Context Management and Personalisation:
A Tool Suite for Context- and User-Aware Computing

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Dipl.-Inform. Andreas Zimmermann

aus Ludwigshafen am Rhein

Berichter:

Prof. Dr. Matthias Jarke

Prof. Dr. Reinhard Oppermann

Tag der mündlichen Prüfung: 23.10.2007

Vorwort des Institutsleiters

Schon seit über fünfzehn Jahren befasst sich das Fraunhofer-Institut für Angewandte Informationstechnik FIT mit Fragen der Anpassung von Informationssystemen an ihre Benutzer und an die Benutzungskontexte. Dies kann zunehmend nicht mehr vollständig in einer anfänglichen Anforderungsanalyse erfasst werden, sondern die Anpassungsnotwendigkeit ergibt sich oftmals erst im laufenden Betrieb. Sowohl die Anpassbarkeit von Systemen durch den menschlichen Benutzer (Adaptierbarkeit) als auch die automatische Anpassung etwa aufgrund der Beobachtung des Benutzerverhaltens (Adaptabilität) sind dabei Untersuchungsgegenstand.

Die Einführung mobiler Kleingeräte – das Mobiltelefon ist ein typisches Beispiel – und moderner Sensorik – wie etwa RFID und WiFi Positionierung – haben diesen Bemühungen im beginnenden 21. Jahrhundert neue Aktualität verliehen und die Herausforderungen wesentlich verbreitert. Stand vorher eine relativ statische Anpassung an eine bestimmte Person oder standardisierte Benutzungssituation im Vordergrund, so werden in mobilen Anwendungen ständig Kontextwechsel vollzogen. Auch die eigene Benutzungshistorie selbst spielt bei der Anpassung eine Rolle – so will der Museumsbesucher vielleicht beim zweiten Besuch eines Exponats nicht noch einmal genau die gleiche Erklärung hören wie beim ersten Mal.

In einer Vielzahl von Projekten hat Fraunhofer FIT mit derartigen Anforderungen Erfahrungen gesammelt und erfolgreiche Einzellösungen gestaltet. Dabei entstand zwangsläufig die Frage, ob es nicht möglich ist, die vielfältigen Dimensionen von Personalisierung und Kontextanpassung in einem einheitlichen Modellierungsrahmen zu erfassen und durch einen einheitlichen Werkzeugkasten zu unterstützen. Eine Lösung für dieses Problem vorzustellen ist Gegenstand des vorliegenden Buches.

Im Gegensatz zu bisherigen Ansätzen für Kontext-Werkzeugkästen, von denen der bekannteste am Georgia Tech entstanden ist, beschränkt sich die Unterstützung nicht auf die Gruppe der Systementwickler, sondern bezieht all diejenigen ein, die an der nachträglichen Anpassung eines Systems ein genuines Interesse haben: die Domänenexperten, welche das Anwendungswissen strukturieren, die Autoren von Informationsinhalten und vor allem auch die Endbenutzer selbst. Zusätzlich zu einer systematisch strukturierten Schichtenarchitektur mit den Hauptebenen Sensorik, Semantik, Kontrolle und Aktuatorik entstehen so spezielle Werkzeugangebote, die dem Bedarf dieser Gruppen jeweils speziell angepasst sind.

Der Werkzeugkasten entstand in einem iterativen Designprozess über mehrere Jahre und wurde immer wieder im Kontext unterschiedlicher Anwendungen validiert und weiter verbessert. Hohe Sichtbarkeit erzielte dabei vor allem die Anwendung im EU-Projekt LISTEN, in dem gemeinsam mit dem Fraunhofer IMK (jetzt Teil des IAIS) beispielsweise eine Ausstellung des Malers August Macke mit einem differenzierten Audioraum ermöglicht wurde. Ebenso fand eine interaktive Plakatwand, die sich automatisch oder benutzergesteuert der Situation etwa in einem Bahnhof anpasst, großes Besucherinteresse auf der CeBIT 2004. Auch bei der multimodalen kontext-adaptiven Unterstützung von Lagerarbeitern im Projekt MICA – Teil des Leitexponats „Future Factory“ der SAP auf der CeBIT 2007 – fand der Werkzeugkasten Anwendung. Der vorgestellte Modellierungsansatz und seine Unterstützung durch den Werkzeugkasten erweist sich damit als außerordentlich erfolgreich und in einer Vielfalt von Kontexten anwendbar.

Das vorliegende Buch entstand im Rahmen des Promotionsvorhabens von Andreas Zimmermann am Lehrstuhl für Informationssysteme der RWTH Aachen; Korreferent war Prof. Dr. Reinhard Oppermann, Universität Koblenz und Fraunhofer FIT. Es bietet dem Leser nicht nur eine genaue Darstellung des eigenen Konzepts, dessen Umsetzung und Validierung, sondern auch einen guten Überblick über die verwandten Problemstellungen und Lösungsansätze der Forschung in diesem weltweit hochaktuellen Bereich.

Aachen und Sankt Augustin, im November 2007

Prof. Dr. Matthias Jarke

Institutsleiter, Fraunhofer FIT

Abstract

Rapidly changing requirements and dynamic environments drive the development of context-aware applications. Research into context-aware computing focuses on programming frameworks and toolkits that support the development of context-aware applications (Chen, 2004; Dey et al., 2001; Efstratiou, 2004; Henricksen and Indulska, 2006). However, current approaches emphasise developers as the main actor in the software development process and lack properties making context-aware computing transparent and applicable for other actors. Developers cannot anticipate all potential situations and all possible ways of application behaviour during the development phase. During runtime, a change in the configuration, structure or content of the developed prototype is difficult.

This work claims that the extension of the spectrum of actors participating in the design, implementation, authoring and configuration of context-aware applications beyond developers substantially tackles the reduction of usability problems introduced by context-aware computing.

Addressing these issues requires this work to accomplish both a conceptual and a software framework. The conceptual framework bases on a comprehensive understanding of the processes involved with context-aware computing in general that can be communicated to the entire spectrum of actors comprising developers, domain experts, authors and end-users. The software framework implements the conceptual framework and supports the actors on diverse implementation skill levels in their roles within the software development cycle for context-aware applications. The core contribution of this work comprises a design view of context-aware applications, which permits the control over the internals of the application during design- and runtime, and a Context Management System, which provides different tools and abstraction levels according to the actors' roles within the software development process.

Two case studies document and evaluate the application of the system for the construction, authoring, maintenance and tailoring of context-aware applications and their behaviour. As operational and adaptable context-aware applications, these case studies prove the validity and general applicability of the tool suite, the software architecture and the concepts behind.

Kurzfassung

Sich ständig verändernde Anforderungen und dynamische Umgebungen treiben die Entwicklung von kontextsensitiven Anwendungen voran. Die Aufmerksamkeit des Forschungsgebiets des Context-Aware Computing richtet sich auf Programmierumgebungen und Werkzeuge, die eine Unterstützung bei der Entwicklung kontextsensitiver Anwendungen bieten (Chen, 2004; Dey et al., 2001; Efstratiou, 2004; Henricksen and Indulska, 2006). Allerdings stellen derzeitige Ansätze den Entwickler als Hauptakteur im Software-Entwicklungsprozess heraus und verzichten auf Eigenschaften, die Context-Aware Computing für andere Akteure transparent und anwendbar macht. Entwickler können während der Entwicklungsphase nicht alle potentiellen Situationen und alle Möglichkeiten des Verhaltens der Anwendung antizipieren. Zur Laufzeit erweist sich eine Veränderung der Konfiguration, Struktur oder des Inhalts der entwickelten Anwendung als schwierig.

Die vorliegende Arbeit verfolgt den Ansatz, das Spektrum der am Software-Entwicklungsprozess kontextsensitiver Anwendungen beteiligten Akteure über den Entwickler hinaus zu erweitern, um eine substantielle Reduktion der Gebrauchstauglichkeitsprobleme zu erzielen.

Zur Verwirklichung dieses Ansatzes entwickelt diese Arbeit sowohl ein konzeptuelles, als auch ein software-technisches Framework. Das konzeptuelle Framework basiert auf einem weitreichenden Verständnis der Konzepte und Prozesse aus dem Bereich Context-Aware Computing, das sich an das gesamte Spektrum von Akteuren kommunizieren lässt, bestehend aus Entwicklern, Domänenexperten, Autoren und Endbenutzern. Das software-technische Framework implementiert das konzeptuelle Framework und begleitet die Akteure mit ihrer unterschiedlichen Qualifikation als Entwickler durch den Software-Engineering-Prozess kontextsensitiver Anwendungen. Der wesentliche Beitrag dieser Arbeit besteht aus einer Entwurfssicht auf kontext-sensitive Anwendungen, die eine Kontrolle über die Interna der Anwendung zur Design- und Laufzeit ermöglicht, und einem Kontext-Management-System, das aus unterschiedlichen Werkzeugen besteht und verschiedene Abstraktionsebenen bietet, die zu den Rollen der Akteure im Prozess der Softwareentwicklung passen.

Zwei Fallstudien dokumentieren und evaluieren die Anwendung des Systems bezüglich der Konstruktion, Inbetriebnahme, Administration und Anpassung kontextsensitiver Anwendungen und ihres Verhaltens. Als funktionsfähige und anpassbare kontextsensitive Anwendungen zeigen diese Fallstudien die Gültigkeit und die allgemeine Anwendbarkeit der Werkzeuge, der Softwarearchitektur und der zugrundeliegenden Konzepte.

Acknowledgements

In addition to the physical traces my thesis left on paper, white boards, and hard drives, the entire creation process has influenced friends, colleagues, researchers, family and myself. For the future, I hope that the contents of my thesis is going to leave further traces and impressions on many people's minds.

My employment at the Fraunhofer Institute for Applied Information Technology in Sankt Augustin has provided a space of creativity for me. I took advantage of the situation, arranged my initially quite disordered ideas, and I have compiled the essence of them into a hopefully valuable contribution to this lively research area. However, this work would have never been possible without the help of many people.

I express my gratitude to my advisors, Prof. Matthias Jarke and Prof. Reinhard Oppermann, for giving me the opportunity to complete my Ph.D. Your constructive criticism and collaboration have been enormous drivers to improve my work. Thank you for putting so much faith in me and providing me with direction and temporal constraints.

Three sources of inspiration and energy accompanied the process of working on my Ph.D.: I am grateful to my colleagues, mentors, friends Marcus Specht, Andreas Lorenz and Markus Eisenhauer, who offered a creative atmosphere at the office and helped me form my understanding of context in endless discussions. I received a lot of input and ideas from you in the early phase of my work and your reviews lead to substantial improvements of this work. I have learnt so much from all of you.

Andreas Lorenz deserves special thanks because sharing an office with me has without doubt been quite demanding for you from time to time. You are the confidant I discuss my troubles and complaints with. Nevertheless, you have supported me all the years and have become such a great friend. Laughing and yelling with you kept me alive in hard times, and hopefully, I can clear debts soon.

I have been also very fortunate to work in a great group of colleagues. Thank you, Alexander Schneider, Stefan Apelt, Christian Prause, Marius Scholten, Oliver Kaufmann, Lars Zahl, Barbara Schmidt-Belz and Rossen Rashev for creating such a supportive, creative and friendly research environment. What would I have done without all the chats, laughter and shared lunch experiences?

I would like to apologise to my best friends Bernhard Rausch, Christoph Kiener, Boris Schwerdt, Markus Reinhart, Stefan Nessler, André Cambeis and Barbara Dellen about the

lack of time during the past year. I especially want to thank Barbara Dellen, who insistently encouraged me to stick to my plans and kept me going further. Each time I finished a chapter I remembered your severe bets on keeping deadlines.

Finally, I dedicate this work to my family, who believed in me throughout my entire life. I am particularly grateful to my parents, Ellen and Heinz, who have given me strength, endless support and love from my childhood on. I would have never made it through the tough times in my life and I would have never come as far as I have without you. I am forever in your debt.

Bonn, June 14th, 2007

Andreas Zimmermann

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent trends towards computing paradigms such as ubiquitous and pervasive computing claim for moving computing off the desktop and into the environment in order to create a more natural appearance of the computer (Weiser, 1991). The computer as a tool disappears from the centre of the user's attention and the human-computer interaction moves beyond the desktop into the real world. In contrast to the desktop, which constitutes a well-known and well-controlled environment, the real world exposes complexity and dynamics. The challenge in ubiquitous and pervasive computing lies in the creation of usable applications and services that are functional in all those manifold situations emerging in the real world.

Changing requirements and dynamic environments are drivers for context-aware applications because they are highly autonomous and responsive to changes in the context of use and in the user's demands. Context constitutes a powerful concept in human-human and human-computer interaction because implicit context information allows for the interpretation of explicit activities. The objective of context-aware applications consists in the assistance of the users by pro-actively supplying what is actually relevant and needed with respect to the current situation. Thus, such an application enhances the quality of system usage through adapting aspects like the supplied information, functionality or presentation.

Although many prototypes have proven the potential of context-aware applications, they have also revealed that the design, development and maintenance of this application type constitute crucial challenges. The implementation of context-aware applications demands the consideration of three main functional areas: context acquisition, context synthesis, and context use. The increasing complexity of the three areas in combination with the need for a rapidly decreasing time to market entry made the existence of supporting tools indispensable. Therefore, most of the research into context-aware computing focuses on developing frameworks and toolkits to assist developers in building context-aware applications (Chen, 2004; Dey et al., 2001; Efstratiou, 2004; Henricksen and Indulska, 2006).

1.1 Problem Description

Programming frameworks and toolkits ease the creation of context-aware applications for developers through guiding the software engineering process and hiding complex underlying technical details. The supplementary application of a user-centred design process yields to a richer understanding of the context of use and guarantees a certain degree of user acceptance. However, some types of modern context-aware applications require instant adaptation due to their exposure to increasing situational dynamics. The operational environment will change, the tasks will be distinct, the end-users will be heterogeneous, and their competences and expectations will evolve.

Even if the adaptations automatically performed by the implemented context-aware application are very desirable in many cases, they always represent the perspective of the developer. The importance of involving and empowering other roles and actors in the design, development and operation of context-aware applications has often been neglected in current programming toolkits and infrastructures. In fact, designers, product managers, authors or end-users exhibit more in-depth knowledge about the required application behaviour than any developer.

The developer can only draw an image of the deployed context-aware application because on-site conditions and information are often unavailable during the development phase. During runtime it can be unclear how the developed prototype will react and usually it will be difficult or impossible to change its configuration. In addition, it is impossible for developers to anticipate all potential situations and all possible ways of application behaviour. For the user of such an application this bears the risk of getting into situations, in which the context-aware behaviour implemented by this developer is inappropriate, undesired or even embarrassing or dangerous, and the automatically performed adaptations potentially cause user discomfort.

Without control of the application the user becomes a passive recipient of automatic mechanisms. Automatically performed adaptations of the application are only possible to the extent that the system is able to gather the required information basis of performing some adaptation function. Context information may be unreliable, inaccessible, difficult to acquire or results from an interpretation process may take very little recorded operation steps into account. In addition, most context-aware applications limit the modelling of the user's context to aspects like the location and the physical environment and abandon the explicit modelling of users themselves like their preferences, goals, and intentions, or their cognitive and emotional states.

The external context alone may inadequately determine the most appropriate adaptation to the individual user. A combination of user models and context models would empower designers of context-aware applications to increase the application's ability to adapt to the user. However, the potential for designing an application that performs the wrong action and seriously annoys the user still persists. In addition, the lacking transparency of context-aware

applications makes the adaptation decisions inaccessible to the end-user and disallows an overriding of the behaviour. Without control and means of customization the end-users will abandon useful context-aware services.

Consequently, context-aware applications are associated with a number of typical usability problems. In some cases a decrease of the system's usability outweighs the benefits of adaptation. The overall goal is to ensure an adequate fulfilment of the usability goals without eliminating the benefits of the adaptation processes. The anticipation and prevention of usability side effects should form an essential part of the design of context-aware applications.

1.2 Thesis Statement

This thesis contends that current development support for context-aware applications emphasises developers as the main actors in software development processes of such applications. Furthermore, current approaches lack properties that make context-aware computing transparent and applicable for everyone.

The thesis claims that the extension of the spectrum of actors participating in the design, implementation, authoring and configuration of context-aware applications beyond developers substantially tackles the reduction of usability problems introduced by context-aware computing. The thesis proposes that context-aware applications need to provide mechanisms where their context-aware behaviour can be reconfigured without the need for reimplementation.

The thesis contributes a novel and comprehensive understanding of context-aware computing, a conceptual framework with an associated software architecture and a corresponding tool suite, which support users at different development skill levels in the realization of context-aware behaviour. At the core of this contribution are a design view of context-aware applications, which permits the control over the internals of the application during design-time and runtime, and the integration of contextualization and personalization.

1.3 Research Methodology

The involvement of a multitude of actors in the software engineering process of context-aware applications accompanies both a conceptual and a software framework. The conceptual framework bases on a comprehensive understanding of the processes involved with context-aware computing in general that can be communicated to the targeted group of people. The software framework implements the conceptual framework and guides the actors on diverse programming skill levels through the software development cycle for context-aware applications. In order to accomplish both frameworks, the research presented in this thesis

comprises theoretical and practical parts. Besides the concrete problem definition and a broad analysis of existing approaches, this work follows a research methodology in five stages:

Requirements

Based on the results of the literature study of existing approaches towards development support for context-aware applications, requirements are derived that address the extension of this development support to involve other actors than developers. These requirements guide the definition of the conceptual framework.

Conceptualization

The deepened analysis of context-aware applications from different domains results in a comprehensive understanding of the processes, results, tasks and roles of actors involved in the construction, integration, authoring, administration and tailoring of context-aware behaviour. The derived concepts need to be understood by the entire spectrum of actors ranging from developers to end-users.

Design

The derived conceptual framework is formalized to an extent, to which software and tools can be applied for support. Therefore, the components of the conceptual framework are mapped to a generic software architecture for context-aware applications that supports as much different applications as possible. This software architecture needs to be methodology- and philosophy-neutral, as well as sustainable.

Prototyping

The prototypical implementation of a Context Management System and the decomposition of this system into its foundational components and tools need to prove the validity of the software architecture and the concepts behind. The resulting system offers a tool suite providing different abstraction levels for users on several programming skill levels and an open platform for the easy development and smart maintenance of context-aware applications and information services. In addition, an initial set of software components needs to be available to enable the development of applications that automatically adapt to the users and to changes in the context of use.

Proof of Concept

As a proof of concept for the general applicability of the software architecture and the utility of the tool suite, two exemplary case studies are implemented. These case studies are operational and adaptable context-aware applications that sense their environments, construct a model of the user's context, and adapt their behaviour according to changes in the context. These implementations depict the advantages and drawbacks of the approach proposed by this thesis.

1.4 Thesis Outline

The organization of the remainder of this thesis divides as follows:

Chapter 2 Chapter 2 introduces and defines the basic terms used throughout this thesis and presents the background of the fundamental concepts. In course of this chapter, the most common related definitions elaborated by the research community are examined and reasons for studying context-aware applications are motivated.

Chapter 3 Chapter 3 outlines the research framework of this thesis and surveys relevant literature. It identifies current research directions and challenges and investigates the state of the art regarding programming toolkits and software infrastructures that aim at facilitating software engineering processes of context-aware applications for non-developers.

Chapter 4 Chapter 4 provides the theoretical foundation for the derivation of a universally applicable software architecture for context-aware applications as a prerequisite for facilitating the development and maintenance of such applications. Based on this theoretical foundation, the chapter introduces a four-layer architecture of context-aware applications.

Chapter 5 While Chapter 4 elaborated the conceptual framework and derived a software architecture for context-aware applications, Chapter 5 characterizes the concrete implementation of this architecture as part of a Context Management System. This system comprises functionality for the facilitated construction, integration, authoring, administration and tailoring of context-aware behaviour.

Chapter 6 The validation of the universal applicability of the conceptual framework and software architecture defined in Chapter 4 takes place in Chapter 6. This chapter demonstrates the application and utility of the Context Management System developed in Chapter 5 and describes the realization of two case studies in detail: a museum guide and an intelligent advertisement board.

Chapter 7 Chapter 7 summarises the contributions of this thesis and suggests several possible research topics for future work that can base on the presented concepts.

