Figure 71, a further division of the system's user interface has been made. In the upper part of the browser window users find the standard contents of the portal. The window part beneath serves to provide information that is selected on the basis of users' contexts. The frame on its left side displays all contexts the user who is logged on to the system may be in. The user's current context is highlighted, and the pieces of information that are particularly relevant in this context are listed on the right. Whenever the context of the user changes, the lower window part of the portal is automatically updated. Our application thus ensures that the provision of information is constantly adapted to users' contexts. In both the upper and the lower window part of the portal users may navigate and open documents as is

usual. This means that in particular our application provides users with the opportunity to manually change their current context by clicking on one of the contexts displayed in the lower part of the browser window. The user interface of our context-aware application was designed with the aim to be unobtrusive and user-friendly. It has been based on the desire to prevent context changes from causing a disruption of the users' current interactions with the portal. For this reason the standard contents of the KXS are always visible in the upper part of the window and are not affected by context changes. In addition, we believe that users should always be given the opportunity to explicitly set their current context themselves. By being able to thus override the information provided by context sensors users are given increased control over the system. As a consequence, feelings of heteronomy that may overcome people when using context-aware applications are minimized.

In contrast to other context-aware applications our portal does not operate on a fixed set of contexts that is the same for all users. According to the target of information logistics, an optimized information supply in line with users' demands, the application we have developed provides users with a means of defining individual contexts. Consequently, our application comprises another user interface that serves the purpose of context definition called Context Editor. By means of the Context Editor users are enabled to specify the contexts that they may be in and that entail specific information demands. In our application the information demands themselves are not explicitly defined by users. Instead, the application derives context-dependent information demands from the context attributes' values users specify in the Context Editor and transforms them into appropriate search queries that can be made to the KXS. We have chosen this simplified approach towards information demand definition and description, because we do not want to burden users with the need to explicitly input too much information. Besides, since our application is mainly concerned with the dimension of context, the other dimensions of information logistics are attached secondary importance.

In our context-aware portal context is gathered by means of three different sensors. The first one is an RFID-based indoor location sensor installed in the rooms of the Fraunhofer ISST's branch in Dortmund as well as in the Fraunhofer Office Innovation Center in Stuttgart. The second context sensor employed consists of the electronic schedules of users that are stored in a Microsoft Exchange Server and accessed by users through Microsoft Outlook client applications. Finally, the third sensor our application uses is the KXS itself which provides information concerning the reachability of users in terms of the computers on which users are logged on to the portal. Thus, in rough outline our context-aware portal monitors the contexts of users and checks whether the determined context of a user matches one of the contexts she has specified using the Context Editor. If this is the case, the application executes the previously generated query on the KXS and updates this system's user interface by displaying the information that in the current context is particularly relevant to the user.

7.2.2 Application architecture

Our application makes use of specific context sensors, content services, and components on the access layer. In addition, since it is primarily concerned with the dimension of context, some core components of information logistic applications are not needed. This subsection illustrates the architecture of the context-aware portal and describes its overall functionality.

The application's architecture which is shown in Figure 72 mainly corresponds to the subsystems and components the information logistics framework defines. On the access layer, however, we do not make use of a variety of communication channels. Instead, the communication with the user is always carried out by means of browser-based applications. These applications comprise the KXS as the output channel supplying users with information and the Context Editor employed by users to input their contexts and – implicitly – their information demands into the application. Furthermore, in our application the Context Component uses three specific context sensors as mentioned above. The Content Broker accesses a single content service only which, too, is based on the KXS. The functionality of this service is described shortly.

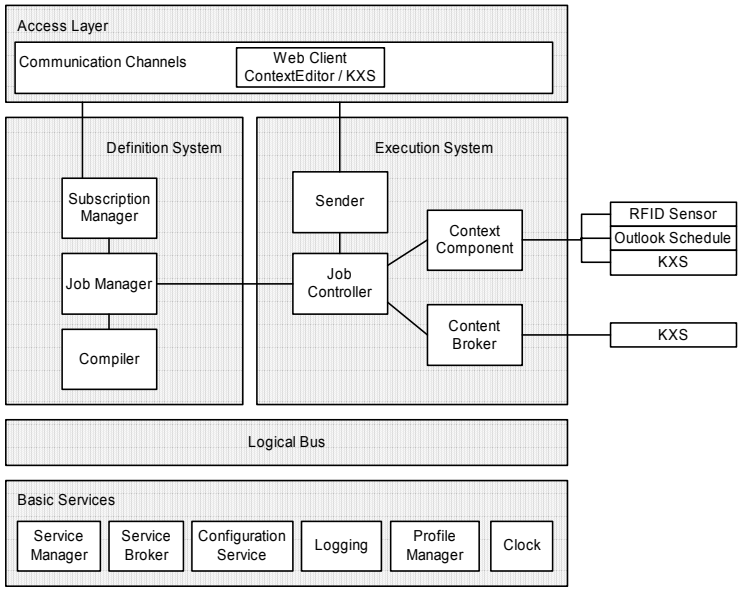

Figure 72: Architecture of the context-aware portal

Since our application aims at demonstrating the validity of our solutions and the benefits of context awareness to information logistics, some simplifications concerning the consideration of dimensions other than context have been made. Our application does not make use of the Timer, because information supply can be carried out at any time and is triggered by context changes only. Furthermore, a history of user interactions with the application is not required. Due to the fact that only a single communication channel and a single content service are made use of the Presentation Producer likewise is not needed. Since information supply depends on the dimension of context only, there is no need for overall optimizations across several dimensions. Thus, our application does not comprise a Decision Component either.

The overall functionality of the context-aware portal is provided in several successive steps. At first, users define their individual possible contexts by means of the Context Editor. On the basis of a context's attribute values specified by a user the Context Editor derives the user's corresponding information demand. Information demands are represented as search queries that can be executed on the KXS. The values of context attributes are mapped to keywords defined in the KXS, and a corresponding search query is created and inserted into the KXS'

database. The Context Editor furthermore creates subscriptions containing the contexts and corresponding information demands of users and passes them to the Subscription Manager. This component assigns identifiers to these subscriptions and persistently stores them in an Oracle 8i database. After that it forwards the subscriptions to the Job Manager. The Job Manager charges the Job Compiler with the transformation of subscriptions into jobs and assigns identifiers to the jobs it is returned.

The Job Manager then passes these jobs to the Job Controller which is responsible for their execution. For this purpose the Job Controller makes asynchronous requests to the Context Component. In each request the context defined by a user is passed to the Context Component as a `contextSpecification` parameter. The Context Component obtains data from its context sensors and examines them for the fulfillment of the specifications it has been supplied with. When a user's context matches a specification the Context Component fires a corresponding event to the Job Controller. Upon receipt of this event the Job Controller calls upon the Content Broker to be returned the information that is to be supplied in this context. For this purpose the context is passed to the Content Broker. The Content Broker itself again forwards this request and its parameter to the content service it is associated with. In our application the information that is to be supplied to users is represented by URLs each pointing to a search query defined by the Context Editor. The content service accesses the KXS' database with the context data passed to it to obtain the corresponding search query. After that it generates and returns a URL that can be used by the KXS to locate and execute the search query and display the resulting pieces of information. This URL is passed from the Content Broker to the Job Controller which queries the Context Component synchronously for the reachability of the user the information is to be supplied to. The Job Controller then forwards both the URL and the reachability information to the Sender. The Sender communicates via HTTP with an application installed at the recipient's computer and transmits the URL to it. The receiving application is a container application within which the KXS is executed. Upon receipt of the URL this container application induces the KXS to reload its lower browser window and to display the results of the search query the URL points to.

7.2.3 Application design and implementation

The development of any context-aware application is to a large extent based on the requirements of the application's intended users regarding the dimension of context. A major goal of analysis and requirements engineering activities thus is to find out which context elements and which attributes and values of them are relevant to users and accordingly have to be taken into account by the application that is to be developed. In addition, possible context sensors that are able to determine these context data have to be identified. It is furthermore important to gain a detailed understanding of the way context affects the information demands of users.

The development of our context-aware information logistic portal has taken place within the scope of the information logistics project of the Fraunhofer ISST and has not been carried out by order of a specific customer. As a consequence, the analysis of potential users' requirements has been based on discussions with business partners and researchers, scenarios for information logistic applications, and our own experience with office work. During the early stages of development we intended to demonstrate the usage of a variety of complex contex-

tual information in our application in order to cover the features of the Context Component as extensively as possible. However, our first prototypes led us to the conclusion that this approach is not feasible. We have had to recognize that the multitude and complexity of contextual information the Context Component is capable of handling overwhelms office workers when defining contexts. A very extensive set of possible context attributes and values for them decreases the user-friendliness of context-aware applications. In addition, the feedback given to us by several interested parties from the industry has shown that in most real-world scenarios only a subset of the contextual information the Context Component can deal with is needed. Therefore, the context-aware portal we have developed makes use of a limited set of context elements, context attributes, and values for them only. However, we are convinced that our application is sufficiently complex and mature to prove the validity of our solutions.

One of the main characteristics of today's office work are the varying activities office workers carry out in the course of the day. These activities are frequently related to various customers or suppliers, projects, files, goods, or the like and accordingly require an access to different pieces of information. In addition, the increased mobility of office workers leads to frequent changes of locality, both within and outside the office. Calls on customers, visits to trade fairs, or varying workplaces within the office are common and likewise affect the information demands of office workers. Since the context elements of location and state, in particular activity, therefore are of overriding importance to the domain of office environments, it is these elements the context-aware information logistic portal makes use of. In addition, the application also considers reachability in terms of the computer on which a user is logged on to the KXS when information is to be supplied to her. This context element is taken into account in order to ensure that information is delivered to the appropriate user's KXS client. Reachability information therefore is only made use of internally to carry out information supply; it is not specified by users when defining contexts. The portal furthermore allows to characterize the activities of office workers with attributes concerning the project in the context of which an activity takes place and the persons the activity is related to or carried out with.

In the following we discuss the design and implementation of those parts of our context-aware application that are concerned with the dimension of context. This includes the Context Editor as well as the Context Component. In conjunction with the Context Component we pay special attention to the way the context model is used and to the context sensors employed.

7.2.3.1 Context Editor

As mentioned when describing the overall functionality of our context-aware portal, the Context Editor is an application that serves users to individually define the contexts they may be in and that derives users' information demands from these contexts. In addition, the Context Editor also manages the profile data of users. Before being able to log on to the Context Editor for the first time users are required to provide their forename and surname as well as a login name and password. Furthermore, in order to store the additional data required for the transformation of sensor data into context data at a centralized place the Context Editor also prompts users for the login names with which they access the KXS and the Microsoft Exchange Server, respectively, as well as for the identifier of their RFID tag. These profile data are stored in an Oracle 8i database and are accessed by sensor adaptors when transforming sensor data as described later on.

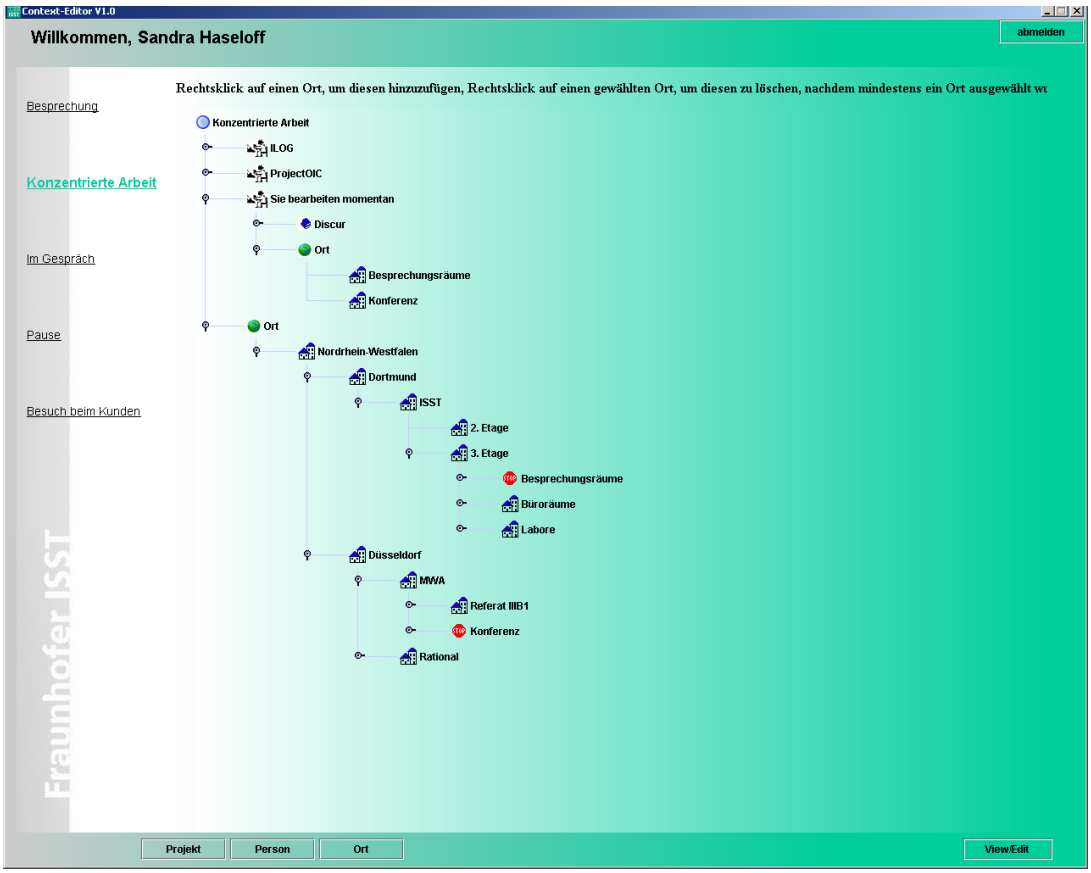

Figure 73: Definition of contexts in the Context Editor

The Context Editor allows users to define contexts that consist of the context elements of location and state/activity. To ensure that only contexts that can be detected by the available sensors are specified the Context Editor accesses the `ContextMetadata` interface of the Context Component. By means of this interface's method the Context Editor is provided with the supported context elements as well as with their possible attributes and values. The design of the Context Editor's user interface aims at enabling users to define contexts in an easy and user-friendly manner. Therefore, we have decided to use the context element of state/activity as a starting point for context definitions. The specification of a context always begins with selecting one of five possible activities. The activities made use of in our application are »being in a meeting«, »working with concentration«, »conversing«, »having a break«, and »calling on a customer«. After selecting an activity users may define attributes that further characterize the activity and locations the activity may take place at as exemplarily shown in Figure 73 for the activity of »working with concentration«. On the basis of the selections made by a user the Context Editor creates one or more `ContextElementSpecification` instances and `ContextSpecification` objects containing them. In doing so, the Context Editor also examines the data the user has selected for validity with respect to existing context constraints. We have defined a context constraint which stipulates that the activity of »calling on a customer« is required to take place at the respective customer's place of business.