Chapter 2

Background and Definitions

This chapter aims at providing a comprehensive understanding of important terms that coin the research area of *context-aware computing* and recur throughout this thesis. In addition, the sections of this chapter convey a sense of the application of these relevant terms and present the background of the fundamental concepts of this research area. The introduction to this chapter reveals the need for adaptations by means of an example application scenario of a context-aware application. In the following, the notion of the term *context* is highlighted for different research areas that exhibit some alternative views of context. The examination of the most common context definitions provided by the research community directs this variety of different meanings of context to an interpretation of this term that is used within the scope of this thesis. The definition of context comprises three canonical parts: a definition per se in general terms, a formal definition describing the appearance of context and an operational definition characterizing the use of context and its dynamic behaviour. The chapter concludes with a closer examination on usability issues introduced by context-aware computing. The discussion on critical usability issues leads to the introduction of the term *adaptable context-aware application* and serves as a source of key considerations in the design of the tool suite presented in succeeding chapters.

2.1 The Need for Adaptation

A software system passes through a potentially long software engineering cycle (for further reading cf. Booch et al. (1999) or Gilb (1998)) and before delivery requirement engineers, designers and developers realize the components of the system. However, it is impossible to anticipate the requirements of all users, and a single best or optimal system configuration is impossible. The *active involvement of users* and clear understanding of user and task requirements is a challenge in the development of computer-based interactive systems for two reasons: first, the potential user groups are not known a priori, but need to be identified according to future scenarios; these groups need to be revised as the visions evolve because

there may be various groups of potentially affected users. Second, the visions of the aspired project are far-sighted and not close to users' current experiences; therefore, users may not be confident and precise about their needs concerning this future system. With their norm for "Human-centred design processes for interactive systems" (ISO13407, 1999) the *International Organization for Standardization* gives guidance on user-centred design activities throughout the life cycle of computer-based interactive systems. One of the core tasks of user-centred design is to negotiate and facilitate the communication across the well-known user-developer gap while acknowledging the different forms of expression and different requirements on each side. However, despite the implementation of a human-centred design process, some types of modern applications require instant adaptation due to their exposure to increasing situational dynamics. The following example applications scenario illustrates this need for adaptation.

2.1.1 Example Application Scenario

The consideration of the context is of importance in a variety of modern application domains. In each of these application domains context plays one or more specific roles in order to support the user of a computing system to accomplish a task. Context-awareness has been widely studied in the past years and context-aware prototypes have emerged from different research areas. The following example application scenario provides an extract of prevalent domains, for which context-aware applications have been developed, and illustrates the need for the automatic and manual adaptation of computer applications. Furthermore, this example application scenario pervades this thesis and serves as a means of illustration of achieved concepts.

> "In the afternoon, after a hard working day, Carmen arrives early at the Steigenberger Hotel in Berlin in order to meet potential customers. Prior to this meeting she has an appointment with her boss, Anne, who wants to speak about the strategy to pursue during the meeting, and therefore she books a hotel room for an hour. As she enters her room, it adopts her 'personality' and automatically adapts the room temperature and the default lighting according to her preferences. The lighting appears a bit too bright to Carmen and thus, she slightly adapts the light levels using her smart phone as a remote control.
>
> Because Anne seems to be late, Carmen decides to sit down on the chair near the table and create some relaxing atmosphere. She uses the voice command "music" that initiates her smart phone to assemble a list of her favourite songs. According to Carmen's listening history, interests and mood her smart phone filters the music database and displays the selection on the room's video wall. Carmen is not fully convinced by the list the system provided and decides to examine the system decision. Using her smart phone as a remote control, Carmen is able to acquire and browse through a rich visualisation of what the system learnt about her on the video wall of the hotel room. She realises that the system assumes she would like rock 'n'

roll in stressful moments, and so she decides to correct this. After some manual modifications of the system, Carmen receives a revised music recommendation.

Carmen works for Anne since two years now and already knows her habit of conducting focussed and goal-oriented meetings. In the case of such a meeting taking place Carmen has manually configured her smart phone to automatically switch to a silent meeting mode. This is why the smart phone immediately stops the music and automatically raises the communication access thresholds to block out anything but emergency messages as Anne knocks on the door and enters the room. Without losing much time Carmen holds her smart phone near the room's video wall, which starts to display the presentation. Carmen and Anne have half an hour of intensive discussion and decide to make some changes to their presentation. Anne suggests integrating some images of a company-internal report they both have read recently. Carmen trusts her boss in this regard and copy-pastes the images on additional slides.

Well prepared for the meeting they come out of the hotel room and go downstairs in the hotel's seminar room where the customers were already waiting. The meeting is rough, but Carmen feels it was a success. The day has been long and stressing, and as Carmen drives home, she really feels tired. To make matters worse, her Global Positioning System (GPS) device runs out of power and thus, her navigation system fails. Because she feels unfamiliar with this particular area of the city, she needs to use a map to get on the motorway. After a while she finally enters the motorway feeling even more tired. Suddenly, the car crash protecting system detects a growing fatigue of Carmen for longer than two seconds. The alerting system temporally turns up the volume of the car radio in order to shake her up a bit. This warning makes the desired impact on Carmen and forces her to drive home more carefully."

This example application scenario highlights the diversity of possible adaptations context-aware applications might perform. In addition, it points to the multiplicity of sources of information about the user and her environment such applications base their adaptation decisions on. Furthermore, the example application scenario conveys the need for a situation-specific tailoring of automatic mechanisms. The following subsections further investigate the need for adaptation.

2.1.2 Adaptivity and Adaptability

Even if the user-centred design process implemented in a project guarantees a certain degree of user acceptance and yields a richer understanding of the context of use, the completed product's ability to adapt to changing conditions still plays a central role for a broad acceptance. The operational environment will change, the tasks will be distinct, the end-users will be heterogeneous, and their competences and expectations will evolve. Here again it is impossible for developers to anticipate all possible requirements modifications. As the example application scenario in the previous section shows, the dynamics of changing conditions shifts the customisation process of the system's characteristics from the

development phase to its usage and operation phase because the time needed for a professional development is too short or the new features are too costly.

For this reason, developers implement techniques of adaptation into the system in order to react to changing conditions as fast as possible. The example application scenario clearly shows an important distinction concerning such adaptation techniques: the differentiation between manually and automatically performed adaptation processes. Accordingly, the term *adaptation* decomposes into the two terms *adaptivity* and *adaptability*. Adaptivity indicates a system that adapts automatically to its users according to changing conditions, i.e. an *adaptive system*. Adaptability refers to users that can substantially customise the system through tailoring activities by themselves, i.e. an *adaptable system*. This distinction can be more fine-grained and complemented by activities dedicated to the specific actors in the entire adaptation process (see Section 4.5).

The aim of adaptivity is to have systems that adapt themselves to changes in user-related or environmental properties. Such automatically performed adaptations base on the evaluation of the user behaviour and assumed user needs, or taking explicit user input into account. The objective is to assist the users by pro-actively supplying what is actually needed. Adaptive systems try not to distract users from their primary task by searching and selecting information or services, as well as by performing extensive customisation tasks. The example application scenario described above illustrate that such systems automatically adapt to situations where computers outperform the user (e.g. faster response time, identification of toxic gases) or the user is unable to perform a specific task and the computer takes over control (e.g. unsuited clothes, cognitive overload).

The aim of adaptability is to empower end-users without or with limited programming skills to customise or tailor computer systems according to their individual or environment-specific requirements. Such adaptable systems allow for fast adaptations to dynamically changing requirements by letting the end-users put their domain-specific expertise to the task of system customisation. After the user identified a change of her needs, she is enabled to manually adapt the system's features. This approach provides adaptation methods and tools that are under the control of the user instead of the system. Therefore, adaptable systems enable users to override certain functions or correct decisions if the system's model of the user or the environment and the reality mismatch (e.g. refinement of preferences, change modality).

Adaptive and adaptable systems are complementary to each other (Oppermann, 2005). Both methods increase the match between user needs and system behaviour once the development of the system has been finished. Thus, the system is kept flexible during usage.

2.1.3 Catalysts for Adaptation

The example application scenario described above shows that changing conditions trigger the execution of an adaptation. Many characteristics might be taken into account as catalysts for such an adaptation process. They can be clustered into three main categories: inter-individual, intra-individual and environmental differences.

Inter-Individual Differences address varieties among several users along manifold dimensions. Physiological characteristics like disabilities are of major concern for application designers if they want to have their system accepted by a large community. The consideration of user preferences like language, colour schemes, modality of interaction, menu options or security properties, and numberless other personal favourite preferences is one of the most popular sources of adaptation and can be reused in different applications. An important prerequisite for the content adaptation is the awareness of the user's interests and disinterests. The user's interests configure filters for the information presented to the user. Additionally, a broad spectrum of psychological personality characteristics exists like emotions, self-confidence, motivation, beliefs or idols, which are difficult to assess automatically. The same holds for the user's level of factual, general and domain-specific knowledge (e.g. beginners or experts), which is a valuable source of adaptive operations.

Intra-Individual Differences consider the evolution and further development of a single user, as well as the task over time. A static system falls short of changing user requirements as the user's activities and goals evolve. In an extreme case users are overstrained by the system in the beginning and perceive the same system as cumbersome and restricted as the user's expertise increases. In the same manner, the need for a higher flexibility of computer systems is pushed by the changing of the tasks to be accomplished with such a system.

Environmental Differences result from the mobility of computing devices, applications and people, which leads to highly dynamic computing environments. Unlike desktop applications, which rely on a carefully configured and largely static set of resources, ubiquitous computing applications are subject to changes in available resources such as network connectivity and input and output devices. Moreover, they are frequently required to cooperate spontaneously and opportunistically with previously unknown software services in order to accomplish tasks on behalf of users. Thus, the environment surrounding an application and its user is a major source to justify adaptation operations.

Inter- and intra-individual differences particularly refer to adaptation effects based on changes in characteristics of the users. Jameson (2003) defines all systems that automatically perform an adaptation to the individual user in some nontrivial way as *user-adaptive systems*.

Definition 1: User-Adaptive System

A user-adaptive system is an interactive system that adapts its behaviour to individual users on the basis of processes of user model acquisition and

> *application that involve some form of learning, inference, or decision making.* (Jameson, 2003)

User-adaptive systems must be able to observe the user's behaviour, generalise these observations and make assumptions about the user. The information about the user is usually collected in a so-called *user model* and administrated by a *user modelling component*. Wahlster and Kobsa (1989) define these two fundamental concepts as follows:

Definition 2: User Model

> *A user model is a knowledge source in a system which contains explicit assumptions on all aspects of the user that may be relevant to the behaviour of the system. These assumptions must be separable by the system from the rest of the system's knowledge. (Wahlster and Kobsa, 1989)*

Definition 3: User Modelling Component

> *A user modelling component is that part of a system whose function is to incrementally construct a user model; to store, update and delete entries; to maintain the consistency of the model; and to supply other components of the system with assumptions about the user.* (Wahlster and Kobsa, 1989)

These two definitions emphasise dialog systems and demand for a detailed, explicit model of the user to adapt the behaviour of the system. User modelling components are constructed in such a way that they are tightly intertwined with the application. Several generic user modelling servers support this kind of adaptive technology through facilitating the provision of user modelling services to application systems (Kobsa, 2001). In this connection, Fink (Fink, 2004) presents research on the requirements, design, and evaluation of user modelling servers.

Prominent functions for user-adaptive systems are product recommendation, interface adaptation, learner support, or information retrieval (cf. Kröner (2001)). The e-commerce sector prefers the more popular term *personalization* focusing on systems that tailor products, services and information to better cater to each individual user. Their primary driver for using personalization is the belief that they can establish a stronger relationship with a customer through treating them individually (see the *Amazon Web Services* at Amazon (2007) or BroadVision (2007) as examples).

This section primarily discussed user-related differences that catalyse adaptation processes. If now environmental differences are integrated with inter- and intra-individual differences into a coherent whole, the notion of context becomes apparent. The comprehension and definition of context and related terms will be the main focus of the remaining sections of this chapter.

2.2 The Notion of Context

The notion of context varies across many different research areas. In general, context is understood as a knowledge type that facilitates a selective and focused processing of information. The following subsections illustrate the role of context in human-human dialogue, computer science and ubiquitous computing.

2.2.1 Context in Human-Human Dialogue

In using a rich language, humans produce information intended to be interpreted by one or more other people. The words of a sentence carry meanings. However, humans are able to use implicit information, the context of a conversation, additionally. The notion of context might cover a wide range of issues such as the common history of the participants engaged in communication, the social setting, the culture, body postures, the choice of specific words etc.

Through a common understanding of the world and an implicit understanding of everyday situations, humans enhance the conversational bandwidth of a dialogue. Some words cannot even be determined out of context. An audience can only construct an appropriate meaning of the speaker's intention through inferences based on context. For the humans, the awareness of the context is an essential capability for understanding the implicit information that is associated with the activities that they conduct (Chen, 2004).

Context emerges in dialogue and is only effective when it is shared among the communication partners (Winograd, 2001). Clarke and Cooper (2000) mention shared contexts that are composted of a shared understanding and a shared environment. During a conversation, humans utilise various instruments to generate a common ground (Clark, 1996) as a shared interpretation of the context around them. The context becomes animated by the process of its interpretation.

2.2.2 Context in Computer Science

In research subjects related to computer science context experiences a broad application; however, the notion of context depicts variations and alternative perspectives. In natural language processing context is a means of disambiguating the meaning of utterances (Lenat, 1998). Sources for this process are the linguistic context, which encompasses words of the sentence, sentences themselves, and the context, which includes information about the speaker and the environment.

In artificial intelligence context arises in assorted areas like knowledge representation, machine learning, and intelligent information retrieval. Furthermore, it contributes to feature selection in order to achieve an increased efficiency in reasoning (Lieberman and Selker, 2000). The context is what does not intervene directly in a problem solving but constrains it (Brézillon, 1999). Guha (1995) introduces context as a knowledge base for solving problems in artificial intelligence and presents a formalisation of the term context in this field. The

disciplines *cognitive psychology* and *artificial intelligence* are intimately connected with each other because artificial intelligence utilises models offered by cognitive psychology. A broader discussion of this topic is available in Öztürk and Aamodt (1998).

The main reason for studying *formal* context in logical reasoning is to resolve the "problem of generality" in artificial intelligence. McCarthy and Buvac (1993; 1994) understand context as a "generalization of a collection of assumptions" because an algorithm often necessitates a partial redesign with changing conditions. He investigates on a form of reasoning within contexts using specific relations for transferring information within and between contexts. Benerecetti et al. (2000) provide a foundation of a theory of contextual reasoning in artificial intelligence from the perspective of knowledge representation.

There exists little overlapping among the abovementioned research areas regarding their understanding and requirements of context. Each of these fields lacks a precise definition of the term *context*. The same holds for ubiquitous and pervasive computing that rely on context to a great extent. The following paragraph presents characterisations and definitions of the term context-aware applications and context-aware computing that have been published.

2.2.3 Context in Human-Computer Interaction

In the field of human-computer interaction, a large amount of contextual information is shared between a human and a computer during its use, as well. Already a (at first sight) static desktop might accommodate a "focussed window", a "current selection" within a window or a "default background colour". Some of this shared knowledge might be relevant for the interaction and influences the actions executed by a workstation, some might not. The analogy with human-human dialogue is obvious: the fact of working with a computer already creates a situation of communication.

However, to enhance the conversation between humans the utilization of the implicit information about a situation cannot be transferred naturally to a human-computer dialogue (Dey, 2001). Compared to the communication among humans, the communication between a human and a computer is subject to several restrictions that result from the computer's limited perception of its user's current situation, needs, desires, tasks, etc. As a consequence, the user needs to provide such information in an explicit manner, which reduces the user acceptance.

Furthermore, the emergence of mobile, ubiquitous (Weiser, 1991) and pervasive computing (Hansmann et al., 2001) significantly increases the potential situations, in which an interaction between a human and a computer might take place. The vision of a transparent and unobtrusive interaction with computers motivated the research into context-aware computing. In human-computer interaction, context-awareness is a kind of intelligent computing behaviour. The following section provides the relevant definitions of terms related to context-aware computing.

2.3 Definition of Relevant Terms

Today, the term *context* is widely used in different research communities each with a very diverse understanding. Even encyclopaedias need to distinguish specific academic disciplines for their definitions of context. In communications and linguistics Wikipedia (2007) describes context as ”*the meaning of a message (such as a sentence), its relationship to other parts of the message (such as a book), the environment in which the communication occurred, and any perceptions which may be associated with the communication.”* The Free On-line Dictionary of Computing (FOLDOC, 2007) determines context in broader terms leaving room for interpretation: *“context: that which surrounds, and gives meaning to, something else.”* Thesaurus (2007) enumerates a long list of synonyms for context and substantiates the assumption that context is loaded with a lot of different meanings: *“ambience, background, circumstance, condition, connection, dependence, environment, framework, location, meaning, matter, occasion, perspective, relation, setting, situation, status, substance, surroundings, text, vocabulary”* The following paragraphs delineate the evolution of the definition of context-aware computing and context in the research area of computer science.

2.3.1 Defining Context-Aware Computing

The term *context-aware computing* was first introduced and defined by Schilit and Theimer (1994) and indicates software that “*adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time.*” This specification characterises context-aware applications as applications that exploit their changing environment.

Brown (1996) and Pascoe (1998) studied mobile computing systems and provided a more general understanding: context-awareness is the “*ability of the computer to sense and act upon information about its environment.*” During this early period, nearly all context-aware applications were based exclusively on the exploitation of location and proximity information.

Dey (2000) elaborated architectural support for context-aware applications and draws an application-oriented view on such systems. In this sense, a system is a connection of software or hardware components facilitating the flow of information and designed for software to run on. An application is such software that is associated with and running on a system. Both a system and an application can be context-aware:

Definition 4: Context-Aware Application / Context-Aware System

> *An application or 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. (Dey, 2000)*

The common goal within this newly established field of *context-aware computing* is to improve the usability of applications by adapting their behaviour dependent on the context.

The application's usability implies a user- and task-centred view because it is the measure of how specific users in their specific context reach their specific tasks with effectiveness, efficiency and satisfaction (cf. Section 2.6). For future computing applications emerging from fields like ubiquitous or pervasive computing, context-aware computing will be the enabler for putting the major visions into practice.

2.3.2 Defining Context

Since the term context-aware computing was first introduced by Schilit and Theimer in 1994, a large number of further definitions of the terms context and context-awareness has been proposed in the area of computer science. This section first presents a survey of previous definitions of context and then introduces an operational definition that complies with the perspective and requirements of this thesis.

Previous Context Definitions

In his survey, Dey (2001) presents alternative views on context and its definition. Basically, the majority of existing definitions of the term *context* can be categorized into definition by synonyms and definition by example. Context experienced various characterizations using synonyms such as an *application's environment* (Hull et al., 1997) or *situation* (Katz, 1994; Kriste, 2001). Many authors such as Brown et al. (1997), Ryan et al. (1998) or Gross et al. (2001) define context by example and enumerate context elements like *location*, *identity*, *time*, *temperature*, *noise*, as well as the *beliefs*, *desires*, *commitments*, and *intentions* of the human involved (Chen, 2004). For the operational use of context, such indirect definitions by synonym or example suffer from generality in the first and incompleteness in the latter case.

Addressing these quite limited notions and early definitions of context, Dey (2001) provided the following general definition, which is perhaps now the most widely accepted:

"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between the user and the application, including the user and the applications themselves." (Dey, 2001)

This application-centred definition clearly states that context is always bound to an entity and that information describing the situation of an entity is context. However, in using indefinite expressions such as "any information" and "characterize" the definition becomes too broad. Practically, the provided notion of context includes any kind of information that is relevant to the interaction between a user and an application. Thus, any application defined as adaptive in traditional terms, is actually a context-aware application.

Dey (2001) also introduces the user's task as an important concept in context-aware computing through his definition of context-aware applications (cf. Definition 4:). The task itself is also part of the context as it "characterizes" the situation of the user. This central role of the task is shared by Crowley et al. (2002) who assume that user's actions are generally goal-driven. They introduce the term activity to accurately capture the observation that the

user is concerned with several tasks simultaneously. In a more recent work, Chen (2004) documents his understanding of context that "*extends to modelling the activities and tasks that are taking place in a location*". Henricksen (2003a) even puts the task in the centre in her specific definition of context: "*the context of a task is the set of circumstances surrounding it that are potentially of relevance to its completion.*"

In many application domains a well-defined model of activities or tasks may be difficult to specify. Even though a domain-dependent decomposition of more general goals into fine-grained activities and tasks may seem complex, the effort will result in a solid model.

Operational Context Definition

Each of the provided definitions introduces a considerable amount of expert knowledge that needs to be incorporated in further research. Dey's definition is intended to be adequately general to cover the work conducted by research into interaction based on context. In order to further constrain its universality, this general definition needs to be enclosed by a formal and an operational part.

Definition 5: Context

> *Context is any information that can be used to characterize the situation of an entity* (Dey, 2001). *Elements used for the description of context information fall into five categories: individuality, activity, location, time, relations. The activity predominantly determines the relevancy of other context information in specific situations. Location and time primarily drive the establishing of relations to other entities that enable the exchange of context information among entities.*

The formal additive constricts and clusters context information into five fundamental categories, which facilitates the handling of context in concrete applications and the specification of a context model. Section 2.4 further refines these categories of context information. The operational additive of the definition qualifies the reference to a certain task and the exploitation of context information. A further description of inter- and intra-context operations can be found in Section 2.5.

In order to build a bridge to the field of software engineering, the introduction and definition of the terms *context model*, *context information* and *context attribute* is required. Henricksen (2003a) explicitly stresses the lack of a clear separation between the concept of context modelling and context information. Developers need to model context during the engineering process of a context-aware application. Henricksen defines the term *context model* as follows:

Definition 6: Context Model

> *A context model identifies a concrete subset of the context that is realistically attainable from sensors, applications and users and able to be exploited in the execution of the task. The context model that is employed by*

> *a given context-aware application is usually explicitly specified by the application developer, but may evolve over time.* (Henricksen, 2003a)

This definition determines that a context model constitutes an explicit representation of context in a context-aware application, which abstracts the real-world context. In addition, the operation of such applications requires the acquisition of extensive context information. Henricksen interprets context information as an enabling mechanism of applications to perform tasks on behalf of users in an autonomous and flexible manner and provides the following definitions:

Definition 7: Context Information

> *Context information is a set of data, gathered from sensors and users, that conforms to a context model. This provides a snapshot that approximates the state, at a given point in time, of the subset of the context encompassed by the model.* (Henricksen, 2003a)

Evidently, a context model has to cover all relevant information the desired context-aware application needs to operate appropriately. The determination of a context model may include a description of value types, a possible range of these values and relationships among them. Because the context information can be acquired from various sources like sensors, user input or databases and may vary in complexity, the context model needs to represent information at different levels of abstraction. Additionally, a context model is required to be extensible in order to cope with new and unexpected forms of context. As a summary, Winograd (2001) reveals that the conceptual structure of the context model should be:

- broad enough to handle all different kinds of context
- sophisticated enough to make the needed distinctions
- simple enough to provide a practical base for programming

In addition, the term *context attribute* needs to be introduced as a structural element of the context model:

Definition 8: Context Attribute

> *A context attribute is an element of the context model describing the context. A context attribute has an identifier, a type and a value, and optionally a collection of properties describing specific characteristics.*

A context attribute designates the information defining one element of context, like for example “username” or “daytime”. It is either regarded as a label associated with at least one value at a given moment (i.e. attribute-value pair) or as a more complex structure. Other common expressions for context attribute are *context feature*, *element*, *parameter*, *property* or *variable*.

2.3.3 Defining Situations

This subsection provides the definition of meaning of the term *situation*. In the literature, the situation is regarded as a part of context. Brézillon (2002) describes a situation as *"a subpart of the overall context"*, and (Oppermann, 2005) defines situation as *"the relevant context characteristics at a specific pointing time and space."* These two perspectives form the basis of the definition of the term situation in a general form:

Definition 9: Situation

> *A situation is the state of a context at a certain point (or region) in space at a certain point (or interval) in time, identified by a name.*

A situation is considered as a structured representation of a part of the context, which can be named and where location and time are used as spatio-temporal coordinates of this situation. Like a snapshot taken by a camera a situation captures the momentary profile of the context attributes. A situation does not include the long-term development of the context attributes (Oppermann, 2005), i.e. the history, while the context covers the whole history of the interaction, in order to establish continuity. However, the observation that each process can be subdivided in periods with sequences of momentary situations influenced Coutaz and Rey (2002) in their understanding of a context at a certain point in time as a composition of multiple situations over a period of time. Thus, no situation exists without history.

The generality of the definition provided above implies an incomplete description of a situation because a full description of the momentary real-world state is almost impossible. Hence, the situation description is an abstraction of the real world and reduced to characteristic properties. Additionally, several descriptions on different levels of abstraction may reference the same situation, and thus, a hierarchy of nested situations is introduced. The situation "at work" of a person exemplifies this observation: this person's situation might simply be described as "at work" or more detailed as "located at company", "sitting at desk" and "writing a report". This leads to the notion of situations as patterns or abstract classes of context. Each situation pattern marks a certain set of characteristic properties determining this specific situation. In other words, a situation places constraints, i.e. a situation predicate, on relevant attributes of the context description in order to capture momentary conditions of the context. A situation holds in a given context exactly when the situation predicate is satisfied by the description of the context.

Humans characterise and describe situations differently based on their tasks, experiences, expectations, emotions, and knowledge. Thus, the selection and weighting of the characteristic properties of a situation is always subjective (Schmidt and van Learhoven, 2001). For example, one person describes the situation "at-work" by means of the more or less exact location, whereas other people characterise the same situation by means of the workload or by the tools and supplies used.

Furthermore, humans memorise situations and recall them from the past in order to match them with the current situation. This accumulated knowledge contributes to the experience that humans exploit in order to determine their behaviour in the current and future situations. The memorisation of a situation occurs through a simple recording of the situation or a further generalisation of similar situations from the past.

2.4 Categories of Context Information

This section introduces a formal structure of context information to further enhance the notion of context. It presents fundamental categories of context information that determine the design space of context models. A comprehensive modelling of context needs to represent all relevant context states, parts of which are activated only in a specific situation. Because the continuum of all context states is infinite in principle (Brézillon, 2003), any context modelling approach will only capture a section of context. Besides the set of the considered context information, the relevancy of the modelled section in relation to the respective application is essential for the quality of the system adaptation. This is the reason for attaching value to a prior analysis of potentially occurring use cases of context information during the context modelling process. In the end, a comprehensive context modelling approach needs to incorporate the entire spectrum of all possibly relevant context factors.

A context model that meets these requirements becomes large and complex very easily and can only marginally comply with demands on the comprehensibility and manageability. Thus, a structuring of the context into several categories is vital. Although many definitions of the term context are definitions by example, and therefore, suffer from incompleteness offering a structure of context appears to be a pragmatic approach and facilitates the engineering of a context model for context-aware applications. The aim of this section is the deduction of such a structure, for which a justification is provided in Section 2.5.

2.4.1 Previous Categorization Approaches

Schilit et al. (1994) enumerate constituents of the context as "*the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time.*" On a conceptual level it is also argued that further issues, such as lighting, noise level, communication cost, and social situation are of interest and can be regarded as context. Dey et al. (2001) extend their definition of context with the statement "*context is typically the location, identity and state of people, groups, and computational and physical objects.*" A high amount of enumerations discriminates context into *personal* and *environmental* context (Gross and Specht, 2001; Mitchell, 2002). Most of the issues that are classified as personal context are often also referred to as user profiles and usually stay the same during the operation of the application. Environmental contexts are of a more general nature and include attributes like "*the time of day, the opening times of attractions and the current weather forecast*" (Mitchell, 2002).

In the field of modelling and reasoning within real-world knowledge, Lenat (1998) suggests to concretely define context as a point in a twelve dimensional space in which context information is characterised. These contextual dimensions organise the background knowledge for reasoning processes. Four of these dimensions refer to spatio-temporal issues and most of the remaining eight dimensions allocate human intent. Schmidt (2002) provides some structure for the characterisation of context, as well, and qualifies context as a three dimensional space with the dimensions *self*, *activity* and *environment*. The self dimension introduces a relation of the context to one specific entity (i.e. a user, device, application, etc.). However, his description lacks an approach of how his model captures a setting comprised of many interacting entities, each bound to a context by the self dimension. The dimensions time and location are consciously missing due to the fact that time is implicitly captured in the history and due to the observation that context is not necessarily related to location.

2.4.2 Fundamental Categories of Context Information

Structuring context information in the form of taxonomies, ontologies or categories is a very useful and a flexible way of organising contextual knowledge. The collections of context information categories presented above clearly show that what is considered as context depends on what needs to be contextualised (Klemke, 2002). Therefore, many open models offer a general structure that can be extended to suit the application domains for context-aware applications. Depending on the field of application, the emphasis on the different types of context varies: For example, mobile applications focus on the location information, while recommender applications base upon information characterising the user. However, present categorisations of context information fail to establish any fundamental basis, on which they should be constructed, and omit convincing arguments for why this particular breakdown is significant. Even though given examples demonstrate their utility, their construction basically is driven by the ease of implementation.

Modelling context around entities seems to be a natural way, and therefore, context should be provided in a natural structure: one entity possesses exactly one context, which is bound to this entity (e.g. through an identifier). An entity can potentially be any possible object that can have a context and is relevant for a context-aware application. In addition, this thesis leaves the perspective, that each user context includes properties of surrounding entities, and follows the observation, that the relations between entities are essential for modelling context (Schmidt and van Learhoven, 2001). This approach holds potential for further compositions and extensions, similar to the object-oriented approach in software engineering, and increases the versatility and reusability of components.

Any information describing this entity's context falls into one of the five categories of context information as shown in Figure 1: *individuality*, *activity*, *location*, *time*, *relations*. The category "individuality" contains properties and attributes describing the entity itself. The category "activity" covers all tasks this entity may be involved in. The categories "location" and "time" provide the spatio-temporal coordinates of the respective entity. Finally, the

category "relations" represents information about any possible relation the respective entity may establish with another entity. These five categories of context information that comprise the context model are described in more detail in the following paragraphs.

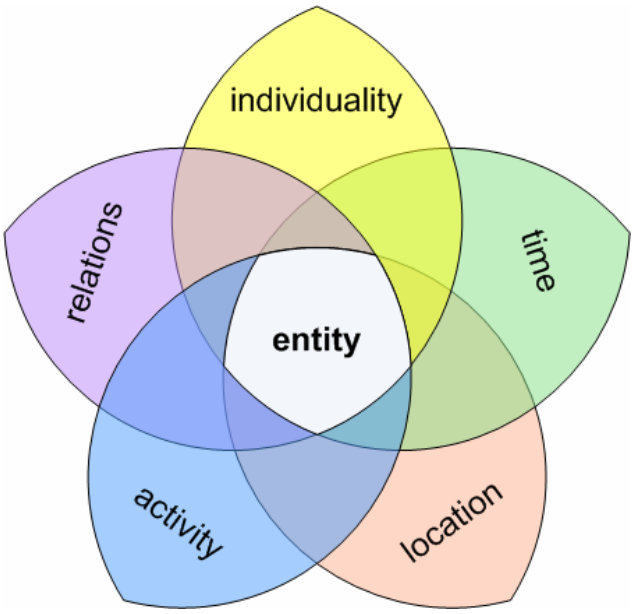


Figure 1 Five Fundamental Categories of Context Information

Individuality Context

This category gives access to contextual information about the entity the context is bound to. This information comprises anything that can be observed about an entity, typically its state. An entity can either be individual entity or groups of entities that share common aspects of the context. Entities can act differently within a context-aware application or obtain different roles. Basically, they can be active, i.e. they are able to manipulate other entities, or passive. In addition, entities can be real, i.e. existing in the real world, or virtual, i.e. existing in information space. Furthermore, there exist mobile, movable and fixed entities.

Because in principle an entity can be any imaginable being or thing, the descriptions of such an entity vary significantly. Thus, a clustering into descriptions of sentient, natural, synthetic, and group entities is advisable. A place or a region are not regarded as an entity and must not be seen isolated because the location information is always related and bound to an entity (context information emerges at places and regions). The following sections illustrate the four types of entity description taking entities as parts of a human-computer interaction as examples.

Sentient Entity Context

Potentially, this category of context information may cover properties of any possible creature. In human-computer interaction the adaptation of the system behaviour to its current user is the main focus of interest. The foundation of this intension is the representation of the personal characteristics of this specific user. In order to automatically perform adaptations that meet the user's necessities adaptive systems need to base their decisions on the evaluation of the user behaviour and consider basic user dimensions such as described in (Heckmann, 2005):

Figure 2 Groups of Basic User Dimensions (Heckmann, 2005)

In addition to these user dimensions, domain-dependent user model dimensions require additional general world knowledge. The consideration of user preferences like language, colour schemes, modality of interaction, menu options or security properties, and numberless other personal favourite preferences is one of the most popular sources of adaptation. Furthermore, an important prerequisite for the content adaptation is the awareness of the user's interests and disinterests because they contribute to the configuration of filters for the information presented to the user.

Synthetic Entity Context

The synthetic entity context denotes products or phenomena that result from human actions or technical processes. In a broad sense, this category covers descriptions of any human-built thing like buildings, computers, vehicles, books, and many more. Software related artefacts resulting from a software engineering process such as a requirements report, the design, a test plan, the product documentation, a software application, services or events are included, as well. Furthermore, this part includes computing hardware descriptions supporting devices such as laptops, Personal Digital Assistants (PDAs) or Smartphones. Sensors as technical components that measure physical or chemical properties are synthetic entities as well. Sensors quantify temperature, lightness, humidity, pressure, sound, magnetism, acceleration,

force, and many more other properties. Furthermore, these components can qualitatively or quantitatively analyse the material composition of the environment.

Natural Entity Context

Unlike synthetic or artificial entities, natural entities are things or procedures that do not result from planed processes. This category comprises all living and non-living things that occur naturally on earth. Thus, it captures things of an environment that are not the result of human activity or intervention. The natural environment may be contrasted to "the built environment" and includes for example plants, stones, countries, etc. because these entities are relating to nature and without any artificial additives. Furthermore, any product of the interaction between nature and humans is part of this category as well.

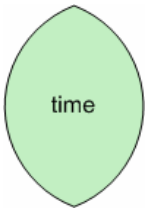
Group Entity Context

A group is a collection of entities, which share certain characteristics or interact with one another. The grouping of entities constitutes a means of denominating associated entities, who have established certain relations between each other (cf. “relations context” on page 27). As members of the group, these entities share a common identity and each entity plays a specific role in this group. Groups may be large (e.g. “the Germans”) or small (e.g. “the Smith family”), and in principle, entity may belong to none, one, or many groups. In addition, groups may develop dynamically or stay stable for a long time. A grouping of entities can take place during the system operation, e.g. based on observations, or in advance, e.g. to express a fixed relation (cf. Section 2.5.3). The main difference between a simple aggregate (e.g. of things) and a grouping (e.g. of people) is that the former comprises merely a number of individuals and the latter exhibits cohesiveness to a larger degree.

The primary purpose of using groups is to structure sets of entities and to capture characteristics that only emerge if individual entities are aggregated together to one unit. In addition, through subdividing entities in subentities the problem of modelling context can be subdivided as well (Schmidt, 2002). Characteristics that members of the group may share include interests, skills, cultural background, or kinship ties in a social sense, and computing power, network connections, or display size in a technological sense.

Example: The example application scenario (cf. Section 2.1.1) provides instances of individuality contexts. As a sentient entity type, Carmen’s individuality context comprises her name, age, interest in music, room temperature preferences and so on. Carmen’s smart phone is a synthetic entity type and possesses characterising properties like screen or display size, the bandwidth or reliability of the accessible network connection. Furthermore, Carmen’s potential customers constitute a group entity that obtains the group-specific property “number of members”.

Time Context



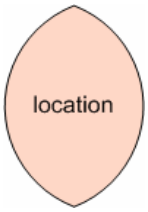
Time is a vital aspect for the human understanding and classification of context. A clear model of time and time intervals is essential because most statements are related over the temporal dimension. Thus, as an important dimension for describing a context (Gross and Specht, 2001), this category subsumes time information like the current time, the time zone or any virtual time. A straightforward representation of time is the Central European Time (CET) format, which facilitates mathematical calculations and comparisons. Overlay models for the time dimension are often applied in context-aware computing and provide categorical scales like working hours or weekends. Other domains require a more procedural view of the time concept (e.g. work flows). The ability to represent intervals of time also constitutes a fundamental requirement on the context model. In combination with the ability to capture and express recurring events (e.g. always on Sundays), time intervals constitute a significant feature for modelling user characteristics.

The past context is always part of the current context, and therefore, the evaluation of the interaction of users with the system includes a history of the usage process in order to establish a continuous user profile. Depending on the adaptation goal, short- or long-term perspectives of the entire context over a certain time period adequately reflect user characteristics. Thus, an additional requirement related to the consideration of temporal characteristics is the desire to persistently store the entire context or situations, which creates a data pool containing a history of obtained contextual information. This history forms the basis of accessing past context information, analysing the interaction history, inferring usage habits of users and predicting future contexts. Moreover, context management issues benefit from the access to historical context information because incomplete or imprecise context values can be extrapolated.

Basically any context information is afflicted with temporal characteristics because it exhibits a timestamp of its acquisition: Both user-supplied and sensed context information may depict significant differences in the rate this information is acquired and in the rate this information de facto changes. A close match of these two rates should be aspired. An additional time delay results from the latency hidden in the processing chain the acquired context information passes through (e.g. because of the physical distribution of hardware components). This temporal diffuse set of information needs to be cumulated into a coherent description of the context, which requires a well-defined model of the concept time.

Example: The time context of Carmen in the example application scenario comprises information such as “May, 14th”, “2007”, “afternoon”, and “17:38”. Furthermore, her history contains information about her lunch time at 11:15 and that she went back to her office at 12:00.

Location Context



With the development of portable computing devices the location has become an important parameter in context-aware applications. Physical objects and devices are spatially arranged and humans move in mobile and ubiquitous computing environments. Since tasks often include mobility, this category describes location models that classify the physical or virtual residence of an entity, as well as other related spatial information like speed and orientation (Zimmermann et al., 2003b). Furthermore, a location may be described as either an absolute location, meaning the exact location of something, or as a relative location, meaning the location of something relative to something else.

Models for physical locations can be discriminated into quantitative or geometric location models, and qualitative or symbolic location models (Stahl and Heckmann, 2004). Quantitative location models refer to coordinates with two, two and a half or three dimensions. The geographic coordinate system exemplifies a coordinate system, which expresses every location on earth in the format degrees, minutes and seconds for the longitude and latitude. Tracking or positioning systems such as the satellite-based Global Positioning System (GPS) supply location information through measuring distances or angles to known reference points and translating these relative positions into absolute coordinates. Furthermore, such systems can be classified according their indoor or outdoor operating mode, their granularity of position determination and their underlying technology, e.g. radio or light signals (Lorenz et al., 2005).

Instances of qualitative spatial information are buildings, rooms, streets, countries, etc. that depict a mutually nested relationship. Such qualitative information increases the transparency for humans regarding their spatial cognition because they introduce several spatial granularity levels. Overlay models allow for an interpretation of quantitative spatial information and transformation into appropriate qualitative information. Stahl and Heckmann (2004) conducted an investigation on spatial concepts and models, and propose a hybrid location modelling approach.

In electronic or virtual space like the World Wide Web, the unified resource locator of a webpage determines the current location of an entity. In addition, a virtual location could also be a technical location like the position within the network identified by an Internet Protocol (IP) address. Thus, an entity always possesses one physical qualitative location, which can be represented by different quantitative locations, but also several virtual locations at the same time.

Example: Carmen's current location is at the Steigenberger Hotel in Berlin. The precise address of this hotel is Los-Angeles-Platz 1 in Charlottenburg, 10789 Berlin (Germany). This address corresponds to the latitude 52.50 north and longitude 13.34 east. Furthermore, Carmen is sitting on a chair near a table. The video wall that Carmen is using to edit her presentation is installed in room number 133.