The available attributes of activities are the project within the scope of which an activity is performed and the persons involved in the activity. Since during concentrated work the identities of present people usually are of no importance, persons cannot be associated with this activity. Analogously, the activity »having a break« does not possess a project attribute as this activity is generally not related to a project.

As can be seen in the above figure, contexts are displayed in a tree structure. A separate tree is constructed for each of the five possible activities and is shown when the user selects the corresponding activity. Accordingly, the root of each tree is an activity. Its children are the contexts defined by the current user that contain this activity. They are displayed in a compressed manner with their names only; the user may, however, browse through the details of these contexts by means of the controls shown on the left of the contexts' names. Furthermore, the last child of each tree serves to define a new context containing the respective activity. When clicking on this node the user is first shown the available values for one of the activity's attributes she may choose from. In the example shown in Figure 73 this is the project attribute. After selecting an attribute value the user may in the same manner define the values of other activity attributes and the locations the activity may take place at. Each chosen value for an activity attribute or a location is inserted into the tree as a new child of the context that is being defined. When the definition of a new context is completed, the Context Editor prompts the user for a context name. The Context Editor furthermore allows to delete previously specified contexts. In addition, by means of the buttons shown at the bottom of the user interface users are enabled to view the available context elements, their attributes and values.

The Context Editor derives users' information demands from the contexts they define. Information demands are represented as search queries that can be made to the KXS. The Context Editor thus maps the attribute values of a context into keywords defined in the KXS and generates a search query that conforms to the KXS' query format. Subsequently, the search query is inserted into the KXS' database. For this purpose the Context Editor connects to the KXS using HTTP and calls an ASP script which is passed the query, the name of the user to which the query belongs, and the name of the context the query refers to. This script carries out the actual insertion of these data into the KXS' database. After the data have been inserted, the newly defined context is added to the user's contexts displayed in the user interface of the KXS. The Context Editor furthermore creates a new `Subscription` instance and inserts the abovementioned `ContextSpecification` objects as well as the identifier of the user into it. After that the subscription is passed to the Subscription Manager.

7.2.3.2 Context Component

In order to represent, gather, manage, and supply context our application uses the implementation of the Context Component. However, since this implementation does not cover application-specific data and functionalities, some extensions to the component have been necessary. This section describes these application-specific extensions as well as the way the Context Component – in particular the context model and context gathering techniques it provides – is employed in our context-aware portal.

Context model

Apart from the basic `Context` class and the `ContextElement` interface our application also makes use of computational elements defined in conjunction with location, reachability, and the state element of activity. In our application locations are always grouped into location sets. The coordinates of the available locations refer to three different coordinate systems. The first one is a geographical coordinate system used to represent addresses. It currently defines the dimensions of name, city, and federal state. Another coordinate system serves to represent a building's floors and defines a single dimension, the floor number. The third coordinate system our application makes use of is a room coordinate system with the associated dimensions name, type, and number. Both the geographical coordinate system and the room coordinate system are already included in the existing implementation of the Context Component. In contrast, the floor coordinate system has been newly implemented and added to the Context Component. The origin of the geographical coordinate system is the system boundary, while both the floor and the room coordinate systems' origins are non-atomic locations. As regards the floor coordinate system its origin is an address. A room coordinate system's origin may be either an address or a floor. Since our application covers a limited and pre-defined set of locations, all dimensions are associated with a validity condition specifying the permissible values for the respective dimension.

The origins of the coordinate systems described above already indicate that the locations employed in our application may be part of a hierarchical structure. Rooms are spatially contained in floors or addresses, the latter of which may also contain floors. Thus, the root node of any location's structure is an address, while floors may be contained in a location graph as either nodes or leaves. Rooms in turn are always leaves in our application; consequently, they are atomic locations. Our application does not make use of the location model's ability to explicitly represent prepositions. Due to the fact that neither relations of users to locations other than their presence at a location nor distances are taken into account in the context-aware portal users are always considered to be at an address, on a floor, and in a room. For the same reason our application does not need to determine the proximity of locations and does not have to calculate whether two locations overlap. Transformation operations are not necessary either, because all possible results of a location's transformation are already contained in the structure of this location. The operations of locations we do make use of are therefore those that serve to determine equality and containment of locations.

The context-aware portal considers the state element of activity only. The existing implementation of the Context Component provides all computational elements required in order to represent the five possible activities our application covers as well as the attributes they can be associated with. Both the project and the persons attributes are specific attributes as they cannot be assigned to all activities. In addition, the attribute values that are permissible in our application are represented by validity conditions.

The reachability information that is relevant in the context-aware portal is restricted to data serving to identify the computer on which a user has logged on to the KXS. Consequently, very few reachability attributes and addresses defined in the implemented instance of the reachability model are being used. Our application requires a single communication medium only. It consists of a device the type of which is »Personal Computer«, the HTTP communica-

tion protocol, and one specific application, the KXS, or, more precisely, the container application in which the KXS is running as described in Section 7.2.2. The addresses by means of which connections can be established are represented by two objects of the `AddressValue` class. One of these instances captures the IP address of the computer on which the user is working with the KXS and on which the container application is consequently running as well. The other one represents the port number the container application is connected to. This reachability information is sufficient for the Sender component of our application to establish a connection to users' computers and deliver the content that is to be provided to them.

As already mentioned, another feature of the context model our application uses is the model's ability to capture the interdependence among context elements by means of context constraints. The context constraint we have defined requires that the activity of »calling on a customer« is always carried out at the respective customer's place of business. The determinant of its `ConstraintEntry` instance is composed of two context element's attribute values, the name of the activity and the value of its persons attribute, to each of which the operator *equals* is attached. The two values are interconnected by a conjunction. Entities are not specified in the determinant as the constraint applies to all users of the context-aware portal. The corresponding implication consists of the value a location's coordinate is required to possess and the dimension it is to refer to. By means of variables we have specified that the coordinate's value and the type and name of its dimension have to be equal to the name, type, and value of the persons attribute contained in the determinant.

RFID-based location sensor

One of the sensors employed to gather context is an RFID-based indoor location sensor developed by Wavetrend Technologies Ltd. The sensor system comprises mobile Link-It™ tags that may be worn by people or attached to objects and stationary RFID beacons. The beacons, in our installation L-RX 200 readers, receive, decode, and validate data from the mobile tags and output relevant data onto a network of readers [Wave01]. As a special node in this network a centralized server maintains the relation of tags and beacons, i.e. the mobile tags' locations. Via the RS-232 port of this server the sensor data in the form of (beacon identifier, tag identifier) pairs can be obtained by external components.

In order to integrate the Wavetrend location sensor into the Context Component we have implemented a sensor adaptor that can be queried both synchronously and asynchronously. This adaptor, however, does not directly access the abovementioned central server. Since the Wavetrend sensor system had already been made use of within the scope of a different project of the Fraunhofer ISST, a component responsible for obtaining data from the server and converting them into (room identifier, tag identifier) pairs had already been implemented. Thus, as depicted in Figure 74, the corresponding sensor adaptor of the Context Component accesses this wrapper component called Location Tracker Server. The Location Tracker Server defines a callback interface its clients have to implement. By means of this interface's methods clients receive a notification whenever the location of a tag changes. In addition, when registering themselves as clients with the Location Tracker Server, they get supplied with the current locations of all known tags. Since the Location Tracker Server can only be queried asynchronously, the sensor adaptor needs to store the sensor data it receives. We have expected the synchronous interface of the sensor adaptor to be accessed infrequently. This is due to the fact that in

our application the Context Component itself is only queried asynchronously for the context element of location as described previously. Synchronous requests are merely carried out by virtual sensors to determine whether the data received in response to asynchronous requests are still valid as we explain in greater detail later on. Thus, based on the assumption that many notifications the sensor adaptor receives from the Location Tracker Server will not lead to the fulfillment of a request made to itself and will not be queried synchronously by a client, respectively, we have opted for the storage of sensor data, i.e. of data that have not yet been transformed. To store sensor data again the Oracle database we have already mentioned is made use of.

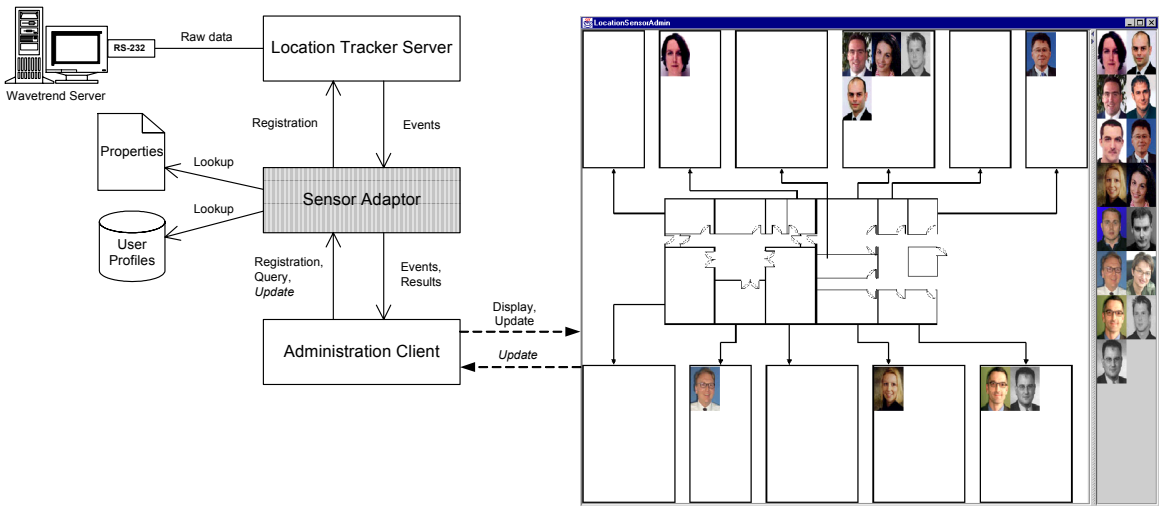

Figure 74: Components for the integration of the RFID location sensor

The sensor adaptor needs to transform the room identifiers it receives from the Location Tracker Server into locations according to the location model provided by the Context Component. In addition, the tag identifiers also have to be converted into entity identifiers used by the information logistic application. For the purpose of sensor data transformation the sensor adaptor for the Wavetrend sensor thus accesses both the user profile data captured and stored by the Context Editor and a properties file. In our context-aware portal users are the only type of entity the context of which is dealt with. Since users are required to input the identifier of the RFID tag they wear into the Context Editor which stores it – along with the user identifier employed within the information logistic application – in a database, the transformation of entity identifiers is carried out by means of a simple database query. Furthermore, in order to transform room identifiers into appropriate locations a properties file has been created. This file contains entries that map the identifier of each room to the room's coordinates referring to the room coordinate system. In addition, for each room information about the location it is contained in – either a floor or an address – is managed in this file. There are entries for each of these parent locations in the file as well as a consequence of which the sensor adaptor is enabled to construct the entire structure the locations are associated with. On the basis of the Wavetrend system's characteristics as well as the data contained

in the properties file and the user profiles the sensor adaptor constructs a `ServiceDescription` object specifying its capabilities. This object is passed to the service registry the sensor adaptor registers itself with.

For demonstration purposes our application also comprises a so-called Administration Client implemented on top of the sensor adaptor for the Wavetrend system. This client uses the data it obtains from the sensor adaptor to display a map showing the current whereabouts of users in the building of the Fraunhofer ISST. In addition, in order to easily demonstrate the effects of location changes this map can also be used as a simulator for the Wavetrend system. The faces of users can be dragged and dropped into rooms, thus causing an event indicating the location change to be sent to the sensor adaptor. As a result, by means of the Administration Client the functionality of the context-aware portal becomes more comprehensible.

Schedules-based location and activity sensor

The second context sensor our application makes use of comprises the electronic schedules of users. In the context-aware portal these schedules are stored in a Microsoft Exchange Server and are accessed by users by means of Microsoft Outlook clients. The schedules of users contain information about the context elements of both location and activity. Correspondingly, we have implemented two sensor adaptors to integrate this context sensor into the Context Component. For each of the two context elements the schedules-based sensor is able to provide a sensor adaptor that can be queried both synchronously and asynchronously exists.

In Figure 75 the components required to integrate the schedules-based context sensor into the Context Component are shown. The sensor adaptors are provided with sensor data by a so-called MAPI Proxy. This component is responsible for obtaining appointment data from the Exchange Server. It is written in the C++ language and uses the Messaging Application Programming Interface (MAPI) and the Collaboration Data Objects (CDOs) library to access the Exchange Server. The MAPI Proxy makes the appointment data available via an event-based interface and can be communicated with by means of sockets. When registering themselves as listeners with the MAPI Proxy, the sensor adaptors are provided with all existing appointments of the context-aware portal's users. Afterwards the MAPI Proxy notifies them about changes made to existing appointments as well as about added or removed appointments. Since the appointment data are persistently stored in the Exchange Server, there is no need for the sensor adaptors to make the data they receive from the MAPI Proxy persistent once more. Instead, while the sensor adaptors are running, the appointment data they receive are kept in memory. We are aware of the fact that storing a possibly large number of appointments in memory does not scale well to many users. Therefore, a planned enhancement involves the implementation of a means to synchronously query the MAPI Proxy.

The most straightforward way for the MAPI Proxy to obtain the appointment data of all users of the context-aware portal would be to directly access the users' calendars stored on the Exchange Server. This, however, is not permitted. The version 5.5 of the Exchange Server installed at the Fraunhofer ISST and its security settings only allow the owner of a calendar to access the data stored in it via the MAPI. Therefore, it has been necessary to create a dedicated user account for the MAPI Proxy on the Exchange Server. The appointments of all users of the context-aware portal are replicated into the calendar of this account. For this purpose we have

implemented an application written in Visual Basic Script that is executed in an Outlook client under the account of the MAPI Proxy. This application accesses the other users' Outlook clients, reads the existing appointments from them, and copies the appointments into the MAPI Proxy's calendar. It furthermore registers itself as a listener with the Exchange Server to be notified about changes made to the other users' calendars after the initial copying process. These changes are again replicated into the MAPI Proxy's calendar for which the MAPI Proxy component itself is registered as a listener with the Exchange Server. As a consequence, the calendar of the MAPI Proxy always reflects the current status of all users' appointments.

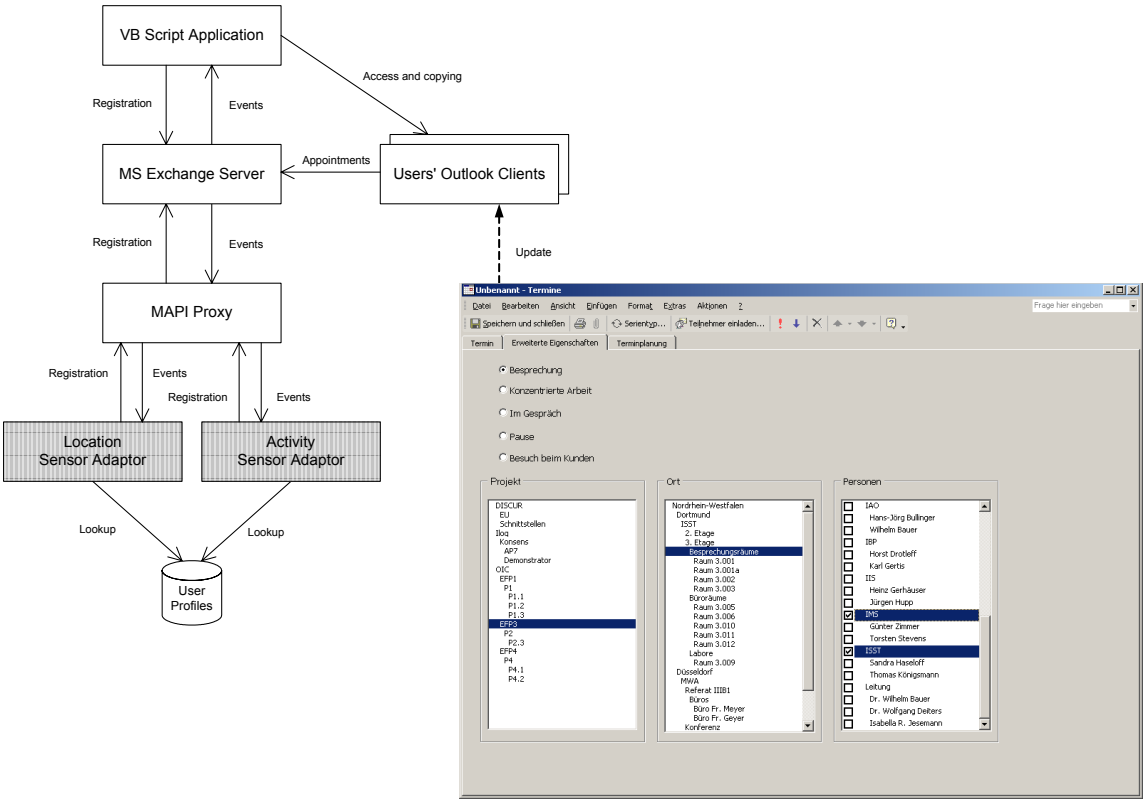

Figure 75: Components for the integration of the schedules-based sensor

We have already pointed out in conjunction with context gathering that electronic schedules usually allow for appointments to be described by arbitrary text. To ensure that contexts defined in the Context Editor can be compared to contexts determined by the schedules-based sensor restrictions concerning the permissible format of appointments are necessary. Therefore, we have designed and implemented a special Outlook form by means of which appointments can be defined. This form which is also shown in Figure 75 allows users to specify the activity carried out during an appointment, the location the appointment takes place at, the appointment's attendees, and the project to which the appointment is related. The form thus ensures that the sensor adaptors are provided with meaningful data on the basis of which `ContextElement` objects can easily be created. The coordinates of the location a user has selected as well as those of its parent location and the names of the selected activity and

its attributes are stored in the appointment. For this reason the only transformation the sensor adaptors for the schedules-based sensor have to carry out is converting the names of the user accounts managed by the Exchange Server into identifiers employed within the information logistic application. In the same manner as described in conjunction with the RFID-based location sensor this is done by means of the user profile database.

As far as the integration of the schedules-based context sensor is concerned, the functionality that is required by all sensor adaptors has been implemented in separate classes both sensor adaptors access. Furthermore, each adaptor for this sensor also constructs a `ServiceDescription` object specifying its capabilities and passes this object to the service registry when registering itself with it. It is worth mentioning that in contrast to the sensor adaptor for the RFID-based sensor the adaptors responsible for integrating the electronic schedules of users are able to provide information about the context data's expected period of validity. In addition, the sensor adaptor that supplies location data supports several formats of this context element as the locations appointments take place at may be either addresses, floors, or rooms. Yet, the quality characteristic of reliability associated with this sensor adaptor has been assigned a value that is lower than that the adaptor for the RFID sensor possesses. This is due to the fact that people occasionally deviate from the plans they had made, while errors in the transmission and decoding of RFID signals occur less frequently.

KXS-based reachability sensor

The KXS itself also acts as a context sensor by making information concerning the reachability of users available to the Context Component. In the context-aware portal the context element of reachability – unlike location and activity – does not have an impact on users' information demands. Its usage serves the purpose of ensuring that information is supplied to the right person. Thus, the gathering and processing of reachability information are indiscernible to users as they only affect the context-aware portal's internal operation. When an information supply is to be carried out, the application has to determine on which computer the recipient is logged on to the KXS. This is done by means of a synchronous request the Job Controller makes to the Context Component. Consequently, for the KXS-based context sensor a sensor adaptor supporting synchronous queries only is needed.

The KXS-based context sensor mainly consists of a script written in the Perl language which is executed on the KXS server. The script is responsible for maintaining a list of all users that are logged on to the system along with the IP addresses of the computers with which they have connected to the KXS server. It can be queried for this information by means of HTTP requests. Thus, when the reachability of a user is to be determined, the sensor adaptor makes a request to the script on the KXS server containing the appropriate user name as a parameter. The script's response provides the sensor adaptor with the IP address of the computer the user is logged on to. Consequently, the user identifiers employed within the context-aware portal have to be transformed into the corresponding KXS user names before the context sensor can be queried. Again this transformation takes place on the basis of the user profile database maintained by the Context Editor. In addition to the IP addresses of users' computers, the number of the port the container application for the KXS is connected to also has to be deter-

mined. This is done by accessing a properties file which contains entries specifying the ports that are allocated to this application on the computers of the individual users. The components required for the integration of the KXS-based context sensor are illustrated in Figure 76.

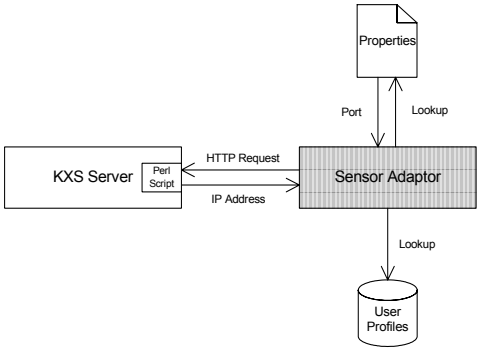

Figure 76: Components for the integration of the KXS-based sensor

The sensor adaptor for the KXS-based sensor transforms the data gathered from the KXS server and the properties file into objects according to the model for the context element of reachability. Since in our application this sensor adaptor is always queried for reachability data with reference to the current point in time and since in addition the Perl script on the KXS server is continuously available, there is no need for the adaptor to store the context data it gathers. As usual, this sensor adaptor possesses a service description and registers itself with the service registry managing all sensor adaptors. In contrast to the service descriptions of the adaptors for the other two context sensors made use of in our application the service description of the sensor adaptor for the KXS-based sensor does not contain an enumeration of all context element instances it is able to provide, because this information is not needed in conjunction with synchronous queries.

When describing the overall functionality of the context-aware information logistic portal, we have pointed out that users are enabled to manually change their context at any time. It would suggest itself to implement appropriate sensor adaptors that communicate with the KXS to be informed about such manual context changes. However, we have refrained from pursuing this approach in our application. When a user selects a particular context in the KXS' user interface and thus causes her context to change, the KXS itself ensures that the user interface adapts accordingly and displays the appropriate information that is relevant in the new context. This is due to the fact that the information demands of users are represented as KXS search queries which are inserted into the KXS by the Context Editor. Therefore, all information and functionality required to properly react to manually performed context changes is already available in the KXS itself. Since apart from delivering suitable content our information logistic application does not perform further operations on the basis of context data – such as maintaining a history of users' contexts, for example –, dedicated sensor adaptors responsible for detecting manual context changes would not provide any additional benefit and besides would result in a prolonged response time. Therefore, to improve the usability and performance of the context-aware portal manual context changes are processed exclusively by the KXS without participation of the underlying information logistic application.

Further context gathering mechanisms

Apart from application-specific sensor adaptors the context-aware portal comprises virtual sensors as a means to pre-aggregate and derive context data. There is no need for context data combination, because in our application all data supplied by sensor adaptors refer to the same entity type, to users. When the Context Component is queried asynchronously for a user's context, it is passed a `ContextSpecification` object containing two `ContextElementSpecification` instances. One of these instances specifies the conditions the user's location has to fulfill, while the other one refers to the user's activity. The virtual sensor responsible for serving this request aggregates a `CompoundCondition` instance that is composed of a `LocationCondition` and an `ActivityCondition` object. Each of these objects makes corresponding asynchronous requests to appropriate sensor adaptors. To fulfill the specification passed to the virtual sensor two events, one referring to the user's location and the other one to her activity, have to be received from the sensor adaptors. These two events usually do not arrive simultaneously at the corresponding conditions. Therefore, as soon as either the `LocationCondition` or the `ActivityCondition` object receives an event from a sensor adaptor, the respective other condition is induced to synchronously query the sensor adaptors it has previously registered itself with for events. This procedure in particular serves to examine whether the context data contained in an event the condition has received before are still valid. As a consequence, virtual sensors are able to ensure that specifications made to them are fulfilled in their entirety before an event is fired and that no events are based on outdated data. The context data derivation carried out in the context-aware portal is based on a Context SRML rule we have already mentioned. Since this rule has been expected to be frequently used, it is already contained in the implementation of the Context Component. The rule specifies that whenever at least two persons are located in a meeting room their activity is assumed to be »meeting«. The schedules-based context sensor therefore is not necessarily needed to detect the context element of activity. Instead, on the basis of data gathered from the RFID-based sensor alone it may be possible to determine that a context specification is fulfilled without the need for the respective user's electronic schedule to contain an appropriate entry.

In the context-aware portal requests made to the Context Component with regard to the context elements of location and activity are always asynchronous. In contrast to this, reachability information is queried synchronously only. Reachability data are gathered from a single sensor and cannot be determined by means of derivation. Therefore, context data filtering and aggregation are not needed in the portal. Correspondingly, context builders are not made use of, and our application does not comprise an implementation of the `ContextBuilder` interface. To fulfill the requests made to the Context Component the `ContextGatherer` instance merely translates the data it receives from virtual sensors into appropriate result objects and returns them to the `ContextComponentBackEnd` object it has been queried by.

The Context Component's physical configuration made use of in the context-aware portal to a large extent corresponds to the standard configuration for medium-sized applications. The sensor adaptor for the KXS-based sensor as well as all other modules of the component software except for those containing the other sensor adaptors have been deployed on a single node. The sensor adaptor for the RFID-based location sensor and those for the schedules-based sensor are each executed on a dedicated node to avoid bottlenecks. This configuration is employed both for development and testing and during the operation of the application.

7.3 Summary and Assessment

This chapter has described our implementation of the Context Component as well as of a context-aware information logistic application using it. The software we have developed serves to prove the soundness and validity of our concepts, models, and architecture for the Context Component. By means of the context-aware information logistic application we furthermore aim to demonstrate how information logistic information supply can benefit from the consideration of context as a key dimension of information logistics. This section briefly sums up our explanations concerning the implementation of the Context Component and the context-aware application and in addition provides an evaluation of the results we have achieved.

The implementation of the Context Component complies with the reference architecture we have defined for this component. Consequently, the Context Component seamlessly integrates into the information logistics framework. Since our implementation of the component aims at providing universally applicable and reusable software, it only covers the generic parts of the Context Component. If required, application-specific functionality is expected to be implemented during the development of the particular applications. This distinction between generic and application-specific modules facilitates the maintenance of the Context Component and its integration into information logistic applications as well as the development and deployment of context-aware information logistic applications using the component.

Following our explanations concerning the implementation of the Context Component we have presented the context-aware information logistic application that has been designed and developed to illustrate the usage of the Context Component. This context-aware portal not only adapts information supply unobtrusively to the dimension of context, but also allows users to individually specify the contexts that are relevant to them. We have explained the architecture of the context-aware portal and the way its functionality is made available. Subsequently, details regarding the design and implementation of the portal have been presented. In doing so, we have in particular discussed extensions that have been made to the Context Component in order to enable it to handle application-specific context data and context sensors. The three context sensors employed in the context-aware portal have been described along with the sensor adaptors we have developed to integrate them into the Context Component. In addition, further special characteristics of the context gathering processes that takes place in the context-aware portal have been pointed out.

During the development of the context-aware portal we have learned by experience that any context-aware application can only take a subset of all the concepts and mechanisms we have developed into account. Due to the sophistication and complexity of the features the Context Component is able to provide it is hardly sensible to integrate the entirety of these features into a single application. The users of such an application would very likely be overwhelmed with the available variety of functionalities and alternatives and would be unable to grasp the application's purpose and benefits. As a result, the context-aware information logistic application would suffer from extremely poor usability and acceptance. To demonstrate how each of the concepts we have developed can come into action in a useful software system we would have to implement a multitude of different context-aware information logistic applications. This, however, is beyond the scope of this thesis. Therefore, we have designed and developed a context-aware information logistic application that supports a limited number of context

attributes and a subset of all possible sensor systems and techniques for context gathering only and that thus ensures usability. Yet, by making use of several different pieces of contextual information and considering the key aspects of context gathering and management the application shows how the solutions we have presented in this thesis allow context-aware information logistic applications of high quality to be developed. The context-aware portal is mainly targeted on demonstrating that the functional requirements made onto context awareness in information logistics are fulfilled by our approach. We therefore do not leave unmentioned that non-functional requirements – as can be seen, for example, in the design of the sensor adaptors for the schedules-based context sensor – have been paid slightly less attention. Similarly, since the context-aware portal serves to prove the validity of our solutions to making information logistic applications context-aware, it comprises only a small subset of the existing concepts regarding the other key dimensions of information logistics.

The context-aware portal shows that by considering the dimension of context the quality of information supply can significantly be increased. The main benefits of this application are:

- According to their context users are automatically supplied with information that is particularly relevant to them. As a result, they are enabled to focus on their actual tasks rather than having to spend valuable time on searching for the information they need.
- Since users are enabled to define individual contexts, information supply is personalized to a large extent. Users' information demands are therefore met very accurately.
- The context-aware information supply is unobtrusive and does not interrupt the current activities of users.
- By allowing users to manually change their contexts at any time the acceptance of the application is increased, and feelings of heteronomy are avoided.
- The usage of context sensors allows to adapt information supply to the individual needs of users without requiring them to make any explicit input.

Generally speaking, an integration of context awareness into information logistics contributes to providing people with a quick and easy and thus cost-effective access to relevant information at any time and place. A consideration of the dimension of context therefore is essential to information logistics to achieve a comprehensive optimization of information supply that results in a precise fulfillment of people's demands.