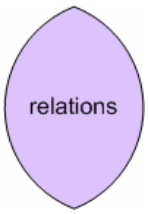
Activity Context

For context-aware applications the activity category can be considered as a very important feature of an entity because an activity determines the current needs of this entity to a great extend. The activity context covers the activities the entity is currently and in future involved in and can be described by means of explicit goals, tasks, and actions. In most cases when interacting with a context-aware application, an entity is engaged in a (potentially demanding) task that determines the goals of the performed activities (Brusilovsky, 1996). Computer systems may even work on several tasks at the same time. A task is a goal-oriented activity expectation and represents a small, executable unit (Klemke, 2002). Some task might be well-practiced, other less-practiced, depending on the level of experience the entity has gained during several performances of the task. Altogether, the activity context answers the question, what does the entity want to achieve and how?

Tasks include operation sequences with a determined goal, to which a context-aware application can adapt the necessary functions and sequences of functions. Humans sometimes change their goals very frequently depending on quickly appearing conditions or decisions, even without leaving the session with a computing system. Therefore, a differentiation between low-level goals, which can change quite often, and high-level goals, which are more consistent, is reasonable. Accordingly, the activity context can be represented by (domain-specific) task models that structure tasks into subtask hierarchies. A hierarchy or tree of tasks is the most advanced representation of possible user goals (Vassileva, 1996). Each context-aware application supports a set of possible goals or tasks, which it can recognise. In some cases, the set of goals is very small and the goals are not related to each other. The determination of the current goal is either specified by the entity or a choice from the set of goals, which depicts the highest probability.

Example: Carmen was involved in a chain of activities until her boss entered the room: she opened the door to her room, adjusted the lighting conditions of this room, sat on a chair, etc. The activity of the video wall entity mainly consisted in displaying the music selection and later on Carmen's presentation.

Relations Context



This category of context information captures the relations the respective entity has established to other entities. Such surrounding entities can for instance be persons, things, devices, services, or information (e.g. text, images, movies, sounds). The set of all relations of the respective entity builds a structure that is part of this entity's relations context. A relation expresses a semantic dependency between two entities that emerges from certain circumstances these two entities are involved in. Additionally, each relation exhibits certain strength of how the two involved entities cohere.

The term *relation* is not identical to the terms *interrelation* or *correlation* because an interrelation between two entities is a special kind of relation. An interrelation between two entities means that changes of one entity immediately leads to respective changes of the related entity. Thus, a relation is not necessarily an interrelation. The characteristics of the entity's environment (i.e. presence and the arrangement of entities like artefacts, natural objects or people) are primarily determined by and strongly depending on the spatial and temporal context of this entity. Secondarily, the individuality of the respective entity description impacts the relations (e.g. people of the same age).

As a general rule, relations are always named by a verb form like "X uses Y" or "X works with Y" and in each relation the two concerned entities play one relation-specific role. Potentially, one entity can establish several different relations to the same entity. Additionally, relations are not necessarily fixed or static and may emerge and disappear dynamically. The analysis of known relations, e.g. through exploitation of the transitivity of relations, leads to the inference and explication of implicit relations between entities. This results in new "entity neighbourhoods" that can be exploited. The exploitation of a relation leads to the creation of an internal model of the concerned entity the relation points to (cf. Section 2.5.3). This model consists of facts that result from gaining insight into public and accessible parts of this entity's context, and assumptions that arise from observation or derivation.

Since the set of possible relation types between two entities is large, a clustering of relations regarding the types of the entities involved is helpful. Therefore, the relations category is subdivided into social, functional and compositional relations.

Social Relations

This subcategory describes the social aspects of the current entity context. Usually, such interpersonal relations are social associations, connections, or affiliations between two or more people. For instance, social relations can contain information about friends, neutrals, enemies, neighbours, co-workers, and relatives. One important aspect in a social relations context is the role that the person plays in this context. Social relations differ in their levels of intimacy and sharing, which implies the discovery or establishment of common ground. Information about conjoint characteristics a person shares with other people, or about individual differences, also contribute to the characteristics of a specific person. From this, patterns in behaviour may be derived or groups of people with identical interests, goals, or levels of knowledge.

Functional Relations

A functional relation between two entities exists, if one entity makes use of the other entity for a certain purpose and with a certain effect, e.g. transferring a specific input into a specific output. Functional relations may exhibit physical properties like using a hammer or show communicational and interactional properties like typing in a word or speaking into a

microphone. Moreover, this relations subcategory indicates mental and cognitive properties like reading an article, giving a presentation or reasoning a concept.

Compositional Relations

A very important relation between entities is the relation between a whole and its parts. Parts of a composition are existentially dependent on the whole because they will no longer exist if the containing object is destroyed. The aggregation is a weaker form of the ordinary composition because it does not imply ownership and parts can have more then one whole they belong to (e.g. a fax machine may belong to different secretariats or different departments).

Example: The example application scenario of Section 2.1.1 provides a large set of social relations that constitute Carmen's relations context: she is married to her husband, an employee of her boss, talking to the customers, etc. The example application scenario further describes some functional relations: Carmen is operating her smart phone, sitting on a chair, and giving a presentation. Furthermore, Carmen's relations context exhibits strong compositional relations because she owns arms, fingers, legs, etc. In addition, Carmen's car aggregates four wheels, one motor, two front lights, one car radio and so on.

2.5 The Use of Context

Context obtains a specific role in communication because it is an operational term: something is context because of the way it is used in interpretation, not due to its inherent properties (Winograd, 2001). When interacting and communicating in everyday life, the perception and the interpretation of context constitute a major part for humans. They have an informal sense of how to make use of context, some of which can be transferred to context-aware applications. The following dynamic and operational properties of context foster a systematic foundation of the use of context in context-aware applications. The first section enumerates several characteristics of context information that can be found in the literature. Section 2.5.2 identifies basic properties of transitions between contexts. Finally, the meaning of sharing contexts among entities is investigated.

2.5.1 Characteristics of Context Information

Context information can be structured into five categories regarding the type of content it represents. Furthermore, context information exhibits a range of characteristics that are orthogonal to these categories. A context-aware application needs to take these characteristics into account in order to ensure an effective and efficient management of the context information they exploit. This thesis inherits the investigations of researchers such as Kari and Candolin (2003), Henricksen (2003a) or Strang and Linnhoff-Popien (2004), who have classified the following characteristics of context information:

Interrelation

A considerable difference exists between the contextual information that is gathered from the environment and the processed information that is used to assist an interaction. The raw contextual information can take on many forms when combined with other context information. Interrelations may exist within one category of context information (e.g. in the *location* category the context attribute "gps_position" determines the context attribute "city") or between them (e.g. the context attribute "speed" is determined by information provided by the categories *location* and *time*). Figure 1 on page 22 illustrates interrelation through the differently shaded intersections between context information categories. The characteristics of the derived information are intimately linked to the properties of the information it is derived from (Kari and Candolin, 2003). The more context information is combined the more semantically enriched or higher-level context information emerges. Common transformation techniques for contextual information are fusion, derivation, aggregation and interpretation, which differ in the knowledge that is required for their application.

Heterogeneity

The sources from which context information can be acquired vary significantly and need to be merged in order to perceive the entire context of an interaction. Context information can be supplied by users (e.g. asking her for the time of the day), by devices (e.g. by a thermometer), or emerges from observations (e.g. many museum visitors walk "clock-wise") or from derivation (e.g. speed of a car results from the time it requires to travel a certain distance). Henricksen (2003a) discriminates four types of sources of context information: *sensed*, *static*, *profiled* and *derived*. The integration of context information from diverse sources may lead to changes in characteristics like persistence and quality.

Persistence

Context information exhibits a range of temporal characteristics. In terms of context information persistence means the time in which this information stays the same. Context information roughly can be subdivided into static (i.e. predominantly invariant) and dynamic (i.e. predominantly fast changing) information. The user's date of birth is an example for the former and for the latter the GPS coordinates of a car. Additionally, past context information may be needed to understand the full state of the environment. Dynamic information can be accumulated continuously, frequently and automatically. Thus, histories of context information become relevant. Furthermore, the freshness of (rather static) information can be relevant.

Quality

Any source of context information is afflicted with some quality of the provided information because some sources are more reliable than others. Devices tend to malfunction under adverse conditions and even people sometimes deviate from the truth if they are asked for a

self-assessment. Additional reasons for the doubt over the soundness of the context information arise due to the speed at which this information alternates because the "*delay between the production and use of context information*" (Candolin and Kari, 2003) is a concern. The loss of confidence in the correctness of a context description arises from unknown, ambiguous, imprecise, and erroneous information provided by various sources and compared to the true state.

Granularity

Contextual information can be available on different levels of granularity or levels of abstraction. For example, one sensor for measuring the volume of a sound may only distinguish the two states "on" respectively "off", whereas another sensor for the same kind of information may provide measured values in the range between 0 and 100. The same holds for the varying human perception of this contextual information. This matter of accuracy and resolution can be referred to as horizontal granularity because it considers only one kind of information. Vertical granularity indicates abstraction levels involving more than one kind of information or even the entire description of the context.

Representation

Various alternative representations exist for the same contextual information without changing the level of abstraction. A human could say "this car is red" and a computer system could represent this information as a variable assignment in the form "car_color = red". Furthermore, contextual information can be represented qualitatively (e.g. "the room temperature is moderate") or quantitatively (e.g. "the room temperature equals 22 degrees Celsius"). The use of different types such as Booleans, numbers or symbols for describing information introduces additional multiplier for alternative representations because each type indicates a set of values that have the same sort of generic meaning or intended purpose.

Distribution

Contextual information lacks a central instance from which it can be requested, and thus, is distributed by nature. Since entities are always located at a certain position in space, the perception of what something looks like or what someone feels or hears depends on the position and the spatial distances to the source of context information (Schmidt, 2002). The spatial distribution of context information determines its accessibility. While moving through real or virtual space (i.e. information space), context information becomes accessible or inaccessible.

2.5.2 Context Transitions

Entities and particularly humans move between contexts and two consecutive contexts are never exactly alike in this process. The knowledge necessary for context changing is basically

contained in the context itself, and thus, closely enlaced with the categories of context information and their characteristics. The following paragraphs describe the coherences of how context attributes change from exiting one context and entering another.

Variation of Approximation

While migrating from one context to another, the contextual knowledge represented by the current context experiences a specialization or an abstraction. The level of specialisation or abstraction of the context is closely connected to the different levels of granularity exhibited by context information (cf. Section 2.5.1). A representation of the real context of an entity is always an approximation because it abstracts away some aspects. Figure 3 shows the variation of approximation within the boundaries of the context model.

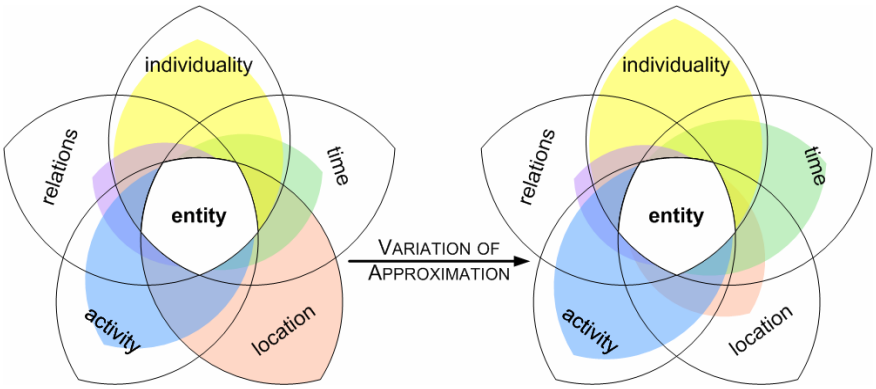


Figure 3 Variation of Approximation

The notion of approximation is relative: one representation is more approximate than the other because it abstracts away details that the other takes into account. Through varying the degree of approximation a partial ordering over contexts emerges: if two contexts are compared, one contains all the information of the other and probably more. An additional mechanism of varying the approximation degree is the memorisation of past situations. As snapshots of the current context, situations continuously contribute to the context history, and thus, further account for a context specialisation. This accumulated knowledge leads to making experiences explicit in the context representation and to transferring knowledge from one category to another.

Example: In the moment Carmen enters the hotel, her GPS position cannot be obtained anymore and this precise location information is replaced by the more abstract location information "Steigenberger Hotel". The context description of Carmen's hotel room may experience a variation of approximation as well, if the context description would rather involve walls, doors, and windows than the chemical components of the furniture.

Change of Focus

The focus of a context refers to the reachability or accessibility of specific elements of the context description in a specific situation. The value of a context attribute is inherently bound to a location. Therefore, this context information represents the context for this specific position or region and is fully relevant at this location. The point in time, at which the value of the context attribute originates, exhibits a similar semantic as for location. For an entity, the spatial and temporal distance to the source of this context attribute determines whether this context attribute is in focus or out of focus.

Context information has a time and a point or region of origin, at which the focussing or relevancy of this context information is maximal (Schmidt, 2002). In general, values that emerge at a certain point in time are more relevant than those created earlier or later. As Figure 4 shows, this relevancy as well as the certainty on the correctness of the provided value decreases with an increasing temporal or spatial distance from the origin of the context information. This fact can contribute to the disambiguation of multiple values for the same type of context information.

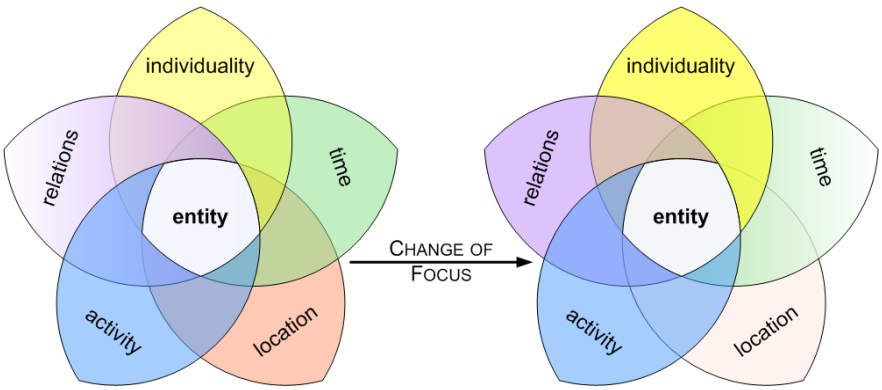


Figure 4 Change of Focus

Example: Because Carmen's GPS device runs out of power and looses the connection to the satellites, the context attribute "gps_position" as a part of her location context gets out of focus. On the other hand, in the moment she starts determining her location using a map, the context attribute "street" gets into the focus.

Shift of Attention

The activity and the task an entity is currently involved in identify the need for knowledge required for their processing, including contextual knowledge. More precisely, the activity determines the focus of attention on specific aspects of the contextual knowledge (i.e. context information). As Winograd (2001) states, features of the world become context through their use. The focus of attention is switched when the activity of an entity changes, signalling that a new task is to be performed. The tasks and subtasks are the loci where a shift in the focus of

attention may be evoked. A switch in the attention focus and thus, a changing need for contextual knowledge, leads to different perspectives on the context information. A coherent set of differently weighted aspects of this context information constitutes such a perspective. Figure 5 illustrates a shift of attention towards a more location-oriented perspective. Each aspect of the context plays a specific role during the performance of a task and this role might show considerable variance across the course of an activity.

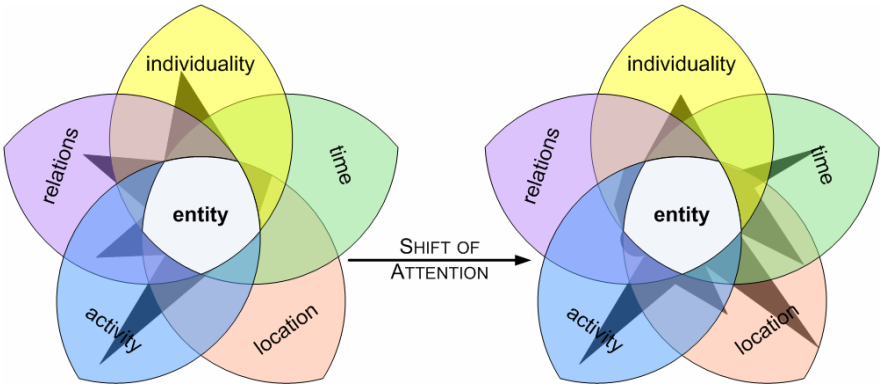


Figure 5 Shift of Attention

Example: After the meeting Carmen gets into her car to drive home, but her GPS device runs out of power. Because she feels unfamiliar with this particular area of the city, she needs to use a map to get on the highway and determines her location by trying to find the names of nearby streets on her map. Thus, her location, which was irrelevant during the meeting, comes back into the focus of her attention.

2.5.3 Shared Contexts

A shared context emerges when the contexts of two entities overlap and parts of the respective context information become similar and shared. Besides the occupancy of its own context, an entity can belong to one or more different contexts or context parts owned by other entities. Thus, through sharing contexts an entity can be viewed under different perspectives. Additionally, a group of entities sharing certain context parts share knowledge of how things are done and understood in this group. In the following, the emergence and exploitation of shared context is described: First, the correlation concerning time and space enables the detection of a relation between two or more contexts. Second, the regression of time and space enables estimating which type of relation is detected. Third, the newly established relation is consolidated.

Establishing Spatio-Temporal Relations

In everyday life it can be observed that without proximity physical interaction (e.g. with a computer) would not be possible. Humans always reside at a certain position in space, which

is usually the centre of attention, action and perception. The time bears analogy to the location because a human can only act at a certain point in time. Thus, the human's perception of context is restricted to the point in time and strongly dependent on the position where someone is situated. This observation can be transferred to humans or, more general, entities that converge in one or the other way: spatial and temporal proximity enable them to start responding to each other.

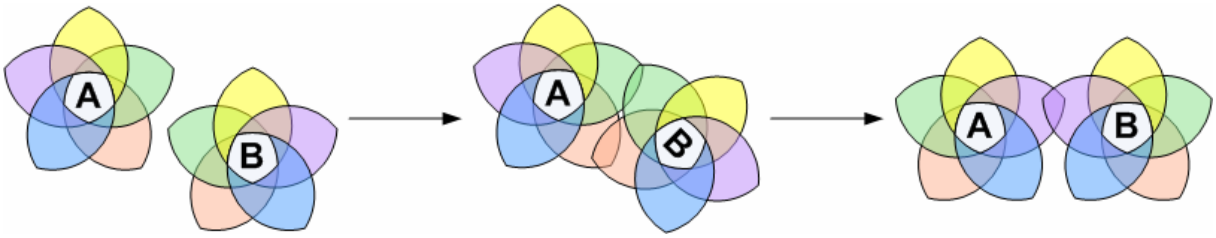


Figure 6 Establishing a Relation

Before two entities can build a shared contextual knowledge, a shared time and space is required because distance is a barrier to shared contexts in this case. Time and space are the cardinal bridging mechanisms of detecting similarities between two contexts and of establishing relations. Figure 6 depicts the process of establishing a relation between two entities A and B: The two entities approach each other, time and location overlap, and a new relation between A and B is established. Additionally, temporal and spatial proximity leads to reciprocity of two entities' contexts, and thus, forms the basis of the creation of entity groups or communities. It is worth mentioning that similar locations in particular appear in various forms: visitor in front of a painting, people on the same bus, or two persons accessing the same webpage.

Example: In the example application scenario provided by Section 2.1.1 the hotel room constitutes the enabler for establishing relations of various types. Carmen's presence in this hotel room allows her to establish functional relations to entities such as the video wall, the chair, or the table. The arrival of Carmen's boss Anne in this hotel room temporally leads to a special social relation between the two. Furthermore, their presence in front of the video wall, which displays Carmen's presentation, forms the basis of an additional relation that could be named "Carmen works on the presentation with Anne".

Adjusting Shared Contexts

Humans possess a different perception of contextual information, which mostly is due to the availability of context information on different levels of granularity or abstraction (cf. Section 2.5.1). The parties participating in an interaction need to share the same understanding or interpretation of the meaning "behind" a description of the context. Figure 7 exemplifies such an adjustment of the abstraction level regarding one topic: entity A and B need to adjust their respective knowledge regarding this specific topic because entity A has deeper and different knowledge compared to entity B. Once a relation between two humans is established, they use

specific mechanisms or rules to obtain a common understanding of their shared context. Humans are able to assess specific aspects through observation or clarify other aspects through asking their counterpart. Such an adjustment expands a shared context and provides a common ground and a shared framework for the communication between two parties. The same object can have different names in different contexts and by taking into account the "translation" of a representation into another a change of the perspective is possible.