8 Conclusions

This thesis is based on the insight that in order to achieve the goal of an optimized information supply information logistics needs to take the dimension of context into account. Contextual information has a significant impact on the relevance of particular pieces of information to individuals and on the way information is best made available to them. However, the dimension of context has not been considered in information logistics so far. This thesis has presented solutions that integrate context as a key dimension into information logistics and thus make information logistic applications context-aware. As a result, the information demands of individuals can be fulfilled to a significantly greater extent, and the quality of information supply is increased substantially. The consideration of context in information logistics enabled by the findings of this thesis therefore allows to truly optimize information supply with regard to various dimensions.

To facilitate and speed up the development of information logistic applications a reference architecture for this type of software systems, the information logistics framework, has been defined. Information logistic applications are composed of several specialized components each of which provides a particular part of the applications' overall functionality. The desire to integrate a new dimension, the dimension of context, and novel features related to it into information logistic applications therefore requires a new component to be added to the information logistics framework. This new component which we have named the Context Component is responsible for dealing with the various aspects related to the dimension of context. This thesis has provided conceptual solutions and a reference architecture for the Context Component that enable information logistic applications to become context-aware.

When introducing the fundamental concepts related to the research of this thesis in Chapter 2, we have clarified the meaning and scope of context in the area of information logistics by defining the term context and other terms closely related to it. The Context Component is responsible for dealing with those pieces of contextual information which so far have not been addressed in information logistics. Thus, the context elements taken into account by this component are location, state, reachability, and surroundings. In addition, Chapter 2 has identified and described the requirements the Context Component has to fulfill in order to enable an adaptation of information supply to the dimension of context. Essentially, the Context Component is responsible for providing a context model as a basis for the consistent processing of context throughout information logistic applications. In addition, context data have to be gathered, managed, and supplied to other components in an efficient manner. The architecture of the Context Component has to ensure that the component fulfills the functional and non-functional requirements made onto it in a high-quality manner and that the concerns of various stakeholders are addressed comprehensively. The examination of existing approaches and standardization efforts related to the area of context-aware computing in Chapter 3 has shown that the currently available solutions do not satisfactorily fulfill the requirements made onto the Context Component. We have therefore recognized the need to develop novel concepts regarding the modelling, gathering, management, and supply of context in information logistic applications and to design an innovative reference architecture for the Context Component in order to meet the requirements we have identified and to enable an extensive consideration of the dimension of context in information logistics.

The first major contribution of this thesis is the model for context presented in Chapter 4. This model allows all components of information logistic applications to consistently represent and efficiently process context. In addition, it ensures that contexts defined as parts of users' demand profiles can be compared to contexts that have been gathered by means of sensors. The context model covers the abovementioned context elements of location, state, reachability, and surroundings and provides a means to describe all relevant attributes of each of these elements in a structured and formal manner. Contextual information can be captured at different degrees of complexity and precision, thus suiting the needs of diverse applications. Since in the context model several novel approaches towards representing context are incorporated, the model is more expressive than other proposed models for context. During the design of the model special attention has been paid to aspects of extensibility and generality. As a consequence, the context model is universally applicable to any application domain and can flexibly be adapted to different environments. The model for context we have presented thus allows to comprehensively capture a variety of contextual information and enables the development of sophisticated context-aware information logistic applications.

In Chapter 5 techniques for an efficient gathering and management of context which constitute the second main contribution of this thesis have been presented. These context gathering techniques are based on novel programming abstractions related to the different tasks that have to be carried out in conjunction with context gathering. They enable the Context Component to obtain data from heterogeneous context sensors. Criteria for the assessment of context sensors and a procedure for transforming sensor data into data according to the context model facilitate the selection of suitable sensors and their integration into the Context Component. In addition, configurable mechanisms allowing to augment the data received from context sensors in various ways significantly increase the value of the data the Context Component provides. In order to enable an assessment of the data supplied by the Context Component context data can be associated with information about their quality. As a result, our context gathering techniques ensure that other components of information logistic applications are provided with meaningful contextual information according to their specific needs. Due to their generic and flexible nature the context gathering techniques are universally applicable and are able to meet the demands of various application domains and environments.

This thesis' third contribution is the reference architecture for the Context Component described in Chapter 6. This architecture ensures that the Context Component can easily be integrated into information logistic applications and into heterogeneous environments. It addresses the concerns of various stakeholders and thus provides them with a guideline covering all aspects of the Context Component's life cycle. The reference architecture for the Context Component supports the fulfillment of both the functional and the non-functional requirements made onto the component and ensures that the component software is of high quality. Consequently, the Context Component's architecture facilitates the component's design, implementation, test, operation, and maintenance and enables powerful information logistic applications that are able to optimize information supply with regard to the dimension of context to be developed.

An implementation of the Context Component and of a context-aware information logistic application using it has been presented in Chapter 7. As the fourth major contribution of this thesis this implementation shows how our conceptual solutions come into operation in infor-

mation logistic applications. It demonstrates the usage of the Context Component in conjunction with the conceptual design, development, and operation of context-aware information logistic applications and the benefits of a consideration of context to information logistics. The implementation of the Context Component and the context-aware information logistic application therefore proves the validity and usability of the context model, the context gathering techniques, and the reference architecture for the Context Component we have presented.

This thesis has provided solutions that enable information logistic applications to become context-aware. However, due to the large scope the research area of context awareness has some issues related to context-aware computing are left to further investigation. Such aspects the consideration of which can contribute to improving our approach in the future include:

- Context-dependent information demands

  The need for information logistic applications to become context-aware is to a great extent determined by the fact that users' information demands frequently depend on context. Since the dimension of context, however, has so far not been considered in information logistic applications, no mechanisms to determine, model, and manage information demands that explicitly consider the specific characteristics of this dimension exist. Therefore, issues such as an automatic derivation of information demands on the basis of historical data about users' contexts or a representation of the relationship between context and desired information can be addressed by future research to further improve the quality of information supply and the user-friendliness of information logistic applications.

- Privacy and security issues

  Context-aware information logistic applications gather and process a lot of personal data about individuals. Contextual information is particularly sensitive as it describes potentially very intimate aspects of people's lives. It is obvious that these data have to be protected against unauthorized disclosure and modification. Since the desired level of protection varies between different users and different pieces of contextual information, the required privacy and security mechanisms have to be highly flexible. Thus, mechanisms to ensure the protection of context data are of great importance to the smooth operation and acceptance of information logistic applications.

- Further applications

  The various features the Context Component is able to provide need to be explored and made use of in more context-aware information logistic applications. These applications can address the requirements of several other domains, support possibly more compelling scenarios, incorporate additional optimization strategies, and finally drive the further development of the Context Component and of information logistic applications in general.

Appendix: Context SRML DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Context Simple Rules DTD based on ILOG Simple Rules DTD -->

<!--
   A constant expression has a required type (enumerated) and
   a value (cdata). The DTD provides the common types that
   exist in the programming languages. The notations allowed
   for specifying the literal values (for instance hexadecimal
   numbers or special characters) are the same as for the Java
   programming language.
-->
<!ELEMENT constant EMPTY>
<!ATTLIST constant
   type (string | boolean | byte | short | char | long | int | float |
         double | null | class) #REQUIRED
   value CDATA #REQUIRED
>

<!--
   Defines an assignable entity. An assignable entity is a variable
   or a field. An assignable entity has a value and can change values
   using assignments. As an assignable entity is also a value, it
   can appear in an expressison.
-->
<!ENTITY % assignable "(variable | field)">

<!--
   A variable has simply a name. A variable is created using "bind"
   or as the object variable of a simple condition. A variable is
   an "assignable" and can change values using "assignment".
-->
<!ELEMENT variable EMPTY>
<!ATTLIST variable
   name NMTOKEN #REQUIRED
>

<!--
   A field value. A field value operates on an object (an expression)
   and a field name. If the object is not specified, it is set to
   be the current default object. In a simple condition or a not
   condition, the default object is the one currently matched by the
   condition. For "assert" and "modify", the default object is the
   target of the assertion or the modification.
-->
<!ELEMENT field (%expression;)?>
<!ATTLIST field
   name NMTOKEN #REQUIRED
>
```

```
<!--
   Defines the expressions of the language. An expression can be
   an assignable (variable or field), a constant (a literal), an
   arithmetic expression or a boolean expression (a test).
   It can further be a method defined on an object.
-->
<!ENTITY % expression "(%assignable; | constant | unaryExp | binaryExp |
                        naryExp | method)">

<!--
   Defines the unary expressions. A unary expression acts on a
   single argument. The operator is specified using an enumerated
   type (the "operator" attribute). Note: the traditional
   signs like +, -, ! can not be part of enumerations. This
   justifies that we use symbolic names for operators.
-->
<!ELEMENT unaryExp (%expression;)>
<!ATTLIST unaryExp
   operator (plus | minus | not) #REQUIRED
>

<!--
   Defines the binary expressions. A binary expression acts on two
   arguments. Binary operators are not specified using enumerated symbolic names
   because the possible operators depend on the context elements or their
   belonging classes and are provided by them dynamically.
-->
<!ELEMENT binaryExp (%expression;, %expression;)>
<!ATTLIST binaryExp
   operator CDATA #REQUIRED
>

<!--
   A N-ary expression acts on N expressions. The number of expressions
   must be at least two. The operators are specified using symbolic
   names. Those names correspond to these signs:
      +, -, *, /, %, &&, ||
-->
<!ELEMENT naryExp (%expression;)+>
<!ATTLIST naryExp
   operator (add | subtract | multiply | divide | remainder | and | or) #REQUIRED
>

<!--
   An assignment. This uses two expressions: a first expression
   (an "assignable") which will be assigned the value of a second
   expression.
-->
<!ELEMENT assignment (%assignable;, %expression;)>
```

```
<!--
   A variable binding declares a variable and sets it to some
   initial value. A variable can change values by "assignment".
-->
<!ELEMENT bind (%expression;)>
<!ATTLIST bind
   name NMTOKEN #REQUIRED
>

<!--
   The ruleset is composed of a list of rules. A ruleset has a name
   (optional). The ruleset is the root element of an XML document.
-->
<!ELEMENT ruleset (rule*)>
<!ATTLIST ruleset
   name NMTOKEN #IMPLIED
>

<!--
   A rule has a name (required), a description (required), a condition
   part and an action part.
-->
<!ELEMENT rule (description, conditionPart, actionPart)>
<!ATTLIST rule
   name NMTOKEN #REQUIRED
>

<!ELEMENT priority (%expression;)>

<!ELEMENT conditionPart (%condition;)+>

<!ELEMENT actionPart (%action;)+>

<!--
   A condition is either a simple condition, a not condition,
   a lookup condition, or a multistage condition.
   These conditions differ by the fact that a simple condition matches
   an object which can be bound to a variable, while the not condition
   does not match any object.
   A multistage condition contains at least two conditions, and a lookup
   condition specifies requirements onto the sensor adaptors used that
   are needed for sensor lookup and query.
-->
<!ENTITY % condition "(simpleCondition | notCondition |
                       multistageCondition | lookupCondition)">

<!--
   Defines the possible actions allowed in the rule action part.
   This includes variable declarations and assignments, as well as
   the traditional assert, retract and modify statements of rule
   languages.
-->
<!ENTITY % action "(assignment | bind | assert | assertobj | modify | retract)">
```

```
<!--
   A simple condition has a class name (required), a variable
   name (optional) to which the object is bound. The body of the
   condition is composed of test expressions. Under the scope
   of a simple condition, the default object is the one tested
   by this condition.
-->
<!ELEMENT simpleCondition (%expression;)*>
<!ATTLIST simpleCondition
   className CDATA #REQUIRED
   objectVariable NMTOKEN #IMPLIED
>

<!--
   A not condition has a class name (required). The body of the
   condition may contain the same test expressions as for
   the simple condition. A not condition can not be bound to
   a variable. Under the scope of a not condition, the default
   object is the one tested by this condition.
-->
<!ELEMENT notCondition (%expression;)*>
<!ATTLIST notCondition
   className CDATA #REQUIRED
>

<!--
   The assert action. This action requires a class name and may
   specify field assignment statements. An instance of the class
   is created, the statements (assignments) are executed, and the
   object is then added to the working memory. Under the scope of
   an assert, the default object is the one currently asserted.
-->
<!ELEMENT assert (assignment | bind)*>
<!ATTLIST assert
   className CDATA #REQUIRED
>

<!--
   Another assert action. This action asserts an object computed from
   an expression. It differs from the previous "assert" by the fact
   that the object may be already created and returned as the value
   of the expression.
-->
<!ELEMENT assertobj (%expression;)>

<!--
   The retract action. This action removes an object from the working
   memory. The variable represents an object previously bound from
   the condition part.
-->
<!ELEMENT retract (variable)>
```

```
<!--
   The modify action. The action modifies an object of the working
   memory. The object is identified by a variable. The block of
   statements the the same as for the assert action. Under the scope
   of a modify, the default object is the one currently modified.
-->
<!ELEMENT modify (variable, (assignment | bind)+)>

<!--
   Each rule has a description which states the priority (optional) and
   reliability (optional) of the rule and gives information about the entities
   (optional) and context elements the rule applies to (required).
-->
<!ELEMENT description (priority?, reliability?, entity?, contextElement+)>

<!ELEMENT reliability (%expression;)>

<!--
   This element gives information about which entities this rule applies to.
   An entity has a class name quoting its type and can be further specified by
   an expression quoting the individual entities the rule applies to.
-->
<!ELEMENT entity (%expression;)?>
<!ATTLIST entity
   typeClassName CDATA #IMPLIED
>

<!--
   A context element has a class name (required) and can be
   further specified by an expression quoting the individual
   context element instances the rule can provide and the
   possible formats of the context element.
-->
<!ELEMENT contextElement ((%expression;)?, format?)>
<!ATTLIST contextElement
   className CDATA #REQUIRED
>

<!--
   A context element's format is specified by an expression
   quoting the individual format values the rule can provide.
-->
<!ELEMENT format (%expression;)>

<!--
   A lookup condition provides details about requirements
   the sensor adaptors employed to apply this rule have to fulfill.
   It is required to have a class name.
-->
<!ELEMENT lookupCondition (%expression;)+>
<!ATTLIST lookupCondition
   className CDATA #REQUIRED
>
```

```
<!--
   A multistage condition consists of a condition
   relating to the context elements provided by the sensor adaptors
   (simple condition or not condition) and a lookup condition.
   There can also be an optional result hash that is constructed
   with the results of the former conditions.
-->
<!ELEMENT multistageCondition ((%condition;)+, resultHash?)>

<!--
   A result hash represents a hashtable in which the results of
   two conditions are stored in relation to each other.
-->
<!ELEMENT resultHash (%expression;, %expression;)>