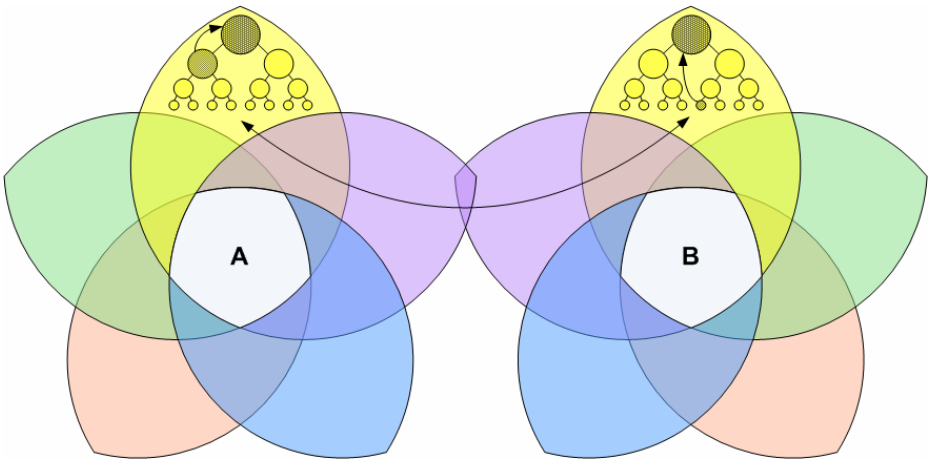


Figure 7 Adjusting Shared Contexts

Example: Carmen prepared the presentation she is about to give and thus, possesses more detailed and precise knowledge concerning each slide. Her boss Anne sees this presentation for the first time and requires additional knowledge, in order to fully comprehend the presentation. Her conversation and discussion with Carmen provides this required knowledge and leads to an adjustment of the shared context. Furthermore, the disambiguated reference to a company-internal document Carmen and Anne have read results in an additional shared experience, and thus, immediately leads to a better understanding of each other through uncovering a lot of background knowledge.

Exploiting Relations

The larger the shared context between two interacting parties, the more it facilitates their communication because they better understand what is expected without being explained in detail. After a relation is established and the shared context is adjusted, an entity gains special insight into the context of the other entity. Such an intense relation enables context fusion or synthesis through three different mechanisms (cf. Figure 8): Building of an internal model of the other entity, extending the own model and transcending relations.

Persistent relations among entities lead to a building of internal models of their respective counterparts based on inquiry or observation (cf. Figure 8a). The internal model of the entity the relation is pointing to consists of two parts: facts, which are references to public and accessible parts of the entity's context, and assumptions, which are uncertain derivations and

inferences about private and inaccessible parts. Because this internal model relies on interpretation and derivation, a mismatch may exist between this model and the real entity. An example is the system's model of the user's context, which will never completely meet the reality because the limited capabilities of the computer system regarding inference mechanisms and representation only allow for an approximation of the user's context. The "intellect" of a computer system comprises the rules and algorithms that it works with and that a developer implemented.

A second way of exploiting an established relation targets the extension of the context description of one entity with parts of the context description of the other entity (cf. Figure 8b). Some relations allow for a more or less certain assumption of a context attribute value of one entity based on the value of the same context attribute of the other entity. Context attributes that constitute potential candidates for a value exchange lie in the intersection of the two shared contexts.

Moreover, the exploitation of relations between entities includes the recognition and discovery of transitive relations that allow for reaching further unknown entities. From within a shared context, entities can "reach" any entity that belongs to the relations of an entity within that context and build an internal model of this entity (cf. Figure 8c). Potentially, this procedure can be repeated recursively with any entity that lies on the path.

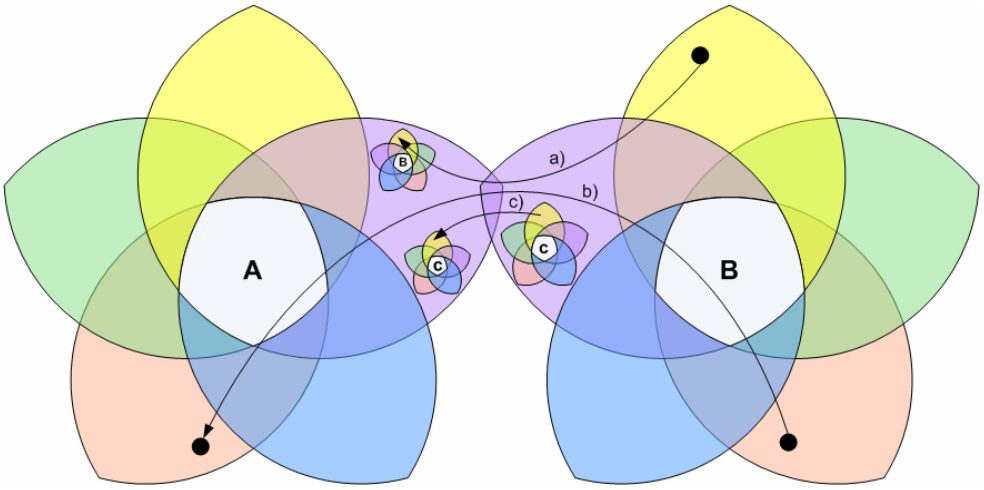


Figure 8 Three Ways of Exploiting a Relation

Example: Carmen works for Anne for two years now, and the exploitation of this relation enabled Carmen to speculate about Anne's character, lifestyle and habits (cf. Figure 8a). Furthermore, Carmen has established a relation "carries" with her smart phone that obtains its position through GPS. Through exploiting this relation, the position of the smart phone can implicitly be transferred to the position of Carmen because it is likely that both entities share the same location (cf. Figure 8b). In addition, Carmen established a relation "trusts" with Anne. Anne in turn has established the same relation with her financial consultant Andreas, and therefore, it is likely that Carmen can "trust" Andreas to a certain degree (cf. Figure 8c).

2.6 Usability of Context-Aware Applications

Although in many cases the adaptations automatically performed by a context-aware application are very desirable, they always represent the perspective of the developer. For the user of such an application this bears the risk of getting into situations, in which the context-aware behaviour implemented by this developer is inappropriate or undesired, and the performed modifications potentially cause user discomfort. The developer draws a specific model of tasks the system is going to accomplish, and if the task of the user does not fit with this model, the user needs to change her way of performing her task.

Context-aware applications are associated with a number of typical usability problems. In some cases a decrease of the system's usability outweighs the benefits of adaptation. The overall goal is to ensure an adequate fulfilment of the usability goals without eliminating the benefits of the adaptation processes. The anticipation and prevention of usability side effects should form an essential part of the design of context-aware applications. This section addresses critical factors embodied by usability problems associated with context-aware applications and presents a way of dealing with them.

2.6.1 Guiding on Usability

According to the norm "Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability" (ISO9241-11, 1998) by the *International Organization for Standardization*, the term *usability* characterises the measure of how specific users in their specific context reach their specific tasks with *effectiveness*, *efficiency* and *satisfaction*:

Effectiveness: Accuracy and completeness with which users achieve their goals.

Efficiency: Resources or effort expended in relation to the accuracy and completeness with which users achieve their goals.

Satisfaction: Exoneration from discomfort and positive attitudes towards the use of the software.

The performance of a task is enabled by the system's dialogue features. The effectiveness and efficiency of this performance can be improved if the requirements of task performance have been satisfied. The usability goals should be assessed taking into consideration user characteristics such as: attention span, limits of short-term memory, learning behaviour, level of work and system experience. In addition, the user's internalised view of the underlying structure and purpose of the system with which the user will interact needs to be taken into account.

2.6.2 Usability Goals

In the following, seven usability goals are first described in detail and then reflected with regard to the specific case of context-aware software. These usability goals are not independent from each other, and it may be necessary to trade off the benefits of one goal against others. Because of these trade-offs, no single solution is appropriate for all of the users all of the time. The applicability and the relative importance will vary with the specific field of application, user groups and the chosen dialogue technique. It may be necessary to establish priorities on a case-by-case basis when applying the principles.

Task Suitability

In order to support the user in reaching the desired goal or task directly, the dialogue should be designed in a way that it considers the complexity of the tasks with respect to the user's skills and abilities. Users benefit from function that is easily accessible and usable, and the format of input and output should be appropriate to the given task and user requirements. A poorly organised interface cluttered with many advanced functions distracts users from accomplishing their everyday tasks.

"*A dialogue is suitable for a task when it supports the user in the effective and efficient completion of this task.*" (ISO9241-11, 1998)

Context-awareness can support users in managing complexity by the careful organisation of information so that it is shown only at the appropriate time. It could help avoiding unnecessary task steps and present the user only with information related to the completion of the task. Thus, basic functions could be immediately apparent, while advanced functions may be less obvious, e.g. to new users. Functions could be included only if a task analysis shows they are needed. Because users want to reach their goals directly, a context-aware dialogue can support the user when performing recurring tasks: any actions that can appropriately be allocated to the interface software for automatic execution can be carried out by the software without any user involvement.

On the other hand, if context-awareness is applied incorrectly and relevant information or functionality is hidden from the user, the task suitability of the application suffers or even disappears completely. The same holds for wrongly drawn inferences about the user's context, in particular about skills and abilities, and a subsequent determination of an inappropriate in- and output modality. The need for additional explicit input required by the user may increase significantly, and thus, users have to concern themselves with how to use the system.

Self-Descriptiveness

A user's actions need to be predictable and should cause the results the user expects. Users should recognize what the computer is doing by means of visual representations of how their actions affect the objects on the screen. Terms and images should match users' task

experience, and help users understand the objects and their roles and relations in accomplishing tasks. If severe consequences may result from the user action, the system should provide explanation and request confirmation before carrying out the action. Feedback or explanations should strictly relate to the situation for which they are needed in order to enhance their value to the use.

"*A dialogue is self-descriptive when each dialogue step is immediately comprehensible through feedback from the system or explained to the user on request.*" (ISO9241-11, 1998)

If the application is context-aware, feedback or explanations can be based on the level of knowledge of the typical user and vary in type and length, based on user goals, needs and characteristics. By allowing users to transfer their knowledge and experience, metaphors provide a cognitive bridge and support user recognition rather than recollection. If the quality of the tailored feedback or explanations is increased, context-awareness could minimize the need to consult user manuals and other external information, thus avoiding frequent media switches.

However, if the context is considered or interpreted incorrectly, the timing of the feedback may become inappropriate and the user easily feels distracted. Furthermore, the disregard of the user's expertise while displaying visible information and choices increases the user's mental workload. If the context-aware software manipulates the representations of information permanently, it constricts the user in predicting and learning the behaviour of the software.

Controllability

Users should be able to use all of their objects in any sequence and at any time, and the application should be as interactive and responsive as possible. In order to give users control over the system, they need to be enabled to accomplish tasks using any sequence of steps that they would naturally use according to their needs and characteristics. The user plays an active rather than a reactive role. The application developer can automate tasks, but needs to implement the automation in a way that allows the user to initiate or control it. User control in this context means that if interactions are reversible, it should be possible to undo at least the last dialogue step.

"*A dialogue is controllable when the user is able to initiate and control the direction and pace of the interaction until the point at which the goal has been met.*" (ISO9241-11, 1998)

Context-aware interfaces are flexible in a way that they are able to accommodate a wide range of user skills, physical abilities, interactions, and usage environments. Most users perform a variety of tasks, being expert at some and novice at others. A context-aware application could recognise and anticipate the user's goals, and offer assistance to ease the task (so-called pro-active assistance). Ideally, assistance should provide users with knowledge that will allow them to accomplish their tasks quickly. The assistance should allow them to become independent at some point when they choose to be so.

Contrary, if the context-aware application automates activities without giving any feedback to the user, the user's freedom of choice of the most appropriate action sequence is restricted. A misinterpretation of the user's needs and characteristics may result in a presetting of inappropriate levels and methods of interaction and may limit the user to specific interactions. Making normally available actions no longer available or having an action cause different results than it normally does, restricts the user's ability to control over how to continue with the dialogue.

Expectation Conformity

The application should allow users to build on prior knowledge, especially knowledge they have gained from experience in the real world. A small amount of knowledge, used consistently throughout an interface, can empower the user to accomplish a large number of tasks. Users should not have to learn new things to perform familiar tasks. The applications should contain concepts and techniques that users already understand from their real-world experiences and allow them to apply this understanding in a variety of situations, and thus, start quickly and make progress immediately.

"*A dialogue conforms with user expectations when it is consistent and corresponds to the user characteristics, such as task knowledge, education and experiences, and to commonly accepted conventions.*" (ISO9241-11, 1998)

A context-aware behaviour of the dialogue assists the user in applying interactive skills they have already learned, allows transferring existing knowledge to new tasks and focussing more attention on tasks. If the system is able to understand the users, their tasks and their shared experiences, the interface can be adapted to fit to the user's habits and provide a familiar experience.

A context-aware application that draws wrong assumptions about the context eliminates the sense of stability because the users have to spend time trying to remember the differences in interaction. An additional critical issue arises if the context-aware behaviour differs from the appearance of the dialogue. If the context-aware application presents dissimilar dialogues for similar tasks, the user cannot develop common task-solving procedures.

Error Tolerance

Users like to explore an interface and often learn by trial and error. An effective interface allows for interactive discovery. An effective design avoids situations that are likely to result in errors and eliminate the opportunity for user error and confusion. The application should prevent any user input from causing undefined system states or failures and additional controls should be provided for commands with serious consequences.

"*A dialogue is error-tolerant if, despite evident errors in input, the intended result may be achieved with either no or minimal corrective action by the user.*" (ISO9241-11, 1998)

A user's mistakes can be physical, e.g. accidentally pointing to the wrong command or data, or mental, e.g. making a wrong decision about which command or data to select. A context-aware application, which takes the physical and mental state of the user into account, detects errors early and can propose appropriate corrections. A context-aware dialogue system is able to correct errors automatically. However, it should advise the user of the execution of the corrections and provide the opportunity to override the corrections.

On the other hand, if the context-aware application does not make automatically executed actions reversible or recoverable without any notification, the system could be put into an unstable state or damaged. If the system displays rather error-unrelated information in a clarification or confirmation dialogue through misinterpretation of the current error situation, the user may come to a wrong and task-unspecific decision to continue.

Individualization Suitability

The interface should be customizable to individual users' needs and desires because no two users are exactly alike (cf. Section 2.1.3). In particular, differences in their culture, levels of experience of the task domain, as well as physical, perceptual and cognitive abilities motivate users to individualize their interfaces. Since such a customization can help make an interface feel comfortable and familiar, mechanisms should be provided to allow the dialogue system to be adapted,

"*A dialogue is capable of individualization when the interface software can be modified to suit the task needs, individual preferences and skills of the user.*" (ISO9241-11, 1998)

This usability criterion basically aims at a manually performed personalisation of a computer interface that can lead to higher productivity and user satisfaction. In this case, automatic context-aware behaviour may become apparent in the proposal of suitable customisation procedures and the retention of manually executed adaptations in order to recall them in specific situations later on. This would lead to individualisation in or for specific situations. Furthermore, a context-aware application could make the set of all automatically executable adaptations available to the user for manual execution.

Learning Suitability

Learning by doing is supported by encouraging the user to experiment, walk through examples during various situations, allowing error correction without the danger of causing potentially negative results. The interactive discovery of the interfaces and the application behind enables the user to learn about the application's internals. Users should feel confident in exploring, knowing they can try an action, view the result, and undo the action if the result is unacceptable.

"*A dialogue is suitable for learning when it supports and guides the user in learning to use the system.*" (ISO9241-11, 1998)

A context-aware dialogue could make rules and underlying concepts which are useful for learning available to the user, thus allowing the user to build up own grouping strategies and rules for memorising activities. In addition, the user should be able to obtain information on the model on which the automatic adaptations are based.

However, if the context-aware application frequently adapts the layout of interface elements in an irreproducible manner, the facilities for learning and comprehension may be limited because of the absence of different means to help the user become familiar with the dialogue elements.

2.7 Summary and Discussion

This section summarises the main contributions of this chapter and discusses the implications of the usability problems context-aware applications introduce.

2.7.1 Summary

The exposure to situational dynamics and changing environments cause today's applications to shift the customization process of the application characteristics from the development phase to the operational phase. Developers implement methods of adaptation to react to such changing conditions as fast as possible (cf. Section 2.1). The two adaptation methods adaptivity and adaptability are complementary to each other and aim at increasing the match between user needs and system behaviour once the software engineering process has been finished. Context-aware applications primarily base on automatically performed adaptations without or with little user consultation.

The active involvement of other actors than developers in the creation and operation of usable applications requires a comprehensive understanding of the notion and application of context that can be communicated to the entire spectrum of actors. The examination of common definitions of context provided by the research community in context-aware computing revealed a variety of different meanings and alternative views of context. Previous definitions of context suffer from either generality or incompleteness (cf. Section 2.3) and fail to establish any fundamental basis of their construction because they are basically driven by the ease of implementation.

The core contribution of this chapter lies in the introduction of a context definition that tackles these drawbacks and aims at bridging the user-developer gap because it provides a natural understanding of the concept for users and eases the engineering of the concept for software developers. This definition comprises three canonical parts (cf. Section 2.3.2): a definition per se in general terms, a formal definition describing the appearance of context and an operational definition characterising the use of context and its dynamic behaviour.

The presented definition promotes an entity-centric view of modelling a context around each entity of the domain. The information describing the context of an entity is structured into five fundamental categories of context information that determine the design space of context models (cf. Section 2.4). This categorisation achieves an integration of user models and context models and empowers designers of context-aware applications to increase the application's ability to adapt to the user because the external context alone may inadequately determine the most appropriate adaptation to the individual user. In addition, the context definition fosters a systematic foundation for the use of context in context-aware applications and introduces an operational aspect of context. This operational definition of context emphasises the dynamic properties of context emerging from context transitions and sharing contexts among entities (cf. Section 2.5).

The provided definition of context in combination with an accordant detailed justification conveys a sense of the employment of context in context-aware applications. This definition is able to express and to cope with inter-individual, intra-individual, and environmental differences of the domain entities. However, the potential for designing a context-aware application that performs the wrong action and seriously annoys the user still persists. Consequently, context-aware applications are associated with a number of typical usability problems (cf. Section 2.6), and in some cases, a decrease of the system's usability outweighs the benefits of adaptation. Due to the lacking transparency of context-aware applications, the adaptation decisions are inaccessible to the end-user, which disallows an overriding of the behaviour. Without control and means of customization the end-users will abandon useful context-aware services.

The following section continues with a discussion on these usability aspects and leads over to an approach ensuring an adequate fulfilment of the usability goals without eliminating the benefits of the adaptation processes.

2.7.2 Discussion

Changing requirements and dynamic environments are drivers for context-aware applications. The automatic execution of system adaptations leads to a successful and effective information and communication system that provides information and functionality that is relevant and at the right level of complexity with respect to the current situation. The objective of context-aware applications is to assist the users by pro-actively supplying what is actually relevant and needed. Thus, such a system enhances the quality of system usage through tailoring aspects like the supplied information, functionality or presentation.

The work of the user can be very complex and correspondingly difficult and cognitively demanding. A context-aware application is able to recognise when a user is cognitively overloaded or even loosing control, and can automatically invoke several countermeasures to support the users or take over control. Through automation users are not distracted from their primary task by searching and selecting. Adaptivity can help reduce the cognitive load on the user by hiding specific functionality that is not pertinent in the current context. Additionally, a

context-specific selection of the in- and output modality results in a lessened deviation of the user's attention. Highly autonomous systems decrease mental effort in the interaction with the system because it requires less input and thought on the part of the user.

A continuous update of the context model results in the severe disadvantage of a complicated creation of a consistent usage model. While the user tries to understand the system and to construct a usage model, the system builds an internal context model and changes its behaviour accordingly. In turn, the user tries to understand this altered appearance and to integrate this experience with the current usage model. This phenomenon is the normal case in human-human communication, but may lead to the impression of inconsistent system behaviour in human-computer interaction.