<!--
   A method is defined on an object given by an object variable.
   It has a name and can have one or more parameters (expressions).
-->
<!ELEMENT method (%expression;)*>
<!ATTLIST method
   name NMTOKEN #REQUIRED
>
```

References

[3GPP03] 3rd Generation Partnership Project, Technical Specification Group Core Network: Universal Geographical Area Description (GAD), Release 5, 3GPP TS 23.032 V5.0.0 (2003-03). Valbonne, 3GPP, 2003

[AdCu01] Addlesee, Mike; Curwen, Rupert; Hodges, Steve; Newman, Joe; Steggles, Pete; Ward, Andy; Hopper, Andy: Implementing a Sentient Computing System. In: IEEE Computer 34 (8), August 2001, pp. 50-56

[AlBo01] Altheim, Murray; Boumphrey, Frank; Dooley, Sam; McCarron, Shane; Schnitzenbaumer, Sebastian; Wugofski, Ted (eds.): Modularization of XHTML™, W3C Recommendation 10 April 2001. W3C, 2001

[Arsa01] Arsanjani, Ali: Rule Object 2001: A Pattern Language for Adaptive and Scalable Business Rule Construction. In: Online Proceedings of the 8th Conference on Pattern Languages of Programs (PLoP 2001), September 11-15, Allerton Park Monticello, IL. 2001. Available at http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/AArsanjani1/PLoP2001_AArsanjani1_0.pdf

[AtFi02] Atkinson, Rob; Fitzke, Jens (eds.): Gazetteer Service Profile of the Web Feature Service Implementation Specification, OpenGIS© Discussion Paper, Version 0.9. OGC, September 20, 2002

[BaBr99] Ball, Gene; Breese, Jack: Modeling the Emotional State of Computer Users. In: Workshop on Attitude, Personality and Emotions in User-Adapted Interaction, International Conference on User Modeling, June 20-24, Banff, Canada. 1999.

[BaCl03] Bass, Len; Clements, Paul; Kazman, Rick: Software Architecture in Practice, Second Edition. Addison Wesley Professional, 2003

[BaKa99] Bass, Len; Kazman, Rick: Architecture-Based Development. Technical Report CMU/SEI-99-TR-007. Pittsburgh, Software Engineering Institute, April 1999. Available at http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr007.pdf

[BaLo94] Bates, Joseph; Loyall, A. Bryan; Reilly, W. Scott: An Architecture for Action, Emotion, and Social Behavior. In: Castelfranchi, C.; Wemer, E. (eds.): Artificial Social Systems, Selected Papers from the Forth European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '92), July 29-31, S. Martino al Cimino, Italy. Heidelberg, Springer, 1994, pp. 55-68

[Bard98] Bardram, Jakob E.: Collaboration, Coordination and Computer Support: An Activity Theoretical Approach to the Design of Computer Supported Cooperative Work. PhD Thesis, University of Aarhus, Department of Computer Science, 1998

[BeBe97] Belussi, Alberto; Bertino, Elisa; Catania, Barbara: Manipulating Spatial Data in Constraint Databases. In: Scholl, M.; Voisard, A. (eds.): Proceedings of the 5th International Symposium on Advances in Spatial Databases (SSD'97), July 15-18, Berlin, Germany. Springer, 1997, pp. 115-141

[BeGe01] Beigl, Michael; Gellersen, Hans-W.; Schmidt, Albrecht: Mediacups: Experience with Design and Use of Computer-Augmented Everyday Artefacts. In: Computer Networks 35 (4), Special Issue on Pervasive Computing, March 2001, pp. 401-409

[BeHa02] Bertolotto, Michela; O'Hare, Gregory; Strahan, Robin; Brophy, Ailish; Martin, Alan; McLoughlin, Eoin: Bus Catcher: a Context Sensitive Prototype System for Public Transportation Users. In: Huang, B.; Ling, T. W.; Mohania, M. K.; Ng, W. K.; Wen, J.-R.; Gupta, S. K. (eds.): Proceedings of the 3rd International Conference on Web Information Systems Engineering Workshops (WISE 2002 Workshops), December 11, Singapore. IEEE Computer Society, 2002, pp. 64-72

[Beig00] Beigl, Michael: MemoClip: A Location based Remembrance Appliance. In: Personal Technologies 4 (4), September 2000, pp. 230-234

[BoQu96] Boncz, Peter A.; Quak, Wilko; Kersten, Martin L.: Monet And Its Geographic Extensions: a Novel Approach to High Performance GIS Processing. In: Apers, P. M. G.; Bouzeghoub, M.; Gardarin, G. (eds.): Advances in Database Technology, Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96), March 25-29, Avignon, France. Springer, 1996, pp. 147-166

[BoTa01] Boley, Harold; Tabet, Said; Wagner, Gerd: Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In: Cruz, I.; Decker, S.; Euzenat, J.; McGuinness, D. (eds.): Proceedings of SWWS '01 The First Semantic Web Working Symposium, July 30 - August 1, Stanford University, CA. Stanford University, 2001, pp. 381- 402

[Bott02] Botts, Mike (ed.): Sensor Model Language (SensorML) for In-situ and Remote Sensors, Version 0.7. OGC, December 20, 2002

[BrBo97] Brown, Peter J.; Bovey, John D.; Chen, Xian: Context-aware Applications: from the Laboratory to the Marketplace. In: IEEE Personal Communications 4 (5), October 1997, pp. 58-64

[Brow96] Brown, Peter J.: The Stick-e Document: A framework for creating context-aware applications. In: Brown, A.; Brüggemenn-Klein, A.; Feng, A. (eds.): Proceedings of the Sixth International Conference on Electronic Publishing, Document Manipulation and Typography, September 24-26, Palo Alto, CA. John Wiley and Sons, 1996, pp. 259-272

[Brow98] Brown, Peter J.: Triggering information by context. In: Personal Technologies 2 (1), September 1998, pp. 1-9

[BrPa04] Bray, Tim; Paoli, Jean; Sperberg-McQueen, C. M.; Maler, Eve; Yergeau, François (eds.): Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation 04 February 2004. W3C, 2004

[Buch03] Buchler, Kurt (ed.): OpenGIS® Reference Model, Version 0.1.2. OGC, March 4, 2003

[BuDe99] Busse, Susanne; Deiters, Wolfgang; Fuchs-Kittowski, Frank; Lienemann, Carsten; Neuhaus, Jan; Pfennigschmidt, Stefan; Wojciechowski, Manfred: Informationslogistik am Fraunhofer ISST. Internal Report. Dortmund and Berlin, Fraunhofer ISST, November 1999 (in German)

[BuPr01] Burnett, Mark; Prekop, Paul; Rainsford, Chris: Intimate Location Modeling for Context Aware Computing. In: Beigl, M.; Gray, P.; Salber, D. (eds.): Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, September 30, Atlanta, GA. 2001, pp. 77-82. Available at http://www.teco.edu/locationws/13.pdf

[ChKo00] Chen, Guanling; Kotz, David: A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science. Dartmouth College, November 2000

[ChKo01] Chen, Guanling; Kotz, David: Solar: Towards a Flexible and Scalable Data-Fusion Infrastructure for Ubiquitous Computing. In: Online Proceedings of the UbiTools '01 Workshop on Application Models and Programming Tools for Ubiquitous Computing, September 30, Atlanta, GA. Available at http://actcomm.dartmouth.edu/papers/chen:solar.pdf

[ChKo02a] Chen, Guanling; Kotz, David: Context Aggregation and Dissemination in Ubiquitous Computing Systems. In: Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '02), June 20-21, Callicoon, NY. IEEE Computer Society, 2002, pp. 105-114

[ChKo02b] Chen, Guanling; Kotz, David: Solar: An Open Platform for Context-Aware Mobile Applications. In: In: Mattern, F.; Naghshineh, M. (eds.): Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002), August 26-28, Zurich, Switzerland. Berlin, Springer, 2002, pp. 41-47

[ChKo03] Chen, Guanling; Kotz, David: Context-Sensitive Resource Discovery. In: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03), March 23-26, Fort Worth, TX. IEEE Computer Society, 2003, pp. 243-252

[ChKu03] Chong, Chee-Yee; Kumar, Srikanta P.: Sensor Networks: Evolution, Opportunities, and Challenges. In: Proceedings of the IEEE 91 (8), August 2003, pp. 1247-1256

[CoDa03] Cox, Simon; Daisey, Paul; Lake, Ron; Portele, Clemens; Whiteside, Arliss (eds.): OpenGIS® Geography Markup Language (GML) Implementation Specification, Version 3.00. OGC, January 29, 2003

[CoDo00] Cowie, Roddy; Douglas-Cowie, Ellen; Savvidou, Susie; McMahon, Edelle; Sawey, Martin; Schröder, Marc: 'FEELTRACE': An Instrument for Recording Perceived Emotion in Real Time. In: Cowie, R.; Douglas-Cowie, E.; Schröder, M. (eds.): Proceedings of ISCA Workshop on Speech and Emotion: A Conceptual Framework for Research, September 5-7, The Queen's University of Belfast, Northern Ireland. Belfast, Textflow, 2000, pp. 19-24

[CoKa98] Cohn, Jeffrey F.; Katz, Gary S.: Bimodal Expression of Emotion by Face and Voice. In: Ohya, J.; Nakatsu, R.; Reilly, R. (eds.): Proceedings of the Sixth ACM International Conference on Multimedia: Face/Gesture Recognition and their Applications, September 12-16, Bristol, UK. New York, ACM Press, 1998, pp. 41-44

[CoLe02] Cohen, Norman H.; Lei, Hui; Castro, Paul; Davis II, John S.; Purakayastha, Apratim: Composing Pervasive Data Using iQL. In: Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '02), June 20-21, Callicoon, NY. IEEE Computer Society, 2002, pp. 94-104

[CoPu02] Cohen, Norman H.; Purakayastha, Apratim; Wong, Luke; Yeh, Danny L.: iQueue: A Pervasive Data Composition Framework. In: Proceedings of the Third International Conference on Mobile Data Management (MDM 2002), January 8-11, Singapore. IEEE Computer Society, 2002, pp. 146-156

[CoTa95] Cooperstock, Jeremy R.; Tanikoshi, Koichiro; Beirne, Garry; Narine, Tracy; Buxton, William: Evolution of a reactive environment. In: Proceedings of the 1995 ACM Conference on Human Factors in Computing Systems (CHI '95), May 7-11, Denver, CO. ACM, 1995, pp. 170-177

[Cove01] Cover, Robin: Simple Rule Markup Language (SRML). Technology Report, May 2001. Available at http://xml.coverpages.org/srml.html

[DaMi98] Davies, Nigel; Mitchell, Keith; Cheverst, Keith; Blair, Gordon: Developing a context-sensitive tour guide. In: Proceedings of the 1st Workshop on Human Computer Interaction for Mobile Devices, May 21-23, Glasgow, Scotland. 1998, pp. 64-68

[DaWa97] Davies, Nigel; Wade, Stephen; Friday, Adrian; Blair, Gordon: Limbo: A tuple space based platform for adaptive mobile applications. In: Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97), May 27-30, Toronto, Canada. 1997, pp. 291-302

[DeAb01] Dey, Anind K.; Abowd, Gregory D.; Salber, Daniel: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. In: Human-Computer Interaction 16 (2-4), 2001, pp. 97-166

[DeLi01] Deiters, Wolfgang; Lienemann, Carsten (eds.): Report Informationslogistik - Informationen just-in-time. Düsseldorf, Symposion Publishing, 2001 (in German)

[DeLo03] Deiters, Wolfgang; Löffeler, Thorsten; Pfennigschmidt, Stefan: The Information Logistics Approach toward a User Demand-Driven Information Supply. In: Spinellis, D. D. (ed.): Proceedings of the International Conference on Cross-Media Service Delivery, May 30-31, Santorini, Greece. Kluwer, 2003, pp. 37-48

[DeLu00] Deiters, Wolfgang; Lucas, Reinhard: Intelligente Informationsbereitstellung für Knowledge Worker. In: Becker, J.; Kaiser, B.; Wendt, S. (eds.): KnowTech2000 - Knowledge Engineering, Management, Consulting & Training, September 8, Leipzig, Germany. Leipzig, 2000 (in German)

[DeMi02] DeMichiel, Linda G. (Specification Lead): Enterprise JavaBeans™ Specification, Version 2.1. Santa Clara, Sun Microsystems, August 2002

[DeSa99] Dey, Anind K.; Salber, Daniel; Abowd, Gregory D.; Futakawa, Masayasu: The Conference Assistant: Combining Context-Awareness with Wearable Computing. In: Proceedings of the 3rd International Symposium on Wearable Computers (ISWC '99), October 18-19, San Francisco, CA. IEEE Computer Society, 1999, pp. 21-28

[Dey98] Dey, Anind K.: Context-aware computing: The CyberDesk project. In: Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments (AAAI Technical Report SS-98-02), March 23-25, Palo Alto, CA. AAAI Press, 1998, pp. 51-54

[Dey00] Dey, Anind K.: Providing Architectural Support for Building Context-Aware Applications. PhD Thesis, Georgia Institute of Technology, 2000

[Dey01] Dey, Anind K.: Understanding and Using Context. In: Personal and Ubiquitous Computing, Special Issue on Situated Interaction and Ubiquitous Computing 5 (1). London, Springer, 2001, pp. 4-7

[Dijk68] Dijkstra, Edsger W.: The Structure of the 'T.H.E.' Multiprogramming System. In: Communications of the ACM 11 (5), May 1968, pp. 341-346

[DoD97] United States Department of Defense: C4ISR Architecture Framework Version 2.0. December 18, 1997

[DrRi98] Drane, Chris; Rizos, Chris: Positioning Systems in Intelligent Transportation Systems. Boston, London, Artech House, 1998

[EbHu01] Ebling, Maria R.; Hunt, Guerney D. H.; Lei, Hui: Issues for Context Services for Pervasive Computing. In: Proceedings of the Workshop on Middleware for Mobile Computing (Middleware 2001). Heidelberg, 2001

[EfCh01] Efstratiou, Christos; Cheverst, Keith; Davies, Nigel; Friday, Adrian: An Architecture for the Effective Support of Adaptive Context-Aware Applications. In: Tan, K.-L.; Franklin, M. J.; Lui, J. C. S. (eds.): Proceedings of the Second International Conference on Mobile Data Management (MDM 2001), January 8-10, Hong Kong, China. Springer, 2001, pp. 15-26

[EfFr02a] Efstratiou, Christos; Friday, Adrian; Davies, Nigel; Cheverst, Keith: Utilising the Event Calculus for Policy Driven Adaptation on Mobile Systems. In: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), June 5-7, Monterey, CA. IEEE Computer Society, 2002, pp. 13-24

[EfFr02b] Efstratiou, Christos; Friday, Adrian; Davies, Nigel; Cheverst, Keith: A Platform Supporting Coordinated Adaptation in Mobile Systems. In: Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '02), June 20-21, Callicoon, NY. IEEE Computer Society, 2002, pp. 128-137