Automatic adaptations of the system are only possible to the extent that the system can gather the required information basis of performing some adaptation function. Context information may be unreliable, inaccessible, difficult to acquire or results from an interpretation process taking very little recorded operation steps into account. However, many context-aware applications need to base their adaptation decisions on this approximate and limited nature of context information, and consequently, usability problems arise from wrongly applied context-awareness (Flintham et al., 2003; Marmasse and Schmandt, 2000). Even if an adaptation would functionally succeed in spite of this little information, the later result would be adaptations that lead to an increased disorientation of the user.

Adaptivity reduces the user to a passive recipient of automatic mechanisms. A context-aware application can operate with limited user intervention because it places a greater dependence on contextual information. In general, the user's sense of control decreases when the autonomy of the application increases. However, users are willing to accept a large degree of autonomy from applications as long as the application's usefulness is greater than the cost of limited control (Barkhuus and Dey, 2003). On the other hand, the more users know about the system and its potential the more the user demands control over the offered functionalities (Frese, 1987). Thus, a more experienced user exhibits advanced requirements on controllability.

2.7.3 Adaptable Context-Aware Applications

The demand for a far-reaching participation of the user in the adaptation process can be understood as an aspect of controllability. However, if the methodology of the system's automatic adaptivity as implemented by a developer is hidden from the user, she can neither comprehend nor anticipate the modified system behaviour. Even if the user expects working with an adaptive and dynamic system, the creation of a consistent usage model becomes more difficult because she needs to develop an additional mental model of the system's adaptivity. Therefore, the user's mental model of the system needs to be consolidated by a context-aware application revealing its current understanding of its automatic functionality. Furthermore, the user requires means of the correction of this understanding in case of the adaptation process producing unwanted results. By this means, the user can exploit her superior awareness of her

context and her domain-specific expertise in order to customise the application's adaptivity processes. Such a customisation may additionally exhibit dynamic aspects because the context-aware application should allow the user to do some work and gradually shift the control to the system during usage.

Anderson et al. (2003) elaborated on the controllability of automatically executed actions and claimed context-aware applications that ensure that actions they take on behalf of users are both intelligible and accountable. For the user, the transparency of the application's capabilities and its understanding of the current context for the user, the disclosure of actions taken by the system as well as the provision of feedback to the user and mechanisms of user control amplify the accountability of a context-aware application (Bellotti and Edwards, 2001). A context-aware application that features these properties is defined as an *adaptable context-aware application*:

Definition 10: Adaptable Context-Aware Application

> *An adaptable context-aware application empowers its users without or with limited programming skills to customize or tailor the context-aware behaviour according to their individual, context- or situation-specific requirements.*

The adaptability of context-aware applications not only requires an increased qualification from the user, but also contributes to obtaining this qualification. The user achieves qualification prior to and during the usage of the application through a deeper understanding of the system and the means offered for customisation. The acquired knowledge and skills required for the accomplishment of specific customisation enables the user to process more difficult tasks and to master more complex error situations. An adequate qualification of the user plays a significant role because the user's qualification is regarded as a constituent of the human-centred design process and constitutes one characteristic of the resulting system. Apparently, an adaptable context-aware application significantly contributes to and enhances learning suitability and individualisation as well.

The costs that a user needs to spend on the adaptation of the context-aware behaviour have to be rated against the costs that emerge from the further usage of the non-adapted behaviour. Hence, an immediate dependency between the frequency of the usage of adaptable context-aware application and the simplicity of how the adaptation can be performed results from this observation. Therefore, the adaptation performed by an occasional user has to be easy and without any additional work load. In contrast, expert users or users with development skills require advanced adaptation functionality. Several levels of adaptation expertise are required. In addition, with increasing expertise and developed routine work, the experts may aim at a partial automation of procedures.

The inspection and correction of the application's understanding of its context-aware functionality as well as the shifting of control positively contribute to the achievement of usability goals pursued by adaptable context-aware applications. For the realization of such

applications and a deeper integration of the users into the automatic adaptation processes, developers will need tools, development frameworks and software libraries. The following chapter investigates if the development support provided by the current state of the art can be applied or extended for these purposes.

Chapter 3

Research Framework

This chapter presents a discussion and evaluation of the state of the art regarding tools, infrastructures, or other development support facilitating the construction of adaptable context-aware applications. The complete research framework divides into three segments: the research direction of this thesis, the analysis of the current state of the art and a concluding assessment based on key requirements. The research direction presented in Section 3.1 provides an assessment framework for the subsequent investigation of the state of the art because it outlines research aspects of concern for this thesis, and thereby, constricts the set of relevant approaches. Guided by this research direction, Section 3.2 conducts the survey of the state of the art and examines selected approaches of particular relevancy for this thesis. These works are still ongoing and driving research in this area due to their prominence and outstanding features. The insights provided by Section 3.1 and Section 3.2 form the basis of the derivation and identification of key requirements that allow for a detailed assessment of the existing approaches. Based on these key requirements, Section 3.3 highlights the shortcomings of the existing approaches and motivates the research pursued in the following chapters.

3.1 Research Direction

This section constricts the coverage of the subsequent analysis of the state of the art and provides an assessment framework for the approaches investigated in the following section. Recent research into context-aware computing is directed towards the extension of actors participating in the creation of context-aware applications. The thesis' claim for the provision of development support for adaptable context-aware applications further condenses this research direction. The elaboration of the research direction bases on various information sources: Scenario descriptions of context-aware applications, open research questions in context-aware computing, lessons learnt and experiences gained during the prototyping of context-aware applications.

Making context-aware applications adaptable and accessible for everyone affects three main aspects of the realization of such applications: the knowledge base, the development and the operation of context-aware applications. These three main aspects group key characteristics that provide a better understanding at a lower level of abstraction. All three aspects are intertwined at certain points of contact and demands described for the first aspect still hold for the third aspect. The following subsections describe the impact of the extension of the actors in the realization of context-aware applications on the three mentioned aspects in more detail.

3.1.1 The Knowledge Base of Context-Aware Applications

The knowledge base of a context-aware application basically comprises a context model as a representation of context information. Section 2.3.2 provided definitions of the correspondent terminology and emphasised the strong connection of the context model with the specific application domain. During the creation of the knowledge base of the context-aware application, the developer explicitly defines and tailors the context model. In order to grant other actors than developer access and insight into the knowledge base, a uniform understanding and perception of context needs to be achieved. On the one hand, expressing context in a model that users can conceive would put the aspect of involving the end-user in the focus of attention, but inhibit effective processing context information by the application. On the other hand, however, a context representation that is tailor-made for an optimised machine processing of context is unsuited for users for the most part.

A transparent context model aims at the reduction of the mismatch between the system's model of the interaction context and the user's mental model of the system (Coutaz et al., 2005). A uniform structure of context information as described in Section 2.4 ensures that all parts of the context-aware application share a common understanding of the syntax and semantics of context, which constitutes an essential prerequisite for its processing. The realization of the understanding of context as introduced and defined by Chapter 2 requires the consideration of four major aspects described by the following subsections.

Represent User Characteristics

In user-oriented systems the adaptation of the system behaviour to its current user is the main focus of interest. The foundation of this intension is the representation of the personal characteristics of this specific user. The context-aware application bases its decisions on the evaluation of the user behaviour and automatically performs adaptations that meet the user's necessities. Therefore, the context model needs to include user characteristics as described in Section 2.4.2. In addition, the context model needs to express as many types of assumptions about the user as possible at the same time. Furthermore, humans may interpret the same context differently, and thus, several views on the same context exist. The context model needs to reflect this individualised weighting of context attributes.

The representation of user characteristics provides the basis of user-related adaptation: Physiological characteristics like disabilities are of major concern for application designers if

they want to have their system accepted by a large community. The consideration of user preferences like language, colour schemes, modality of interaction, menu options or security properties, and numberless other personal favourite preferences is one of the most popular sources of adaptation and can be reused in different applications. An important prerequisite for the content adaptation is the awareness of the user's interests and disinterests. The user's interests configure filters for the information presented to the user. Additionally, a broad spectrum of psychological personality characteristics exists like emotions, self-confidence, motivation, beliefs or idols, which are difficult to asses automatically. The same holds for the user's level of factual, general and domain-specific knowledge (e.g. beginners or experts), as valuable sources of adaptive operations.

Enable Context Comparisons

The nature of context is that it is shared and the interpretation of context relies on the shared understanding of this concept. Section 2.5 elaborated on the meaning of shared context for establishing and exploiting relations between entities (e.g. museum visitors in front of the same painting). Therefore, comparisons of context information are essential for the determination of similarities or differences between domain entities. Information about conjoint characteristics a user shares with other users, or about individual differences, also contribute to the characteristics of a specific user. Furthermore, patterns in the behaviour or user groups with identical interests, goals, or levels of knowledge can be derived.

Many context-aware applications filter information or services out of a large repository in order to better meet user's information demands in the current context. Therefore, such applications annotate content or services with context information, which allows for an intelligent allocation of these two entity types to the user's current situation at a later stage. The retrieval of an appropriate content or service bases on the comparison of the context information associated with the content or service with the context information available in the user's current situation.

Enabling context comparisons starts with an accordant preparation of the context model and needs to be sustained by corresponding programming abstractions provided to the application developer. The required context comparisons depict a strong variety in complexity. Basic comparisons involve simple checks of values in the context. The complexity increases with the number of values involved and the level of abstraction these values are represented with. The transition from strict regular expressions to more fuzzy similarity measures introduces additional complexity into the calculations. The support of context comparisons must be broken down into two requirements: First, the context model needs to enable context comparisons. Second, the programming needs to provide basic and extensible operations and algorithms facilitating context comparisons.

Provide Mechanisms of Context Transformation

A significant increase in the expressiveness, complexity, and quality of the represented context information can be achieved by transforming the acquired context information in several ways. Common context transformation mechanisms are fusion, aggregation, interpretation and derivation. Mainly these techniques differ in the knowledge that is required for their application. Fusion refers to the consolidation of multiple pieces of information of the same type in order to increase the quality. Information pieces of different types may be subject to aggregation if one piece cannot provide the required information by itself. A semantic model forms the basis of an information interpretation process in order to gain high-level information. Another context transformation mechanism constitutes derivation, which takes the acquired context information as an input of inference algorithms. Context transformation mechanisms merge various aspects of the acquired context information into a coherent whole. In addition, such mechanisms enable the detection of relations between entities, and therefore, form the basis for the exploitation of additional context information.

Offer Metrics of Context Quality

Unexpected incidents happening during the acquisition of context information affect the quality of the acquired context information in a way that it might become incorrect, incomplete, or inconsistent. Incorrectness results from imprecise measurements, incompleteness is an effect due to breakdowns, disappears or jams, and inconsistency emerges from ambiguous values acquired by multiple acquisition methods. In addition, context information is highly dynamic and may quickly become outdated. Even information explicitly provided by the user may show varying quality characteristics. The user is not always willing to divulge her personality characteristics or knowledge to the system mostly due to psychological constraints and to fear of unintentional consequences. However, an automated cognition of these characteristics without explicit user input is difficult and subjected to errors.

Many architectures carry shortcomings and quality characteristics of the acquisition methods up to the application level and leave the decision on the reliability of the context information up to the context consumer. Prior to affecting the adaptation process, the acquired context passes through multiple layers of transformations. Each of these transformations may conceal a certain degree of uncertainty compiled into assumptions the transformation may rely on. Both the context model and the entire architecture need to allow for incorrect, incomplete and inconsistent values of the acquired context information and should support the determination, representation and supply of quality metrics characterising the acquired context information at various levels.

3.1.2 The Development of Context-Aware Applications

Many context-aware applications represent specialised applications (Weber, 2002), tailored to one domain or environment and rarely reusable in other applications. Additionally, the logic

is often programmed directly into the behaviour of the system and this tight coupling leads to rigid implementations and are difficult to maintain. Recent research into programming frameworks that developers exploit for the implementation of context-aware applications limits its focus on tasks like acquisition of context information from sensors, persistent storage of this information within servers and reduction of complexity. However, as inexperienced developers, designers lack conceptual tools and methods to jump-start the development of context-aware applications and still face high application development overheads.

The extension of the spectrum of actors in the development of context-aware applications beyond developers requires an application-oriented view of the software engineering process and an elaboration and exploitation of context at multiple levels of abstraction and complexity. A programming framework, reference architecture and design support need to be combined into a single comprehensive programming and maintenance support. In addition, the development support of context-aware applications needs to address further aspects as described by the following subsections.

Accomplish Holistic Context-Aware Computing

The research into context-aware applications suffers from putting too much focus on input and too little focus on output (Salber, 2000). The output of context-aware applications means both the control of actuators realising real-world actions and in equal measure the control of actions modifying the application's internals. Some changes in context might require some modification of the information contained in the context model or even of the structure of the context model. Recent research results on context-aware applications lose sight of control components that decide what consequences must be taken if certain conditions in the context model appear together. Furthermore, flexible actuator components need to reflect upon these consequences and to map them onto real-world actions. Moreover, a feedback mechanism is required that reports about the success or failure of any action executed. The exploitation of quality measures associated with context information can be taken as an example: The context consumer is in charge of this information deciding on its reliability. Up to now, consumers of context information are not intended to invoke countermeasures like switching on or off sensors that deliver low quality.

Context-awareness starts with the adaptation goal a designer or developer wants to achieve with this methodology. Then, characteristics of the context and the users must be defined based on the purpose of the application (Byun and Cheverst, 2001). Therefore, any development support needs to formalise adaptive methods and make them portable. The integration of holistic, systematic context-awareness into the development process of interactive systems is aspired (Coutaz et al., 2005).

Couple Context-Awareness with the Core System

A tool suite supporting the realisation of context-awareness within applications that lack that functionality may offer this support at several levels. A lightweight support level could be the provision of a single functionality like the encapsulated access to sensor values and a heavyweight level could be the infrastructure's support of the design, development and deployment of a context-aware application as a whole. All intermediate levels of support and integration with the core system might be conceived. In general, the integration methodology is bound to the semantics of the actual target application. However, the application developer decides on the way that permits certain integration. During the realisation of the context-aware application different tasks should be split among already existing parts of the target application and the infrastructure providing context-aware functionality.

The delivery of information dependent on the context as one specific instantiation of context-aware behaviour depicts an additional plausible need for a close coupling of the context-aware functionality with the core system: Adaptivity cannot be separated from the content (Weber, 2002). In order to enable a closer coupling in this case, the data model and the context model must not be separated (Belotti, 2004) and a link between content and context needs to be established. Allowing multiple levels of integration of context-aware functionality with the core system depends on the modularity and on well-defined interfaces provided by the development support.

Establish a Formal Processing Pipeline

The absence of a uniform structure of an architecture, which ensures a common understanding of the syntax and semantics at all parts of the application, complicates sharing of context information with other parts of the system. Apart from the software structure, which deals with the static aspects of the component's computational elements, the processes, in which these elements are involved, are of equal importance. An examination of the processes executed within the software architecture provides information about the architecture's dynamic aspects. In order to overcome the lacking standards for exchanging information, a generic architecture and formal processing pipeline has to be investigated and a straightforward application programming interface needs to be provided. Furthermore, the type of exchanged information is not limited to context information because context-aware applications also control and manage system components. The information flow happening between major building blocks of context-aware applications needs to be identified, understood and formalised as well as their orientation, i.e. the direction of information flow. This understanding contributes to the achievement of a modular and reusable architecture, and enables a structured and consistent processing of information.

Formalize the Design of Context-Aware Applications

A general conduct in the design of context-aware applications is that the context acquisition mechanisms drive the choice of context information processed by the application. This design

procedure might not be the most appropriate method and limit the kinds of applications. Approaches of designing context-aware applications are currently in their infancy, with the present generation of applications largely being developed and tested only in constrained laboratory settings (Henricksen, 2003a). The design process of context-aware applications is poorly understood because only the implementation phase is addressed, but analysis and design aspects are elided. Some researchers have enumerated the design and implementation tasks involved in developing software using their models, however, the tasks are almost always described informally, and without adequate guidance in terms of the process or complex issues that are involved. The software engineering process associated with context-aware applications (in terms of methodologies) needs to be understood as a whole. The design process needs to deal with the exploration and specification of an application's context requirements and must be assisted by design tools for authoring, administration, maintenance and deployment of context-aware applications.

Provide Programming Models and Abstractions

General software design principles need attendance through specific programming models and abstractions that relate to context-aware computing. A programming framework must provide developers with a set of core abstractions that comply with a single computational model and architecture style. In order to reduce high application development overheads and to allow expressing context-aware behaviour in high-level terms, many gaps in programming models need to be filled. Firstly, the context acquisition methodology needs to be extended in order to deal with both implicit and explicit sensing (Salber, 2000). The details of the interaction with sensors for context information and explicit interaction with users must be encapsulated within specific components and hidden from other application parts. Both kinds of values, implicitly or explicitly acquired through several sensing techniques need to be transformed into a uniform representation. Secondly, programming abstractions for several ways of context information exploitation are required because based on the retrieved information actions may automatically be taken that range from context management functionalities over reasoning towards real-world actions. The control of the adaptive behaviour during the implementation and operation phase must be facilitated. Additionally, developers need to be provided with support through mechanisms like triggering of events, using context as explicit query and detecting specific situations. The linking of context to content at production time and the use of context during retrieval is an enabling concept for many types of context-aware applications as well. Finally, the application of personalisation techniques needs to be facilitated in order to address the integration of context-awareness and user modelling. Therefore, universally applicable learning algorithms need to be integrated into the aspired development support to allow developers to gain knowledge about the user.

Offer Reconfiguration Functionality

Independently developed context-aware applications are constructed under a set of assumptions that the developer had to make about the target operating environment. In

addition, context-aware behaviour is often statically incorporated into the application code. Simplicity is a main argument for the programming support of developers of context-aware applications and thus, reconfiguration and customisation are the prerequisites for the universal applicability of a general programming framework of context-aware applications. The functionality of components like actuators and sensors as well as adaptive and reasoning behaviour needs to be parameterised to allow for configurability. Technologically different environments and the introduction of various third party components make the system accident-sensitive and may force the application to automatically take action for debugging and repair. Such context management functionality bases on the ability to self reconfiguration and provides mechanisms of conflict resolution and performance control. Context management functionality comprehends measures like switching to sensors providing the highest quality of information with minimal impact on the context gathering process as a whole.

3.1.3 The Operation of Context-Aware Applications

The developer of a context-aware application designs the context-aware behaviour of her application usually with little user consultation. In addition, this behaviour is often hidden from users or difficult to override, which introduces usability problems. The reduction of such usability problems associated with context-aware computing requires the extension of the actors involved in the design, implementation, authoring and configuration of context-aware applications beyond developers. For the operation of context-aware applications this means the involvement of end-users in the adaptation process. However, the importance of user involvement in the operation of adaptive systems has often been neglected in current systems (Efstratiou et al., 2001).

Section 3.1.1 already addressed the required transparency and intelligibility of the knowledge base built into context-aware applications, in order to reduce the mismatch between the system's model of the interaction context and the user's mental model of the system. Furthermore, Section 2.1.3 described the diversity of possible operation environments of context-aware applications as well as intra- and inter-individual differences among users. During their operation, context-aware applications need to consider accommodate the differences between several users as well as the dynamics of single user requirements as the user's activities and goals evolve. The following subsections illustrate aspects of how to involve end-users in the operation of context-aware applications.

Integrate Adaptivity and Adaptability

Automatic context-aware adaptivity and manual adaptability are important features of computer systems (cf. Section 2.1.2). Both approaches aim at a sustainable catering to the users' needs (Klann et al., 2003) and both depict their specific limitations and disadvantages. In particular, the first is incomplete and imprecise and reduces the users to a passive recipient of automatic mechanisms. The latter is costly, requiring additional effort from the users for

the meta-task of tailoring the system. Adaptivity benefits from adaptability if the required information for some adaptation is not available, unknown, or ambiguous. Adaptability benefits from adaptivity if the user-performed adaptation appears to be very complex, correspondingly difficult and cognitively demanding. Both adaptation processes should be integrated into a coherent whole to gain a more accurate match of the system behaviour with the user's expectations. This integration of adaptivity and adaptability results in a shared initiative between the user and the system (Oppermann, 1994; Oppermann, 2005).

The integration of automatic adaptivity and manual adaptability starts with permitting user control over the application's actions. Context-aware applications need to provide mechanisms where the control of the adaptation can be reconfigured without the need for reimplementation of application parts. Any programming framework or infrastructure should enable the end-users to perform modifications and to explicitly specify how the system should function. In order to let the users modify the adaptive behaviour of application to better suit their needs, the continuum of control mechanisms needs to be identified and described. Potentially, such mechanisms may range from the manual disambiguation of vague or low-quality information to sophisticated end-user configuration techniques.