[Egen89] Egenhofer, Max: A Formal Definition of Binary Topological Relationships. In: Litwin, W.; Schek, H.-J. (eds.): Proceedings of the 3rd International Conference on Foundations of Data Organization and Algorithms (FODO 1989), June 21-23, Paris, France. Springer, 1989, pp. 457-472

[EmHi96] Emery, David E.; Hilliard II, Richard F.; Rice, Timothy B.: Experiences Applying a Practical Architectural Method. In: Strohmeier, A. (ed.): Reliable Software Technologies - Ada-Europe '96, 1996 Ada-Europe International Conference on Reliable Software Technologies, June 10-14, Montreux, Switzerland. Springer, 1996, pp. 471-484

[Enge87] Engeström, Yrjö: Learning by expanding: An activity-theoretical approach to developmental research. Helsinki, Orienta-Konsultit, 1987

[ErSc97] Erwig, Martin; Schneider, Markus: Vague Regions. In: Scholl, M.; Voisard, A. (eds.): Proceedings of the 5th International Symposium on Advances in Spatial Databases (SSD'97), July 15-18, Berlin, Germany. Springer, 1997, pp. 298-320

[EsCu02] Estrin, Deborah; Culler, David; Pister, Kris; Sukhatme, Gaurav: Connecting the Physical World with Pervasive Networks. In: IEEE Pervasive Computing 1 (1), January-March 2002, pp. 59-69

[FeBe01] Ferscha, Alois; Beer, Wolfgang; Narzt, Wolfgang: Location Awareness in Community Wireless LANs. In: Bauknecht, K.; Brauer, W; Mück, Th. (eds.): Informatik 2001 - Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit; Tagungsband der GI/OCG Jahrestagung 2001, September 25-28, Vienna, Austria. Vienna, Österreichische Computer-Gesellschaft, 2001, pp. 190-195

[FjLa02] Fjeld, Morten; Lauche, Kristina; Bichsel, Martin; Voorhorst, Fred; Krueger, Helmut; Rauterberg, Matthias: Physical and Virtual Tools: Activity Theory Applied to the Design of Groupware. In: Nardi, B. A.; Redmiles, D. F. (eds.): Special Issue of Computer Supported Cooperative Work (CSCW): Activity Theory and the Practice of Design 11 (1-2). Dordrecht, Kluwer, 2002, pp. 153-180

[FlPe00] Flynn, Mike; Pendlebury, David; Jones, Chris; Eldridge, Marge; Lamming, Mik: The Satchel System Architecture: Mobile Access to Documents and Services. In: Mobile Networks and Applications (MONET) 5 (4), 2000, pp. 243-258

[FrBo96] Freed, N.; Borenstein, N.: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. November 1996. Available at ftp://ftp.isi.edu/in-notes/rfc2045.txt

[GaHe94] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Addison-Wesley, 1994

[GaSh93] Garlan, David; Shaw, Mary: An Introduction to Software Architecture. In: Ambriola, V.; Tortora, G. (eds.): Advances in Software Engineering and Knowledge Engineering, Vol 1. River Edge, World Scientific Publishing Company, 1993, pp. 1-39

[GeBe99] Gellersen, Hans-W.; Beigl, Michael; Krull, Holger: The MediaCup: Awareness Technology embedded in an Everyday Object. In: Gellersen, H.-W. (ed.): Proceedings of the First International Symposium on Handheld and Ubiquitous Computing (HUC '99), September 27-29, Karlsruhe, Germany. Springer, 1999, pp. 308-310

[GeJe01] Gessler, Stefan; Jesse, Kai: Advanced Location Modeling to enable sophisticated LBS Provisioning in 3G networks. In: Beigl, M.; Gray, P.; Salber, D. (eds.): Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, September 30, Atlanta, GA. 2001, pp. 49-54. Available at http://www.teco.edu/locationws/9.pdf

[GeSc02] Gellersen, Hans-W.; Schmidt, Albrecht; Beigl, Michael: Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts. In: Mobile Networks and Applications (MONET) 7 (5), October 2002, pp. 341-351

[Gree92] Green, O. H.: The emotions: A philosophical theory. Dordrecht, Kluwer, 1992

[GrSp01] Gross, Tom; Specht, Marcus: Awareness in Context-Aware Information Systems. In: Oberquelle, H.; Oppermann, R.; Krause, J. (eds.): Mensch & Computer 2001: 1. Fachübergreifende Konferenz. Stuttgart, B.G. Teubner, 2001, pp. 173-181. Available at http://mc.informatik.uni-hamburg.de/konferenzbaende/mc2001/V33.pdf

[GuBo00] Güting, Ralf Hartmut; Böhlen, Michael H.; Erwig, Martin; Jensen, Christian S.; Lorentzos, Nikos A.; Schneider, Markus; Vazirgianni, Michalis: A foundation for representing and querying moving objects. In: ACM Transactions on Database Systems 25 (1), March 2000, pp. 1-42

[Guet88] Güting, Ralf Hartmut: Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In: Schmidt, J. W.; Ceri, S.; Missikoff, M. (eds.): Advances in Database Technology - EDBT'88, Proceedings of the International Conference on Extending Database Technology, March 14-18, Venice, Italy. Springer, 1988, pp. 506-527

[Guet94] Güting, Ralf Hartmut: An Introduction to Spatial Database Systems. In: VLDB Journal 3 (4), October 1994, pp. 357-399

[GuSc93] Güting, Ralf Hartmut; Schneider, Markus: Realms: A Foundation for Spatial Data Types in Database Systems. In: Abel, D. J.; Ooi, B. C. (eds.): Proceedings of the 3rd International Symposium on Advances in Spatial Databases, June 23-25, Singapore. Springer, 1993, pp. 14-35

[GuSc95] Güting, Ralf Hartmut; Schneider, Markus: Realm-Based Spatial Data Types: The Rose Algebra. In: VLDB Journal 4 (2), April 1995, pp. 243-286

[HaBe89] Hager, John W.; Behensky, James F.; Drew, Brad W.: The Universal Grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS). Technical Manual TM 8358.2. Washington, National Imagery and Mapping Agency, 1989

[HaFi98] Harrison, Beverly L.; Fishkin, Kenneth P.; Gujar, Anuj; Mochon, Carlos; Want, Roy: Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. In: Proceedings of the CHI'98 Conference on Human Factors in Computer Systems, April 18-23, Los Angeles, CA. ACM, 1998, pp. 17-24

[HaHo94] Harter, Andy; Hopper, Andy: A Distributed Location System for the Active Office. In: IEEE Network 8 (1), January 1994, pp. 62-70

[HaHo99] Harter, Andy; Hopper, Andy; Steggles, Pete; Ward, Andy; Webster, Paul: The Anatomy of a Context-Aware Application. In: Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99), August 17-19, Seattle, WA. ACM, 1999, pp. 59-68

[HaMe01] Hansmann, Uwe; Merk, Lothar; Nicklous, Martin S.; Stober, Thomas: Pervasive Computing Handbook. Heidelberg, Springer, 2001

[Hami97] Hamilton, Graham (ed.): JavaBeans™ 1.01 Specification. Mountain View, Sun Microsystems, August 1997

[Hase01a] Haseloff, Sandra: Designing Adaptive Mobile Applications. In: Klöckner, K. (ed.): Proceedings of the Ninth Euromicro Workshop on Parallel and Distributed Processing (PDP 2001), February 7-9, Mantova, Italy. Los Alamitos, IEEE Computer Society, pp. 131-138

[Hase01b] Haseloff, Sandra: Ortsunabhängige Informationsversorgung in informationslogistischen Anwendungen. In: Deiters, W; Lienemann, C. (eds.): Report Informationslogistik – Informationen just-in-time. Düsseldorf, Symposion Publishing, 2001, pp. 277-290 (in German)

[Hase01c] Haseloff, Sandra: Optimizing Information Supply by Means of Context: Models and Architecture. In: Bauknecht, K.; Brauer, W; Mück, Th. (eds.): Informatik 2001 - Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit; Tagungsband der GI/OCG Jahrestagung 2001, September 25-28, Vienna, Austria. Vienna, Österreichische Computer-Gesellschaft, 2001, pp. 206-213

[Hase04] Haseloff, Sandra: Context Gathering – an Enabler for Information Logistics. In: Chamoni, P.; Deiters, W.; Gronau, N.; Kutsche, R.-D.; Loos, P.; Müller-Merbach, H.; Rieger, B.; Sandkuhl, K. (eds.): Multikonferenz Wirtschaftsinformatik (MKWI) 2004, Band 2, March 9-11, Essen, Germany. Berlin, Akademische Verlagsgesellschaft, 2004, pp. 204-216

[Heal00] Healey, Jennifer: Wearable and Automotive Systems for the Recognition of Affect from Physiology. PhD Thesis, Massachusetts Institute of Technology, 2000

[HeDe03] Heuwinkel, Kerstin; Deiters, Wolfgang; Königsmann, Thomas; Löffeler, Thorsten: Information Logistics and Wearable Computing. In: Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCS 2003 Workshops), May 19-22, Providence, RI. IEEE Computer Society, 2003, pp. 283-289

[HeHo03] Hellerstein, Joseph M.; Hong, Wei; Madden, Samuel; Stanek, Kyle: Beyond Average: Toward Sophisticated Sensing with Queries. In: Zhao, F.; Guibas, L. (eds.): Proceedings of the Second International Workshop on Information Processing in Sensor Networks (IPSN 2003), April 22-23, Palo Alto, CA. Heidelberg, Springer, 2003, pp. 63-79

[HeIn02] Henricksen, Karen; Indulska, Jadwiga; Rakotonirainy, Andry: Modeling Context Information in Pervasive Computing Systems. In: Mattern, F.; Naghshineh, M. (eds.): Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002), August 26-28, Zurich, Switzerland. Berlin, Springer, 2002, pp. 167-180

[HeIn03] Henricksen, Karen; Indulska, Jadwiga; Rakotonirainy, Andry: Generating Context Management Infrastructure from High-Level Context Models. In: Proceedings of the 4th International Conference on Mobile Data Management (MDM 2003), Industrial Track Proceedings, January 21-24, Melbourne, Australia. 2003, pp. 1-6

[HeIn04a] Henricksen, Karen; Indulska, Jadwiga: Modelling and Using Imperfect Context Information. In: Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2004 Workshops), March 14-17, Orlando, FL. IEEE Computer Society, 2004, pp. 33-37

[HeIn04b] Henricksen, Karen; Indulska, Jadwiga: A Software Engineering Framework for Context-Aware Pervasive Computing. In: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), March 14-17, Orlando, FL. IEEE Computer Society, 2004, pp. 77-86

[HePi98] Healey, Jennifer; Picard, Rosalind W.: Digital Processing of Affective Signals. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICAASP '98), Vol. 6, May 12-15, Seattle, WA. IEEE Computer Society Press, 1998, pp. 3749-3752

[HePi00] Healey, Jennifer; Picard, Rosalind W.: SmartCar: Detecting Driver Stress. In: International Conference on Pattern Recognition (ICPR'00), Volume 4, September 3-8, Barcelona, Spain. IEEE Computer Society Press, 2000, pp. 4218-4221

[HiBo01] Hightower, Jeffrey; Borriello, Gaetano: Location Systems for Ubiquitous Computing. In: IEEE Computer 34 (8), August 2001, pp. 57-66

[HiBo04] Hirtle, David; Boley, Harold; Damasio, Carlos; Grosof, Benjamin; Kifer, Michael; Sintek, Michael; Tabet, Said; Wagner, Gerd: Schema Specification of RuleML 0.87. August 12, 2004. Available at http://www.ruleml.org/spec

[HiPi00] Hinckley, Ken; Pierce, Jeff; Sinclair, Mike; Horvitz, Eric: Sensing techniques for mobile interaction. In: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST 2000), November 5-8, San Diego, CA. ACM, 2000, pp. 91-100

[HiSz00] Hill, Jason; Szewczyk, Robert; Woo, Alec; Hollar, Seth; Culler, David; Pister, Kristofer: System Architecture Directions for Networked Sensors. In: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems (ASPLOS 2000), November 12-15, Cambridge, MA. ACM Press, 2000

[HoCo02] Horstmann, Cay S.; Cornell, Gary: Core Java 2 Volume I - Fundamentals. Sun Microsystems Press (Prentice Hall PTR), 2002

[HoHe03] Le Hors, Arnaud; Le Hégaret, Philippe; Wood, Lauren; Nicol, Gavin; Robie, Jonathan; Champion, Mike; Byrne, Steve (eds.): Document Object Model (DOM) Level 3 Core Specification, Version 1.0. W3C Working Draft 09 June 2003. W3C, 2003

[HoMa01] Holmquist, Lars Erik; Mattern, Friedemann; Schiele, Bernt; Alahuhta, Petteri; Beigl, Michael; Gellersen, Hans-W.: Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In: Abowd, G. D.; Brumitt, B.; Shafer, S. (eds.): Proceedings of the Third International Conference on Ubiquitous Computing (Ubicomp 2001), September 30 - October 2, Atlanta, GA. 2001, pp. 116-122

[HoNo99] Hofmeister, Christine; Nord, Robert; Soni, Dilip: Applied Software Architecture. Addison-Wesley, 1999

[Hopp99] Hopper, Andy: The Royal Society Clifford Paterson Lecture, 1999, Sentient Computing. Technical Report 1999.12. AT&T Laboratories Cambridge, 1999

[Horn51] Horn, Alfred: On Sentences Which are True of Direct Unions of Algebras. In: Journal of Symbolic Logic 16 (1), 1951, pp. 14-21

[HuNe97] Hull, Richard; Neaves, Philip; Bedford-Roberts, James: Towards situated computing. In: Proceedings of the 1st International Symposium on Wearable Computers (ISWC'97), October 13-14, Cambridge, MA. IEEE Computer Society, 1997, pp. 146-153

[IEEE00] IEEE 1471-2000 IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. IEEE Computer Society, October 2000

[InGo03] Intanagonwiwat, Chalermek; Govindan, Ramesh; Estrin, Deborah; Heidemann, John; Silva, Fabio: Directed Diffusion for Wireless Sensor Networking. In: IEEE/ACM Transactions on Networking 11 (1), February 2003, pp. 2-16

[InRo03] Indulska, Jadwiga; Robinson, Ricky; Rakotonirainy, Andry; Henricksen, Karen: Experiences in Using CC/PP in Context-Aware Systems. In: Chen, M.-S.; Chrysanthis, P. K.; Sloman, M.; Zaslavsky, A. B. (eds.): Proceedings of the 4th International Conference on Mobile Data Management (MDM 2003), January 21-24, Melbourne, Australia. Springer, 2003, pp. 247-261

[ISO96] International Organization for Standardization (ISO): Open Distributed Processing - Reference Model, ISO/IEC 10746-1 to 4. ISO, 1996, 1998

[ISO00] ISO/TC 211/WG 2/Editing committee 19107: Final text of CD 19107 Geographic information - Spatial schema, ISO/TC 211 N 1032. Oslo, ISO/TC 211 Secretariat, December 2000

[ISO02] International Organization for Standardization (ISO): Geographic information - Services, Draft International Standard ISO/DIS 19119. ISO, 2002

[ISO03] International Organization for Standardization (ISO): Geographic information - Positioning services, Draft International Standard ISO/DIS 19116. ISO, 2003

[JoDa99] José, Rui; Davies, Nigel: Scalable and Flexible Location-Based Services for Ubiquitous Information Access. In: Gellersen, H.-W. (ed.): Proceedings of the First International Symposium on Handheld and Ubiquitous Computing (HUC '99), September 27-29, Karlsruhe, Germany. Springer, 1999, pp. 52-66

[Klem00] Klemke, Roland: Context Framework - an Open Approach to Enhance Organisational Memory Systems with Context Modelling Techniques. In: Reimer, U. (ed.): Proceedings of the Third International Conference on Practical Aspects of Knowledge Management (PAKM2000), October 30-31, Basel, Switzerland. Available at http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-34/klemke.pdf

[KlMo02] Klein, Jonathan; Moon, Youngme; Picard, Rosalind W.: This Computer Responds to User Frustration: Theory, Design, Results, and Implications. In: Interacting with Computers 14 (2). Amsterdam, Elsevier, 2002, pp. 119-140

[KlRe04] Klyne, Graham; Reynolds, Franklin; Woodrow, Chris; Ohto, Hidetaka; Hjelm, Johan; Butler, Mark H.; Tran, Luu: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. W3C Recommendation 15 January 2004. W3C, 2004

[Koec97] Koecher, Max: Lineare Algebra und analytische Geometrie. Berlin, Springer, 1997 (in German)

[Koen00] Königsmann, Thomas: Ein personalisiertes Portal zum mobilen Informationsaustausch. Diploma Thesis, University of Dortmund, August 21, 2000 (in German)

[Kope01] Kopecek, Ivan: Personality and Emotions - Finite State Modelling by Dialogue Automata. In: 2nd Workshop on Attitude, Personality and Emotions in User-Adapted Interaction in conjunction with User Modeling 2001, July 13, Sonthofen, Germany. 2001. Available at http://aos2.uniba.it:8080/sonthofen/kopecek-f.ps

[KoPf01] Königsmann, Thomas; Pfennigschmidt, Stefan: ILOG Framework Redesign. Internal Report. Dortmund and Berlin, Fraunhofer ISST, 2001 (in German)

[KoRe01] Kort, Barry; Reilly, Rob; Picard, Rosalind W.: An Affective Model of Interplay between Emotions and Learning: Reengineering Educational Pedagogy-Building a Learning Companion. In: Okamoto, T.; Hartley, R.; Kinshuk; Klus, J. P. (eds.): Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT '01), August 6-8, Madison, WI. IEEE Computer Society, 2001, pp. 43-48

[KoSe03] Koubarakis, Manolis; Sellis, Timos K.; Frank, Andrew U.; Grumbach, Stéphane; Güting, Ralf Hartmut; Jensen, Christian S.; Lorentzos, Nikos A.; Manolopoulos, Yannis; Nardelli, Enrico; Pernici, Barbara; Schek, Hans-Jörg; Scholl, Michel; Theodoulidis, Babis; Tryfona, Nectaria (eds.): Spatio-Temporal Databases: The CHOROCHRONOS Approach. Springer, 2003

[KoTa01] Korkea-aho, Mari; Tang, Haitao; Racz, David; Polk, James M.; Takahashi, Kenji: A Common Spatial Location Data Set. Internet Draft. IETF, May 2001. Available at http://www.hut.fi/~mkorkeaa/papers/draft-korkea-aho-spatial-dataset-01.txt

[KoTa02] Korkea-aho, Mari; Tang, Haitao: A Common Data Set and Framework for Representing Spatial Location Information in the Internet. In: Cluster Computing 5 (4), 2002, pp. 389-397

[KrHa00] Krumm, John; Harris, Steve; Meyers, Brian; Brumitt, Barry; Hale, Michael; Shafer, Steven: Multi-Camera Multi-Person Tracking for EasyLiving. In: Proceedings of the Third IEEE International Workshop on Visual Surveillance (VS 2000), July 01, Dublin, Ireland. Piscataway, IEEE Computer Society Press, 2000, pp. 3-10

[Kris00] Krishnan, Venky: Location Awareness in HP's CoolTown. Position paper. Palo Alto, Hewlett-Packard Laboratories, 2000. Available at http://cooltown.hp.com/dev/wpapers/locationawareness/LocationAwareness.asp

[KrLj98] Kristoffersen, Steinar; Ljungberg, Fredrik: Representing Modalities in Mobile Computing. In: Urban, B., Kirste, T., Ide, R. (eds.): Proceedings of the International Workshop on Interactive Applications of Mobile Computing (IMC '98), November 24-25, Rostock, Germany. Rostock, Fraunhofer Institute for Computer Graphics, 1998

[KrLj99] Kristoffersen, Steinar; Ljungberg, Fredrik: Mobile Use of IT. In: Käkölä, T. K. (ed.): Proceedings of IRIS22, Vol. 2. Jyväskylä, Jyväskylä University Printing House, 1999, pp. 271-284

[Kruc95] Kruchten, Philippe: Architectural Blueprints - The "4+1" View Model of Architecture. In: IEEE Software 12 (6), November 1995, pp. 42-50

[Kruc00] Kruchten, Philippe: The Rational Unified Process: An Introduction, 2nd Edition. Reading, Addison-Wesley, 2000

[Laer99] Van Laerhoven, Kristof: On-line Adaptive Context Awareness Starting from Low-Level Sensors. Licentiaats thesis, University of Brussels, May 1999

[LaFl94] Lamming, Mik; Flynn, Mike: Forget-me-not: Intimate computing in support of human memory. In: Proceedings of the FRIEND 21: International Symposium on Next Generation Human Interfaces, Meguro Gajoen, Japan. 1994, pp. 125-128

[Lans01] Lansing, Jeff (ed.): Geoparser Service Specification, Version 0.7.1. OGC, March 27, 2001

[LaPa93] Larue, Thierry; Pastre, Dominique; Viémont, Yann: Strong Integration of Spatial Domains and Operators in a relational Database System. In: Abel, D. J.; Ooi, B. C. (eds.): Proceedings of the 3rd International Symposium on Advances in Spatial Databases, June 23-25, Singapore. Springer, 1993, pp. 53-72

[Leon78] Leont'ev, Aleksie Nikolaevich: Activity, Consciousness, and Personality. Englewood Cliffs, Prentice-Hall, 1978

[Leon98] Leonhardt, Ulf: Supporting Location-Awareness in Open Distributed Systems. PhD Thesis, University of London, Department of Computing, Imperial College of Science, Technology and Medicine, 1998

[LeSo02] Lei, Hui; Sow, Daby M.; Davis II, John S.; Banavar, Guruduth; Ebling, Maria R.: The Design and Applications of a Context Service. In: ACM SIGMOBILE Mobile Computing and Communications Review 6 (4), October 2002, pp. 45-55

[Lien01] Lienemann, Carsten: Visionen für die (Informations-) Welt von morgen. In: Deiters, W; Lienemann, C. (eds.): Report Informationslogistik - Informationen just-in-time. Düsseldorf, Symposion Publishing, 2001, pp. 305-323 (in German)

[LIF02] Location Inter-operability Forum (LIF): Mobile Location Protocol, LIF TS 101 Specification Version 3.0.0. LIF, 6 June 2002

[LiSc01] Lienemann, Carsten; Schreckenberg, Michael; Wahle, Joachim: Intelligente Verkehrsinformationen durch w@keup. In: Deiters, W; Lienemann, C. (eds.): Report Informationslogistik - Informationen just-in-time. Düsseldorf, Symposion Publishing, 2001, pp. 209-220 (in German)

[LiYe99] Lindholm, Tim; Yellin, Frank: The Java™ Virtual Machine Specification Second Edition. Palo Alto, Sun Microsystems, 1999

[LoKo96] Long, Sue; Kooper, Rob; Abowd, Gregory D.; Atkeson, Christopher G.: Rapid prototyping of mobile context-aware applications: The Cyberguide case study. In: Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking (MobiCom '96), November 10-12, Rye, NY. ACM, 1996, pp. 97-107

[LoPf01] Löffeler, Thorsten; Pfennigschmidt, Stefan; Wojciechowski, Manfred; Mewes, Michael: ILOG Framework Abschlussbericht. Internal Report. Dortmund and Berlin, Fraunhofer ISST, 2001 (in German)

[Mani03] Mani, Murali: Understanding the Semantics of Sensor Data. In: SIGMOD Record 32 (4), December 2003, pp. 28-34

[Marg01] Margoulies, Serge (ed.): Geocoder Service Specification, Version 0.7.6. OGC, March 28, 2001

[Math01] Mathias, Arun: SmartReminder: A Case Study on Context-Sensitive Applications. Technical Report TR2001-392, Dept. of Computer Science. Dartmouth College, 2001

[MiKo02] Minami, Kazuhiro; Kotz, David: Controlling access to pervasive information in the Solar system. Technical Report TR2002-422, Dept. of Computer Science. Dartmouth College, 2002

[Mitc02] Mitchell, Keith: Supporting The Development of Mobile Context-Aware Systems. PhD Thesis, Lancaster University, January 2002

[Momj01] Momjian, Bruce: PostgreSQL: Introduction and Concepts. Reading, Addison Wesley Professional, 2001

[MoPa92] Mouly, Michel; Pautet, Marie-Bernadette: The GSM System for Mobile Communications. Palaiseau, Cell & Sys, 1992

[Nard96] Nardi, Bonnie A.: Context and Consciousness: Activity Theory and Human-Computer Interaction. Cambridge, MIT Press, 1996

[NeEl91] Newman, William M.; Eldridge, Margery A.; Lamming, Michael G.: PEPYS: Generating Autobiographies by Automatic Tracking. In: Bannon, L.; Robinson, M.; Schmidt, K. (eds): Proceedings of the second European Conference on Computer Supported Cooperative Work (ESCSW '91), September 24-27, Amsterdam, The Netherlands. Dordrecht, Kluwer Academic Publishers, 1991

[Nels98] Nelson, Giles John: Context-Aware and Location Systems. PhD Thesis, University of Cambridge, January 1998

[NiHj00] Nilsson, Mikael; Hjelm, Johan; Ohto, Hidetaka: Composite Capabilities/Preference Profiles: Requirements and Architecture. W3C Working Draft. W3C, July 21, 2000

[OaJo96] Oatley, Keith; Johnson-Laird, Philip N.: The communicative theory of emotions: Empirical tests, mental models, and implications for social interaction. In: Martin, L.; Tesser, A. (eds.): Striving and feeling: Interactions between goals and affect. Hillsdale, Lawrence Erlbaum, 1996

[OMG03] Object Management Group, Inc.: OMG Unified Modeling Language Specification - Version 1.5. Needham, Object Management Group, March 2003

[OrCl88] Ortony, Andrew; Clore, Gerald L.; Collins, Allan: The cognitive structure of emotions. Cambridge, Cambridge University Press, 1988

[PaMi97] Pandit, Milind S.; Kalbag, Sameer: The Selection Recognition Agent: Instant Access to Relevant Information and Operations. In: Moore, J.; Edmonds, E.; Puerta, A. (eds.): Proceedings of the 2nd international conference on Intelligent user interfaces. New York, ACM Press, 1997, pp. 47-52

[PaRy98] Pascoe, Jason; Ryan, Nick S.; Morse, David R.: Human-Computer-Giraffe Interaction – HCI in the field. In: Johnson, C. (ed.): Workshop on Human Computer Interaction with Mobile Devices, May 21-23, Glasgow, Scotland. 1998, pp. 48-57

[PeWo92] Perry, Dewayne E.; Wolf, Alexander L.: Foundations for the Study of Software Architecture. In: ACM SIGSOFT Software Engineering Notes 17 (4), October 1992, pp. 40-52

[Pica97] Picard, Rosalind W.: Affective Computing. Cambridge, MIT Press, 1997

[Pica00] Picard, Rosalind W.: Toward Computers That Recognize and Respond to User Emotion. In: IBM Systems Journal 39 (3&4), 2000, pp. 705-719

[Pick00] Pickett, Joseph P. et al.: The American Heritage® Dictionary of the English Language, 4th edition. Boston, Houghton Mifflin, 2000

[PiKl02] Picard, Rosalind W.; Klein, Jonathan: Computers that Recognise and Respond to User Emotion: Theoretical and Practical Implications. In: Interacting with Computers 14 (2), 2002, pp. 141-169

[PiVy01] Picard, Rosalind W.; Vyzas, Elias; Healey, Jennifer: Toward Machine Emotional Intelligence: Analysis of Affective Physiological State. In: IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (10), October 2001, pp. 1175-1191

[RaBl79] Raab, Frederick H.; Blood, Ernest; Steiner, Terry O.; Jones, Herbert R.: Magnetic Position and Orientation Tracking System. In: IEEE Transactions on Aerospace and Electronic Systems AES-15 (5), September 1979, pp. 709-718

[ReHj99] Reynolds, Franklin; Hjelm, Johan; Dawkins, Spencer; Singhal, Sandeep: Composite Capability/Preference Profiles (CC/PP) - A user side framework for content negotiation. W3C Note. W3C, July 1999

[Reki96] Rekimoto, Jun: Tilting operations for small screen interfaces. In: Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology (UIST'96), November 6-8, Seattle, WA. ACM, 1996, pp. 167-168

[RhSt96] Rhodes, Bradley; Starner, Thad: The Remembrance Agent: A Continuously Running Automated Information Retrieval System. In: Proceedings of The First International Conference on the Practical Application of Intelligent Agents and Multi Agent Technology (PAAM '96), London, UK. April 1996, pp. 487-495

[RiSc01] Rigaux, Philippe; Scholl, Michel; Voisard, Agnès: Spatial Databases with Application to GIS. San Francisco, Morgan Kaufmann, 2001

[RoPi00] Roman, Gruia-Catalin; Picco, Gian Pietro; Murphy, Amy L.: Software Engineering for Mobility: A Roadmap. In: Finkelstein, A. (ed.): The Future of Software Engineering. ACM Press, 2000, pp. 241-258

[RyPa98] Ryan, Nick; Pascoe, Jason; Morse, David: Enhanced Reality Fieldwork: the Context Aware Archaeological Assistant. In: Gaffney, V.; van Leusen, M.; Exxon, S. (eds.): Computer Applications in Archaeology 1997, British Archaeological Reports. Oxford, Tempus Reparatum, 1998

[SaAb98] Salber, Daniel; Abowd, Gregory D.: The Design and Use of a Generic Context Server. In: Turk, M. (ed.): Proceedings of the Workshop on Perceptual User Interface (PUI '98), November 4-6, San Francisco, CA. 1998, pp. 63-66. Available at http://www.cs.ucsb.edu/conferences/PUI/PUIWorkshop98/Papers/Salber.pdf

[SaDe99] Salber, Daniel; Dey, Anind K.; Abowd, Gregory D.: The Context Toolkit: Aiding the Development of Context-Enabled Applications. In: Proceedings of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, May 15-20, Pittsburgh, PA. ACM, 1999, pp. 434-441