Support Transparency and Reflection

The users need to understand and inspect the system's mode of operation in order to decide on what modifications to carry out. Inspection, in turn, requires the application to externalise its current state as well as its composition. Externalisation or reflection means the provision of a human-readable description of components (e.g. sensors) plugged into the system, information about the current state of these components (e.g. activated or deactivated), and configuration settings needed for launching and operating the system. Any development support should grant the user read-only access to state information and read-write access to configuration files. Currently, there exists a lack of transparency concerning the application's decision and actions. Since acquisition and interpretation processes may result in imprecise or ambiguous information, a context-aware application needs to be able to provide feedback about potential error-prone processes as a prerequisite, particularly when the consequences are important to the user. This requirement is not advocating a constant feedback about all events, but architectural support of the provision of reflection mechanisms about the developer's determinations if the user wants to understand the system's operation.

Provide Design Support at Several Expertise Levels

Product managers, application designers and system integrators know best about demands for context-aware functionalities in their application. In parallel, they have to react to a shortened delivery time demanded by a competitive and dynamic market. These actors in the creation of context-aware applications face crucial challenges because they possess low development skills and have to handle the system heterogeneity starting from the hardware and software protocols, including the integration within various mobile and wireless environments, and

considering seamless integration of services from multiple providers. The restriction of the programming support to developers is insufficient because actors with less development experience should also be able to predefine, author and maintain the behaviour of context-aware applications, and end-users should further tailor them to their specific needs. A supportive programming framework accompanying the entire life cycle of context-aware applications should comprise a tool suite or workbench composed of tools facilitating the deployment, administration and authoring of context-aware applications in order to support actors at several levels of expertise and with different development skills.

Integrate Context-Awareness and User Modelling

There exists a definite need for user modelling to be applied to context-aware applications (Barrett and Power, 2003). Context-awareness and user modelling share a lot of requirements: Both research activities aim at the same or similar goals regarding architectural support and conceptual understandability, e.g. generality or strong inferential capabilities. Moreover, they both can be subsumed beneath the general concept adaptation because they both exploit knowledge to adapt the behaviour of applications. Context-aware applications may receive an impulse towards the reduction of usability problems if they maintain a set of assumptions about their users and not only react to changes in context. In turn, context-awareness contributes to user modelling by providing easy access to sources of information that remained untouched so far. Since both research directions overlap through many similarities and depict a parallel evolution, an investigation of a possible integration of context-awareness and user modelling needs to be made in order to let context-awareness benefit from user modelling and vice versa.

Preserve the User's Privacy

In context-aware computing applications gather and maintain a high amount of contextual information about users and entities. Since much of this information may refer to very personal and confidential aspects of people's lives, e.g. their current activities or moods, most people do not want this information to be entirely revealed to any other person. Context services that handle extremely sensitive data cannot release context information without validating with the user. Users should be in control of their context information and decide which sources collect and supply data about them. Moreover, users should be notified about the collected information type and have the choice to halt the collection of personal information. Existing context-aware applications as well as toolkits supporting the development of such systems weakly address or entirely ignore privacy issues so far. A related requirement is the avoidance of embarrassing situations by the consideration of information security and privacy. A camera as a sensor placed in a bathroom or inconvenient call forwarding can be cited as examples of such situations. Basically, a distinction between public and private sensors and actuators, and a controlled reception and allocation of public and private information needs to be formalised.

3.2 State of the Art

The previous section enumerated the research directions that delimit the boundaries of the research conducted by this thesis. These research claims provide a framework driving the evaluation and assessment of the current state of the art. This section examines the state of the art regarding programming toolkits and software infrastructures that facilitate the design and development of context-aware applications and involve the end-user in the entire software engineering process. In particular, each paragraph addresses one relevant approach or research project, describes the evolvement of this system over time, and discusses its appropriateness with regard to the underlying research demands.

3.2.1 The Context Toolkit

The Context Toolkit developed by the Future Computing Environments Group at the Georgia Institute of Technology provides programmatic support of the development of context-aware applications (Dey, 2001; Dey et al., 2001; Dey et al., 1999b). The toolkit incorporates various services related to the gathering and supply of context, including an encapsulation of context, access to context data, context storage, and a distributed infrastructure. The Context Toolkit defines the following four main programming abstractions:

- *Widgets*, which encapsulate complex sensor functionality
- *Interpreters*, which raise the abstraction level of context information
- *Aggregators*, which combine different types of context information
- *Services,* which realize changes of the environment

The context widgets reside between the context-aware application and the environment, which is comparable to the way a Graphical User Interfaces widget resides between the application and the user. The widgets are software wrappers for providers of context information, and thus, separate the application from context acquisition issues. One context widget hides the complexity of gathering and managing contextual information, and therefore, encapsulates one single piece of context. Other components or applications exploit this information through a uniform interface. Widgets encompass historical context information using a relational database for persistent storage.

The interpreter abstraction implements context data derivation and transforms context data gathered from sensors into higher-level contextual information. Another fundamental abstraction is the concept of aggregators that can be considered as meta-widgets, which act as gateways between applications and widgets. They aggregate contextual information referring to real-world entities.

The three components discussed so far focus on the acquisition of context and delivering it to interested applications. Context services are an analogy to the context widget and represent components in the framework that execute actions on behalf of applications. They are

responsible for controlling or changing state information in the environment using an actuator. Context services can be synchronous or asynchronous and developers can design and implement services that can be made available to multiple applications.

In addition, the Context Toolkit contains a discoverer which is responsible for the dynamic discovery of appropriate components. These components function independently from applications, and thus, grant several other components or applications the exploitation of the same context information. In order to support transparent distribution, the Context Toolkit relies on a common communication mechanism based on the Hypertext Transfer Protocol (HTTP) and the eXtensible Markup Language (XML). Figure 9 illustrates the interplay between the components of the proposed architecture.

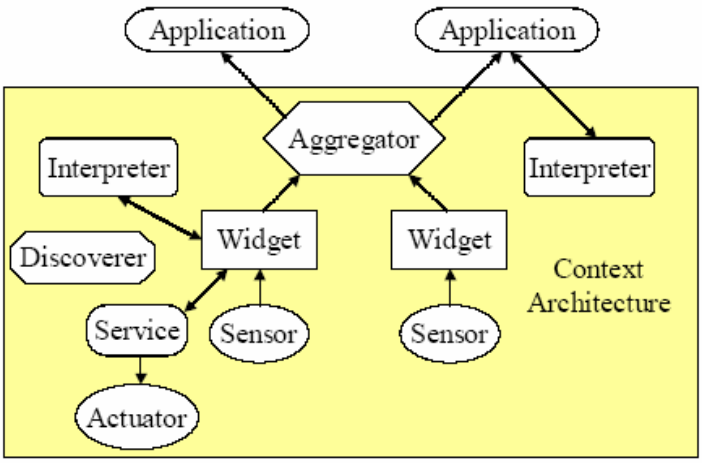


Figure 9 Components of the Context Toolkit (Dey et al., 2001)

The Context Toolkit's principal target is to provide programmers with standard libraries of reusable components for context handling that facilitate the integration of context gathering functionality with existing applications to enable context-aware behaviour. Furthermore, it formulates a design process for building context-aware applications. The utility of the model is demonstrated by an array of prototypical implementations that base on the Context Toolkit. These exemplary applications include a location-aware information board called In/Out Board (Salber et al., 1999), a reminder tool (Dey and Abowd, 2000) and a context-aware conference assistant (Dey et al., 1999a).

Based on this fundament, further research has been achieved by this research group more recently. An additional abstraction provided by the Context Toolkit is the so-called situation abstraction (Dey, 2001), which resides above the aforementioned components of the toolkit. Since a situation is understood as a description of the states of relevant entities, the situation abstraction allows application designers to interact with the infrastructure as a whole instead of querying and subscribing to several components individually.

As a special toolkit extension, the enactor component exposes the application state information in an accessible way via a standard application programming interface

(Newberger and Dey, 2003). This component can be leveraged for building separate user interface components that allow monitoring and control. This allows user interface components such as Macromedia Flash components to easily communicate with enactors. These user interface components can support monitoring and control of the enactor, thereby facilitating the fine-tuning of its behaviour by the designer. Enactors comprise the following standard subcomponents:

- *references* for acquiring context data from widgets
- *listeners* for monitoring changes, and
- *parameters* for controlling the behaviour.

The enactor concept improves end-user experience in context-aware applications by improving monitoring and control.

Dey and Mankoff (2005) present a way of handling ambiguity of sensed and interpreted context. They describe an extension of the Context Toolkit supporting the mediation of ambiguous context, where the term *mediation* refers to the dialogue that ensues between the user and the system. Mediator components either resolve ambiguity automatically or allow the user and computer to communicate about ambiguity. Mediators generally fall into the categories *choice, repetition* and *automatic* mediators depending on the way the user is involved in the mediation process. The handling of ambiguity may be explicit, i.e. the mediator is a dialogue, or implicit, i.e. the mediator displays information at different levels of precisions appropriate to the amount of ambiguity present. If no user is present, the action is paused until the uncertainty is resolved.

The toolkit automatically handles the tasks of routing input to mediators when it is ambiguous. After the ambiguity is resolved, the toolkit informs recognisers and the application about the correct result. This separation of mediation from recognition and from the application means that the basic structure of an application and its interface does not need to be modified in order to add to or change how recognition or mediation is accomplished. An application may opt to bypass mediation and receive ambiguous events directly or ask to only be informed about events that are unambiguous or have had all ambiguity resolved.

Even though the Context Toolkit unites years of experience in context-aware computing and represents the most complete understanding of this area, this solution is primarily intended to be used by application developers and flaws can be identified. The main flaw of the Context Toolkit is the absence of a formal model for context because it is represented by a set of attributes of the context widgets defined in an ad hoc manner. The situation abstraction takes a step in the direction of a formal context model and may allow end-users to perform a further specialisation of context-aware applications to meet their individual needs, but the support for this abstraction is limited. The toolkit provides mechanisms of context derivation by means of interpreters, but their functionality is very restricted as they are usually employed for simple data type conversions. As a result, the support of context comparisons is limited as well. Means of representing user characteristics within the model and high-level abstractions of

context do not exist. In addition, the representation of uncertain context information is not explicitly addressed, although quality information can be captured by additional attributes, if required.

End-users gain insights into the application state through the externalisation mechanisms provided by the enactor concept and the mediator approach involves them in the ambiguity resolution process. However, the end-users are not able to actively modify the system's behaviour. The primary goal of the Context Toolkit consists in programming support and particularly in equipping a context-aware application with a flexible mechanism of acquiring contextual information. Any design support for end-users or developers at different expertise levels is not available. While the Context Toolkit addresses many requirements related to context gathering and supply, many responsibilities regarding reconfiguration and extensibility are incumbent upon the application and the developers. In addition, the enactor approach leads to cleaner code than an unstructured approach, but still results in applications that are difficult to maintain as source code must be modified in order to support additional classes of behaviour and context.

3.2.2 The PACE Middleware

The Pervasive, Autonomic, Context-aware Environment (PACE) project pursues a holistic approach towards a context management infrastructure for pervasive computing environments at the Distributed Systems Technology Centre of the University of Queensland (Henricksen et al., 2005). The infrastructure aims at facilitating the development of context-aware applications through the provision of generally required programming functionality like context gathering, context management, and context dissemination (Henricksen and Indulska, 2006). 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 relationships among context data and temporal aspects (Henricksen, 2003a).

Henricksen et al. (2002) base their context modelling concept on the Object-Role Modelling (ORM) approach and on extensions to it, which they call the Context Modelling Language (CML). Fact types are the basic building blocks for context modelling, which the researchers categorise according to their persistence and source: They can either be static or dynamic and dynamic fact types are again subdivided into the categories sensed, derived, and profiled types. In addition, special temporal fact types outline temporal characteristics of context and captures historical context information by representing start times and end times of facts. Furthermore, the Context Modelling Language takes quality measures of each fact type into account like accuracy, confidence, freshness, resolution or credibility. Based on these measures, consumers of context information can decide on whether to rely on it or not. The CML provides a modelling construct for alternatives in order to address sensors that supply multiple, conflicting pieces of information that cannot be resolved ambiguously. Such alternative facts are consolidated in one set. Fact dependencies are represented by a special type of relationship between facts in a similar manner to derived fact types. Physical or

conceptual objects such as persons or devices are represented by entity types. Entity types play certain roles in fact which can be regarded as associations between entities. A mapping process transforms the context model into a schema of a relational database and makes it processible by the infrastructure. .

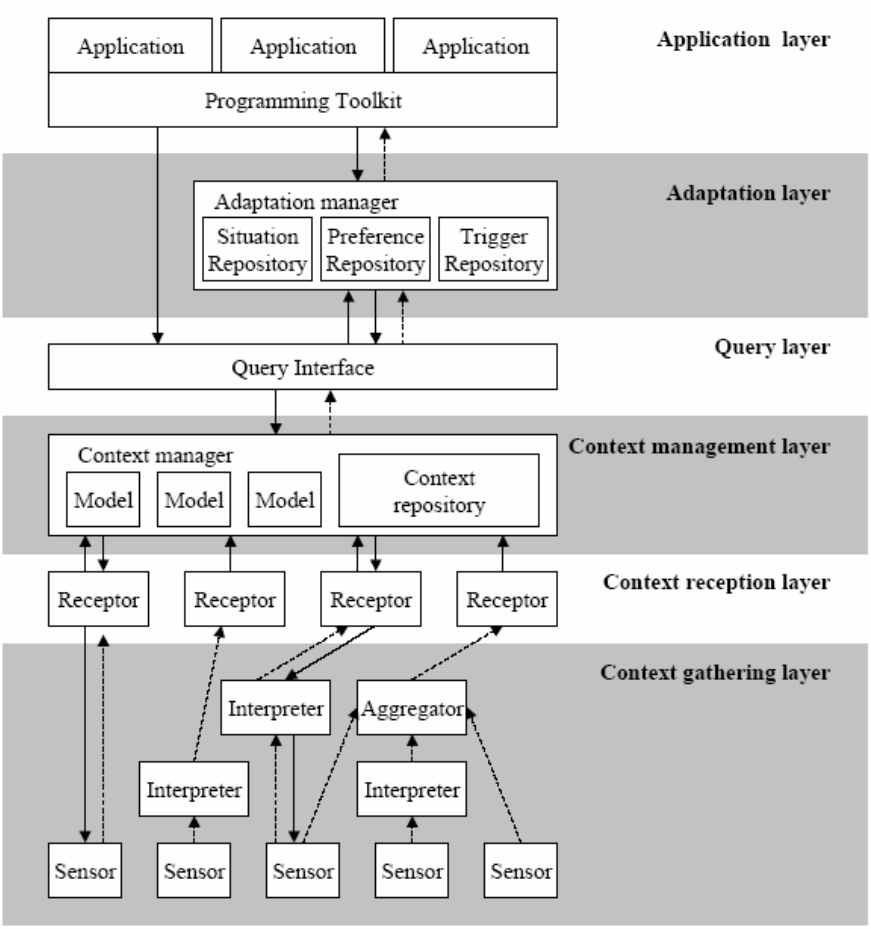


Figure 10 Architecture of the Context Management Infrastructure (Henricksen, 2003a)

Henricksen (2003a) elaborates high-level programming abstractions that ease the development of context-aware applications. The situation abstraction is used to specify context classes by placing constraints on relevant parameters. A situation holds in a given context exactly when a situation predicate is satisfied. Based on the detection of a specific situation, two programming models allow for the execution of actions under predefined conditions:

- *Triggering Model* as an event-condition-action model, where trigger events are activated in response to relevant context changes
- *Branching Model* allowing developers to incorporate context-aware behaviour into their application without the need for complex decision logic

The preference abstraction enables users to express favoured choices in the application behaviour that should be influenced by the context. Hereby, each preference assigns to each candidate choice a score that is determined according to the context.

Figure 10 illustrates the context management infrastructure which follows a layered approach for enabling a dynamic reconfiguration and separation of concerns. The Context Gathering Layer resembles the hierarchically ordered sensor network of Dey's Context Toolkit (Dey et al., 2001). The Context Reception Layer translates processed sensor data into the fact-based model, which is maintained by the Context Management Layer. This layer acts as a repository of context information that serves many context-aware applications. It maintains context information and responds to queries. The Query Layer provides an interface that allows applications to formulate queries on a context model using the fact and situation abstraction. The definition repositories for the programming abstractions are maintained by the Adaptation Layer. This layer provides facilities for the generation, retrieval and evaluation of these definitions. Finally, the Application Layer provides a programming support that application developers can use to exploit the functions of the architecture within their programmes.

The introduced architecture provides an integrated and holistic view of context-aware computing. Even though the process and flow of information between the layers is not formalised and incomplete, the tendency of an exceeding development support going beyond the acquisition and supply of context information can clearly be seen. However, means of adaptation are not discussed and it remains unclear how integration with an existing inadaptable system is achieved. The realisation of the architecture is a lightweight implementation: query and adaptation layer merged into application layer toolkit, context gathering and reception are not implemented, thus omitting the entire live context experience as they do not apply any sensors.

The main objectives are the proposed ORM-based context model and compatible programming models. The context model contains some simplifications that make a comprehensive representation of complex contextual information difficult. Furthermore, the context model is hard to exchange and incompatible to inference algorithms for example. Relations between entities have been taken into account, but no answer is given on how the attributes of the entities and their values are modelled. Employing this kind of context model results in a context data filtering, which is very bound to same elements of the context, and in derivation processes, which are restricted to database views, and thus, not sufficiently powerful. Uncertain, unknown and ambiguous context information is addressed, but it is not automatically treated. The low-quality information is passed to the context consumer that has to decide on the reliability of the information. Context management functionality for resolving such low-quality context information is omitted (e.g. switch on or off certain sensors).

End-user needs are met through the specific preference model. The language used to specify user preferences spares user friendliness and requires expert knowledge. To date, there exists

no support for identifying, evaluating and fine-tuning of preferences. The lack of reconfigurability eliminates the ability of the end-user to change the adaptive behaviour of the system.

3.2.3 The Coordinated Adaptation Platform

The main focus of the Coordinated Adaptation Platform (Efstratiou, 2004) developed at Lancaster University lies on providing support for the coordination of multiple adaptations across several applications. In order to enable the system-wide optimisation of multiple and possibly conflicting adaptations to context at application level, applications delegate adaptation control to a control component. This component receives state information from multiple applications and triggers adaptation in multiple applications. Thus, adaptation control and adaptation action (adaptation mechanism) are decoupled.

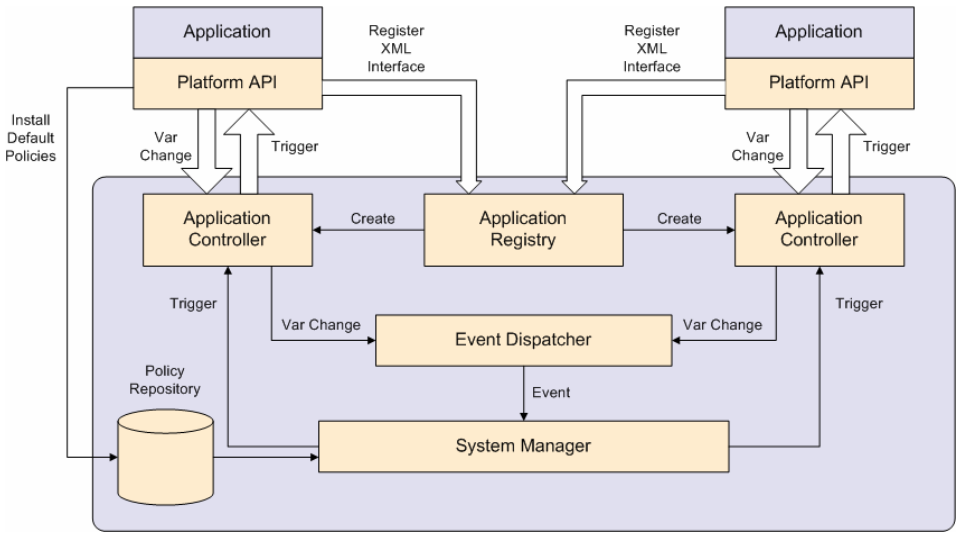


Figure 11 Architecture of the Coordinated Adaptation Platform (Efstratiou, 2004)

The platform utilises a policy-based mechanism of controlling adaptation, which is derived from Kowalsky's Event Calculus logic programming formalism (Kowalsky, 1986). The policy language allows the specification of policy rules where conditions are defined through the expression of temporal relationships between events and entities representing duration. User involvement is achieved by allowing the user to access the policy repository where adaptation policies are installed. In addition, the externalisation of the application state make users aware of and involve them in the adaptation processes (Efstratiou et al., 2001).