[Sand01] Sandkuhl, Kurt: Ein Referenzmodell für informationslogistische Anwendungen. In: Deiters, W; Lienemann, C. (eds.): Report Informationslogistik - Informationen just-in-time. Düsseldorf, Symposion Publishing, 2001, pp. 259-268 (in German)

[SaSc01] Saalmann, Armin; Schoon, Uwe: An Information Model for Information Demand Analysis concerning Information Logistics. In: Bauknecht, K.; Brauer, W; Mück, Th. (eds.): Informatik 2001 - Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit; Tagungsband der GI/OCG Jahrestagung 2001, September 25-28, Vienna, Austria. Vienna, Österreichische Computer-Gesellschaft, 2001, pp. 196-205

[ScAd94] Schilit, Bill N.; Adams, Norman; Want, Roy: Context-Aware Computing Applications. In: Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '94), December 8-9, Santa Cruz, CA. IEEE Computer Society, 1994, pp. 85-90

[ScAi99] Schmidt, Albrecht; Asante Aidoo, Kofi; Takaluoma, Antti; Tuomela, Urpo; Van Laerhoven, Kristof; Van de Velde, Walter: Advanced Interaction in Context. In: Gellersen, H.-W. (ed.): Proceedings of the First International Symposium on Handheld and Ubiquitous Computing (HUC '99), September 27-29, Karlsruhe, Germany. Springer, 1999, pp. 89-101

[ScBe99] Schmidt, Albrecht; Beigl, Michael; Gellersen, Hans-W.: There is more to context than location. In: Computer & Graphics 23 (6), December 1999, pp. 893-901

[Schm99] Schmidt, Albrecht: Implicit Human Computer Interaction Through Context. In: Personal Technologies 4 (2&3), June 2000, pp. 191-199

[Schm03] Schmidt, Albrecht: Ubiquitous Computing - Computing in Context. PhD Thesis, Lancaster University, 2003

[ScLe01] Schaal, Markus; Lenz, Hans-Joachim: Best Time and Content for Delay Notification. In: Bettini, C.; Montanari, A. (eds.): Proceedings of the 8th International Symposium on Temporal Representation and Reasoning (TIME '01), June 14-16, Cividale del Friuli, Italy. IEEE Computer Society, 2001

[ScTa00] Schmidt, Albrecht; Takaluoma, Antii; Mäntyjärvi, Jani: Context-Aware Telephony over WAP. In: Personal Technologies 4 (4), December 2000, pp. 225-229

[ScTh94] Schilit, Bill N.; Theimer, Marvin M.: Disseminating Active Map Information to Mobile Hosts. In: IEEE Network 8 (5), 1994, pp. 22-32

[ScVo89] Scholl, Michel; Voisard, Agnès: Thematic Map Modeling. In: Buchmann, A. P.; Günther, O.; Smith, T. R.; Wang, Y.-F. (eds.): Design and Implementation of Large Spatial Databases, Proceedings of the First Symposium SSD'89, July 17-18, Santa Barbara, CA. Springer, 1989, pp. 167-190

[ShGa96] Shaw, Mary; Garlan, David: Software Architecture – Perspectives on an Emerging Discipline. Prentice Hall, 1996

[StRu83] Stonebraker, Michael; Rubenstein, W. Bradley; Guttman, Antonin: Application of Abstract Data Types and Abstract Indices to CAD Data Bases. In: Proceedings of SIGMOD 1983: Engineering Design Applications. Silver Spring, IEEE Computer Society, 1983, pp. 107-113

[Sun04] Sun Microsystems: Java™ Remote Method Invocation Specification, Revision 1.10, Java™ 2 SDK, Standard Edition, v1.5.0. Santa Clara, Sun Microsystems, 2004

[TaKo01] Tang, Haitao; Korkea-aho, Mari; Takahashi, Kenji: Common Syntax and Coding for Descriptive Location. Internet Draft. IETF, May 2001. Available at http://www-nrc.nokia.com/ietf-spatial/draft-tang-spatial-descriptive-location-00.txt

[TaYa01] Tang, John C.; Yankelovich, Nicole; Begole, James; Van Kleek, Max; Li, Francis C.; Bhalodia, Janak R.: ConNexus to Awarenex: Extending awareness to mobile users. In: Proceedings of the SIG-CHI on Human factors in computing systems, March 31 - April 5, Seattle, WA. ACM, 2001, pp. 221-228

[ThAn02] Theodorakis, Manos; Analyti, Anastasia; Constantopoulos, Panos; Spyratos, Nikos: A theory of contexts in information bases. In: Information Systems 27 (3), 2002, pp. 151–191

[Thor01] Thorpe, Margaret: Business Rule Exchange - the Next XML Wave. In: Proceedings of XML Europe 2001, May 21-25, Berlin, Germany. Graphic Communications Association, 2001. Available at http://www.gca.org/papers/xmleurope2001/papers/html/s15-2.html

[Tous03] Toussaint, Alex (Specification Lead): Java Rule Engine API™ Specification. JSR-94, Version 1.0, September 15, 2003. San Jose, Bea Systems, Inc., 2003

[Vygo78] Vygotsky, Lev S.: Mind in Society - Development of Higher Psychological Processes. Cambridge, Harvard University Press, 1978

[WaCh04] Wang, Jue; Chen, Guanling; Kotz, David: A Sensor-fusion Approach for Meeting Detection. In: Workshop on Context Awareness at the Second International Conference on Mobile Systems, Applications, and Services (MobiSys 2004), June 6, Boston, MA. 2004. Available at http://www.sigmobile.org/mobisys/2004/context_awareness/papers/WangMeeting.PDF

[Wagn02] Wagner, Gerd: How to Design a General Rule Markup Language. In: Tolksdorf, R; Eckstein, R. (eds.): Proceedings of the Workshop XML Technologies for the Semantic Web (XSW 2002). Bonn, Gesellschaft für Informatik, 2002, pp. 19ff

[WaHo92a] Want, Roy; Hopper, Andy; Falcao, Veronica; Gibbons, Jonathan: The Active Badge location system. In: ACM Transactions on Information Systems 10 (1), January 1992, pp. 91-102

[WaHo92b] Want, Roy; Hopper, Andy: Active Badges and Personal Interactive Computing Objects. In: IEEE Transactions on Consumer Electronics 38 (1), February 1992, pp. 10-20

[WaJo97] Ward, Andy; Jones, Alan; Hopper, Andy: A New Location Technique for the Active Office. In: IEEE Personal Communications 4 (5), October 1997, pp. 42-47

[WaLa01] Warneke, Brett; Last, Matt; Liebowitz, Brian; Pister, Kristofer S. J.: Smart Dust: Communicating with a Cubic-Millimeter Computer. In: IEEE Computer 34 (1), January 2001, pp. 44-51

[WAP01] Wireless Application Protocol Forum: WAP Wireless Telephony Application, Version 08-Sep-2001. Wireless Application Protocol Forum, Ltd., 2001

[WAP01a] Wireless Application Protocol Forum: WAP Wireless Telephony Application Interface, Version 08-Sep-2001. Wireless Application Protocol Forum, Ltd., 2001

[WAP01b] Wireless Application Protocol Forum: Wirless Telephony Application Interface GSM-specific Addendum, Version 08-Sep-2001. Wireless Application Protocol Forum, Ltd., 2001

[WAP01c] Wireless Application Protocol Forum: Wirless Telephony Application Interface ANSI136-specific Addendum, Version 08-Sep-2001. Wireless Application Protocol Forum, Ltd., 2001

[WAP01d] Wireless Application Protocol Forum: Wirless Telephony Application Interface PDC-specific Addendum, Version 08-Sep-2001. Wireless Application Protocol Forum, Ltd., 2001

[WAP01e] Wireless Application Protocol Forum: Wirless Telephony Application Interface IS95-specific Addendum, Version 08-Sep-2001. Wireless Application Protocol Forum, Ltd., 2001

[WAP01f] Wireless Application Protocol Forum: WAG UAProf, Version 20-Oct-2001. Wireless Application Protocol Forum, Ltd., 2001

[WAP02] Wireless Application Protocol Forum: Class Conformance Requirements, Version 17-May-2002. Wireless Application Protocol Forum, Ltd., 2002

[Ward98] Ward, Andrew Martin Robert: Sensor-driven Computing. PhD Thesis, University of Cambridge, August 1998

[Wate96] Waterstraat, Jörn: Definition und Nutzung von Kontexten in mobiler, verteilter Datenbankumgebung. Diploma Thesis, University of Rostock, 1996 (in German)

[Wave01] Wavetrend Technologies Ltd.: User Manual for the L-RX200. Document number EAB-00700-03-UM, V 1.0, August 9. Wavetrend, 2001

[Webe92] Weber, Herbert: Die Software-Krise und ihre Macher. Berlin, Springer, 1992 (in German)

[Weis91] Weiser, Mark: The Computer for the 21st Century. In: Scientific American 265 (3), 1991, pp. 94-104

[Well86] Wells, David (ed.): Guide to GPS positioning. Canadian GPS Associates, 1986

[Wibe99] Wiberg, Mikael: Extending the modality of travelling. In: Käkölä, T. K. (ed.): Proceedings of IRIS22, Vol. 3. Jyväskylä, Jyväskylä University Printing House, 1999, pp. 49-58

[Wojc03] Wojciechowski, Manfred: Präsentationsunabhängigkeit von Informationen im Rahmen informationslogistischer Anwendungen. PhD Thesis, Technical University of Berlin, work in progress (in German)

[Worb92] Worboys, Michael F.: A generic model for planar geographical objects. In: International Journal of Geographical Information Systems 6 (5), September/ October 1992, pp. 353-372

[WrGl94] Wray, Stuart; Glauert, Tim; Hopper, Andy: The Medusa Applications Environment. In: Proceedings of the International Conference on Multimedia Computing and Systems, Boston, MA, May 1994, pp. 265-274. An extended version appeared in IEEE Multimedia 1 (4), Winter 1994. Also available as ORL Technical Report 94.3

[Wrig97] Wright, Ian Paul: Emotional Agents. PhD Thesis, University of Birmingham, School of Computer Science, 1997

[Zhao97] Zhao, Yilin: Vehicle Location and Navigation Systems. Intelligent Transportation Systems. Artech House, 1997

[ZhSh02] Zhao, Feng; Shin, Jaewon; Reich, James: Information-Driven Dynamic Sensor Collaboration for Tracking Applications. In: IEEE Signal Processing Magazine 19 (2), March 2002, pp. 61-72

Index

A

Activity theory 75
Affective Computing 37
Application 84
Architectural description (AD) 157
Attributes 74

C

Collaboration Data Objects (CDOs) 208
Common data set 44
Communication protocol 84
Composite Capabilities/Preference Profiles (CC/PP) 44
Context 11, 13
  Object model 47, 101
Context awareness 10, 11
Context builder 148
Context builders 148
  Object model 150
  Tasks 149
    Context data aggregation 150
    Context data filtering 149
Context Component 2, 157
  Architecture 157
    Dynamic viewpoint 163, 173
    External viewpoint 163, 164
    Logical viewpoint 163, 170
    Physical viewpoint 163, 187
    Structural viewpoint 163, 182
    Viewpoint interrelation 191
  Implementation 195
Context constraints 102, 151
Context data derivation 128
  Location 129
  Reachability 131
  State 129
    Activity 130
    Emotional condition 130
    Motion 129
    Physical condition 130
  Surroundings 131
Context Editor 199, 202
Context element 13
Context elements 47, 101
Context gatherer 150
Context gathering 105
Context sensor 105
Context sensors 106
  Assessment criteria 109
  Location 106
  Mapping table 114
  Reachability 109
  Sensor data transformation 113
  State 108
    Activity 108
    Emotional condition 109
    Motion 108
    Physical condition 108
  Surroundings 109
Context Service 41
Context SRML 132
Context Toolkit 34
Context-aware information logistic portal 196
Coordinate system 14
  Origin 14, 56
Coordinated Adaptation Platform 35
Coordinates 14

D

Dependency interface 172
Device 84
Distance 50, 59

E

E911 service 106
Enterprise Java Beans (EJB) 170
eXtensible HyperText Markup Language (XHTML) 96
eXtensible Markup Language (XML) 132

F

Framework for context-aware pervasive computing applications 38

G

Geographical Area Description 43
Geography Markup Language (GML) 43
Global Positioning System (GPS) 106

Global System for Mobile Communications (GSM) 106

I
IEEE 1471-2000 ..... 157, 164
Information demand 17
Information logistic application 9
Information logistics 7
    Framework 9, 158
Information overflow 7
Interdependence 102
ISO Reference Model for Open Distributed Processing (RM-ODP) 157
ISO/TC 211 Geographic Information/ Geomatics 43

J
Java 162

K
Knowledge eXchange System (KXS) 198

L
Location 14, 47
    Accuracy 57
    atomic 14
    Containment 61
    dynamic 55
    Equality 61
    geometric 54
    non-atomic 14
    Object model 47
    Operations 61
    Overlapping 62
    Prepositions 58
    Proximity 62
    symbolic 54
    Transformation 62
Location containers 48
    Location list 48
    Location set 49
    Location span 50
    Repeated location 50
Location structure 51

M
Messaging Application Programming Interface (MAPI) 208
Microsoft Exchange 208
Mobile Location Protocol 43

O
Object model 47
Operation execution services 62
Orientation 48, 74

P
Package 183
Pervasive computing 10

R
Radio Frequency Identification (RFID) ... 106
Reachability 15, 84
    Object model 84
Reference architecture 21
Remote Method Invocation (RMI) 169
Rule Markup Language (RuleML) 132

S
Sensor adaptor 117
Sensor adaptors 105, 117
    Object model 117
        Results 126
        Service description 123
        Service specification 123
Sensor Model Language (SensorML) 44
Sensor network 20
Sentient Computing 31
Simple Rule Markup Language (SRML) . 132
Situation 13
Software architecture 21
SOLAR 40
Spatial database system (SDBMS) 19
State 15, 70
    Activity 75
    Emotional condition 79
    Motion 70
    Object model 70
    Physical condition 78
Surroundings 16, 99
    Object model 99

T
Technology for Enabling Awareness (TEA) 32

U
Ubiquitous computing 10
Unified Modeling Language (UML) 47

V
Viewpoint 163
Virtual sensor 127
Virtual sensors 126
    Object model 138
        Results 143
        Virtual sensor creation 140
        Virtual sensor implementation . 143
    Rules for context data combination and derivation 131
    Tasks 127
        Context data combination 127
        Context data derivation 128
        Context data pre-aggregation . 127

W
WAP browsers 95
Wavetrend location sensor 206
Wireless Application Protocol (WAP) 84