The architecture of the Coordinated Adaptation Platform (Efstratiou, 2004) consists of a set of components that are bundled into a single application providing system support (cf. Figure 11):

- *Application Registry* handles all communication between the platform and individual applications

- *Application Controllers* are responsible for the communication between the platform and one specific application running in the system
- *Event Dispatcher* implements the internal communication layer for the adaptation platform
- *System Manager* decides whether adaptation reactions are required by specific applications

The Application Registry is the first contact point for every application that uses the system. Applications are required to connect to the Application Registry and submit a registration document that describes their adaptive interfaces (XML registration document). Each registered application obtains one Application Controller that parses this description and creates a table of adaptation methods and a table of state variables exported by the application. Listening to any changes in these application state variables, the Event Dispatcher propagates the changes to the appropriate policy rules maintained by the System Manager in a first-come first-serve order. The System Manager evaluates the respective policy and decides whether or not to invoke specific actuators that lead to the execution of commands within the application, and thus, to system changes. 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.

This approach is the first to explicitly address adaptation methods at the application level. Focussing on the coordination of adaptation methods, the approach lacks a clear representation of these methods. It remains unclear whether a specific action can be executed with a particular characteristic, which implies parameterised actions. In addition, no information is provided about how more complex actions or adaptation strategies can be realised, e.g. by nesting of atomic actions in order to accomplish a more comprehensive adaptation of an application.

Even though the proposed approach achieves a certain level of transparency and user control through the externalisation of the application state and adaptive mechanisms, it is questionable if the level of complexity is appropriate for end-users. The user needs to able to understand both the way applications work in the system and how their behaviour can be modified through the utilisation of the Event Calculus Policy Language. For most users the researchers accept that this will be a specialist skill (e.g. the role of an administrator)

Targeted on meeting the key requirements onto adaptable context-aware applications, this approach addresses issues of context gathering and modelling to a small extent only. Any attempts towards acquiring or modelling user characteristics, and supporting users in performing changes to the entire behaviour of the system are abandoned. The understanding of context is tightly coupled with states of applications running on a computer system. For this reason, the context gathering process is restricted to obtaining state values of applications. A formal context model has not been developed so far and context representation rather

comprises tags contained in the XML descriptions of state variables and adaptation mechanisms.

3.2.4 The Context Service Project

The Context Service project aims at gathering, maintaining and supplying context information to clients (Ebling et al., 2001). The project is formally known as Owl and displays the results of the IBM T. J. Watson Research Center. The project pursues a service-based approach to context awareness that provides applications with contextual information at a high level of abstraction. In the Context Service context is modelled using a form metaphor (Lei et al., 2002), in which a form consists of three elements:

- one type of context information
- a collection of related context attributes
- metadata on quality of information

Currently, the quality metrics cover a timestamp labelling the freshness and the confidence of the source. Context forms provide clients with a flexible specification of the selection condition for context, thus making queries to the Context Service possible. The set of form types is extensible, as new context sources can be integrated, and provides a uniform abstraction for taking data heterogeneity into account. The programming model allows for both synchronous queries and asynchronous event notifications.

The architecture of the Context Service shown in Figure 12 provides a dispatcher that forwards client requests to a configurable collection of suitable context drivers via the Context Driver Interface. Context drivers are components that can be plugged into the infrastructure and handle a particular type of contextual information (encapsulation of context sensors). They may either pull data from sensors or receive updated information from it using the Context Push Interface. Four utility components can be used by the context drivers:

- *Context cache* retains recently access context information
- *Work pacer* schedules pulling of information to avoid overloading of sources
- *Event engine* matches context events with registered application requests
- *Privacy engine* controls access to context information based on policies

Furthermore, context drivers may obtain context from other drivers via the client application programming interface, which allows 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 (Lei et al., 2002). Until now it remains unclear whether and how they are to be integrated into the Context Service architecture.

In contrast to many other approaches, the Context Service project explicitly addresses privacy and security issues. The included privacy engine is responsible for controlling access to contextual information. An object model of policies specifies whether or not a proxy application acting on behalf of a requestor is granted access to information about a particular subject. The privacy engine specifies, stores, and retrieves privacy policies, and employs the privacy protection mechanism, which relies on role-based control.

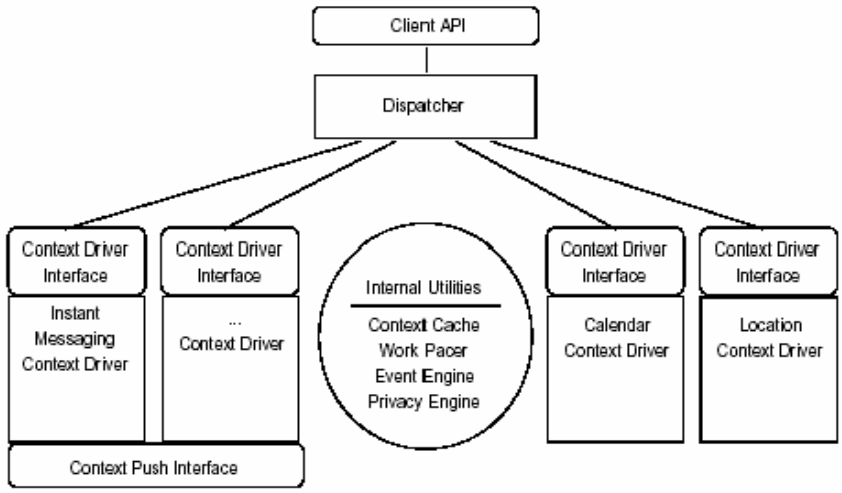


Figure 12 Architecture of the Context Service (Lei et al., 2002)

Lei et al. (2002) describe two example applications that build on top of the Context Service. The Notification Dispatcher uses context for instant messaging the on-line status and calendar events. Based on her current context and on the urgency of messages, the dispatcher routs messages to one of the recipient's communication devices that are considered most appropriate. The Pervasive Content Distribution System (PCD) forecasts users' access to web content taking predicted context into account. The PCD pre-processes and pre-distributes content in order to reduce access latency, driven by a prediction of the user's location and task.

The architecture of the Context Service disregards derivation of context information and context comparisons for filtering purposes. The augmentation of the acquired context information is addressed only marginally. In exchange, an array of quality metrics is covered and provided to the clients. However, the architecture does not exploit quality metrics for automated context management issues. Unlike the previous solution, the Context Service explicitly addresses privacy in a way that is comparable to the Context Fabric (Confab) proposed by Hong and Landay (2001), which is primarily concerned with privacy rather than with context sensing and processing. However, the privacy model assumes a closed and secured system, in which the identity of all subjects is known by the system.

The proposed form abstraction for representing context does not distinguish between different attributes of the context, and thus, decreases flexibility and extensibility. Since the available

sensors gear the design of the forms, a formal context model is not given. For this reason, the representations of user characteristics or even user modelling techniques are not considered. Furthermore, aspects and concepts of involving the end-user in the design process of context-aware applications are not addressed. Although the Context Service tackles several advanced issues related to context awareness, the concepts proposed so far lack maturity in some aspects.

3.2.5 The Context Broker Architecture

The University of Maryland elaborated the Context Broker Architecture (CoBrA), which is an agent-based architecture for supporting context-aware applications in smart spaces, i.e. open and dynamic environments, e.g. intelligent meeting rooms, smart homes, and smart vehicles (Chen, 2004). The main aim of this architecture is to acquire contextual information from sources that are unreachable by resource-limited devices and approach context dissemination as a knowledge sharing problem. Central to this architecture is an intelligent agent called context broker that maintains a shared model of context on the behalf of a community of agents, services, and devices in the space, and provides privacy protections for the users in the space by enforcing the policy rules that they define.

The Context Broker Architecture differs from other similar architectures in exploiting the Web Ontology Language OWL (Smith et al., 2004), a W3C Semantic Web standard, to define ontologies of context. In addition, this language is used to express semantics of contextual information. In a smart space environment populated with multiple Context Brokers each broker is responsible to maintain parts of the space's context. Since different brokers may possess distinctive context knowledge, agents can subscribe to the Context Brokers and acquire different context knowledge and fuse them to form a coherent view of the smart space context.

The Context Broker is also responsible for contextual reasoning. Chen (2004) distinguishes two types of inference procedures:

- aggregating and interpreting the data acquired from the physical sensors and
- detecting and resolving inconsistent knowledge in the context model

In order to reason about contextual information that cannot be acquired by sensors, a rule-based logical inference approach is implemented using context interpretation rules for aggregation and interpretation. For maintaining a consistent model of context the OWL-based ontology reasoning is used to detect knowledge inconsistency and the assumption-based reasoning is applied to resolve this inconsistency.

The Context Broker employs declarative policies for privacy protection by extending the Rei policy language (Kagal et al., 2003). In this approach, users define policies to control the sharing of their private information and send these policies to the Context Broker as they enter a smart space. Before the broker shares a user's information with other agents, it consults the user's policy to check whether or not the communicating agents are permitted to acquire this information. When defining a policy, the user specifies the actions that represent the sharing of their private information. For each action, the user declares the prerequisites of the agents that are permitted or forbidden to perform the action. Users can control the granularity of the shared information, as well, and the context broker will attempt to adjust the information granularity by finding a more general concept in the ontology.

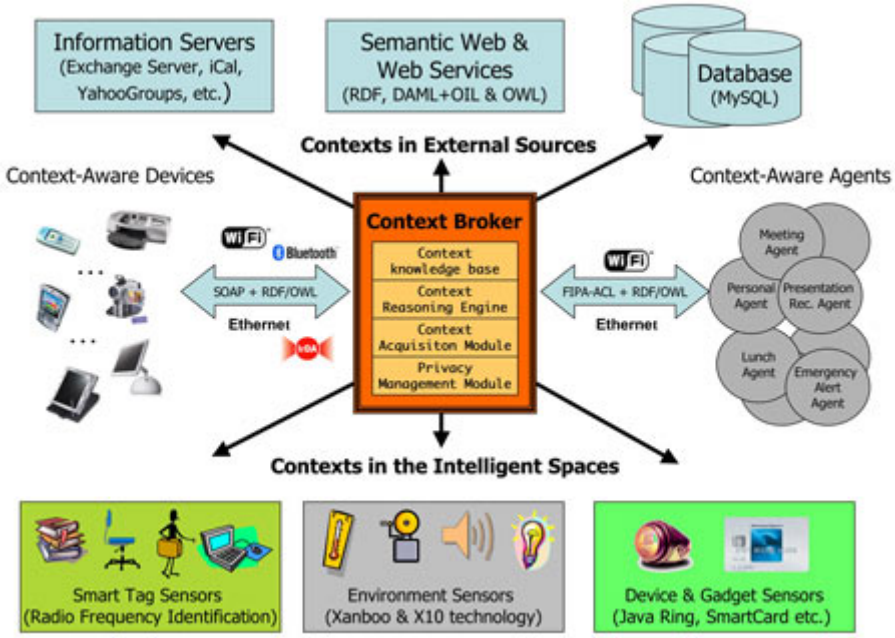


Figure 13 The Context Broker Architecture (Chen et al., 2004b)

The Context Broker is a specialised server entity that runs on a resource-rich stationary computer. Figure 13 illustrates the functional design of the broker and its four components:

- The Context Knowledge Base manages the storage of Ontologies and their instances
- The Context-Reasoning Engine reasons over the acquired contextual information
- The Context-Acquisition Module offers a set of acquisition procedures for context information from sensors, agents and the Web
- The Privacy-Management Module maintains and enforces the users' privacy policies, and controls the sharing of private information

A prototype implementation of the Context Broker Architecture serves for the realisation of EasyMeeting, a context-aware meeting room (Chen et al., 2004b). In EasyMeeting, the context broker acquires contextual information from the sensors in the room and shares that

information with various meeting services. Using this information, the services are able to compose and play personalised greeting messages as people enter the meeting room, help speakers to set up their presentations, and show the audiences the profiles of individual speakers while they are giving their presentation.

The Context Broker provides advanced mechanisms of the acquisition and sharing of context information among smart space devices. The acquisition methods extend sensors by other agents and the Web, but elide explicit input provided by users. The major difference to similar architectures lies in the provision of a formal context model expressed by an ontology language. However, only components that understand this Standard Ontology for the Ubiquitous and Pervasive Application (SOUPA) may benefit from the features of the architecture, which limits the reusability of parts of the architecture.

The ontology enables the Context Broker to provide a coherent view of the context and to perform the context management tasks detecting and resolving inconsistent knowledge in the context model. Additionally, aggregation and interpretation is supported, but limited to a rule-based approach. Furthermore, context reasoning rules often require the special knowledge of domain experts and thereby limit reusability across different domains. User modelling methodologies and learning user needs are not regarded. The privacy protection approach of adjusting the information granularity depending on declarative policies is straightforward taking ontologies into account, but ignores different user privacy preferences as only one global privacy policy exists.

The services described in the EasyMeeting application illustrate the applicability of the Context Broker. However, a description of how the adaptive behaviour of these services can be tailored is nonexistent. The end-users are not involved in the design and development process of context-aware applications and developers are not supported by programming abstractions.

3.2.6 The Technology for Enabling Awareness Project

The main objective of the Technology for Enabling Awareness (TEA) project and its successors, funded by the European Commission and conducted by the Starlab research institute (Brussels, Belgium), constitutes in the augmentation of mobile devices and everyday environments with context-awareness. Schmidt et al. (Schmidt, 2002; Schmidt and van Learhoven, 2001) describe a layered architecture which they employ to synthesise context information from a heterogeneous set of sensors. The project is targeted on resource constraint platforms and primarily 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 autonomous “smart objects”, such as next generation mobile phones. Thereby creating an application-independent add-on component to existing devices and artefacts is accomplished. Due to the integration of diverse sensors and the consideration of abstract context data such as people’s activities, issues concerning the aggregation of sensor data and the derivation of context are explicitly addressed.

The architecture proposed by the TEA project consists of four functional layers as shown in Figure 14. The lowest layer is responsible for the gathering of raw data from a set of heterogeneous sensors. The cue layer on top of the sensor layer consists of several cues, each of which abstracts from the output of a single sensor and represents the feature extracted from the sensor's data stream. Since the cue layer provides a uniform interface to upper layers, it hides the details of the context sensors employed from them. The third context layer aggregates data from several sensors and transforms this data into context, which is considered as a function of the available cues. Finally, the top scripting layer makes use of context by accordingly adapting the behaviour of the application or device.

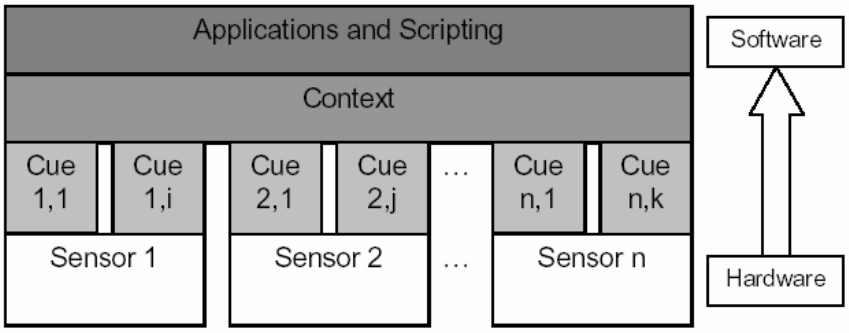


Figure 14 The Technology for Enabling Awareness Architecture (Schmidt, 2002)

The architecture is structured to support a two-phase interpretation process: The first phase maps the output from each sensor into a variety of cues. These are features derived from a set of sensor readings such as the average and standard deviation. The second step combines the cues to form an abstract context description using rule-based algorithms, statistical methods, neural networks (van Learhoven et al., 2001), or other interpretation techniques. The scripting layer provides mechanisms to include context information into the application by considering three states: entering a context, leaving a context and being in a context.

During the course of the TEA project two hardware prototypes have been developed, which are equipped with several sensors including accelerometers, light sensors, microphones, CO sensors, pressure and temperature sensors. In addition, an application called the Context-Call system (Schmidt et al., 2000) has been implemented, which demonstrates the usage of context in the area of telephony. Later research resulted in the development of the Media-Cup (Beigl et al., 2001), a coffee mug augmented with hardware and software for context gathering, processing, and dissemination, and of Smart-Its, small embedded devices that are augmented in a similar way (Holmquist et al., 2001).

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. Schmidt et al. (1999) describe a working model of context, which consists of an enumeration of hierarchically organised 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 (2000) furthermore provides a specification language for context based on Standard Generalized Markup Language (SGML) (Brown, 1996). The context description represents context attributes and their assigned values, both referred to as contextual variables, which can be grouped with matching attributes. The uncertain nature of context information is reflected within the context model through associating a probability estimate with each value and a vector of such value-probability pairs describes the environment. Scripting primitives accompany the model and allow for an automated execution of actions when a context event is detected with a predefined probability.

The TEA project addresses so-called direct context (Gellersen et al., 2002), i.e. context that is gathered and processed by mobile devices themselves rather than by a specialised infrastructure. Its focus lies on making context information accessible within a ubiquitous computing environment, and therefore, entirely ignores involving end-users in the development and customisation process of context-aware applications. The proposed context model allows for the representation of user characteristics, but the project lacks support for the acquisition of this information. In addition, context model lacks formality, transparency and universal applicability. Furthermore, this model 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 only reflected by an annotation of contexts with an indication of their probability. Unknown and ambiguous context information cannot be adequately characterised. The restriction of the infrastructure to hardware with low computation power consequently limits the provision of higher level support for developers like programming abstractions or tools for the design of context-aware applications.

3.2.7 The Context Fusion Network

A research group at Dartmouth College has designed and developed the Context Fusion Network, called SOLAR, a software infrastructure for context acquisition (Chen and Kotz, 2002a; Chen and Kotz, 2002b). Chen et al. (2004a) propose the use of Context Fusion Networks in order to provide data fusion services (aggregation and interpretation of sensor data) to context-aware applications. The approach also addresses issues of scalability as well as of security of contextual information. The software infrastructure bases on a graph abstraction of the specification of connections between components. The nodes of this directed acyclic graph are subdivided into three types:

- *Sources*, i.e. wrappers for context sensors and generators of event streams
- *Operators*, i.e. objects that receive, process and forward event streams
- *Sinks*, i.e. consumers of event streams

The system represents contextual information as events that are structured as a set of tag-value pairs. 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. As new operators can be dynamically inserted as required, the modular structure of operators allows for a flexible operator composition as well as for distribution and reuse.

The implementation of the Context Fusion Network model displays the form of a scalable peer-to-peer platform. Applications produce textual specifications of their context requirements in the form of graphs. SOLAR uses these descriptions to create the required operators and event subscriptions and instantiates the operator graphs at runtime on behalf of context-aware applications. Furthermore, this language is employed for the supply and discovery of resources in SOLAR's context-sensitive resource discovery mechanism.

The SOLAR system interconnects several hosts (so-called *Planets*) within a distributed network. The SOLAR hosts may run on separate nodes and support application and sensor mobility by buffering events during periods of disconnection. Planets address component failures by providing monitoring and recovery as well as preservation of component states. They serve as an execution platform for SOLAR objects such as operators or proxies. Any client may connect and submit requests to any Planet. The Planets parse these requests, create appropriate operator instances according to the specification received from the client, and maintains a list of subscribers, as well as a queue of input events.

The researchers at the Dartmouth College developed the SmartReminder application on the basis of SOLAR (Mathias, 2001). This application uses context to determine the most appropriate time to remind users of appointments. The contextual information considered is the current location of a user, the location an appointment takes place at, and the estimated time the user needs to arrive at this location.

The SOLAR system addresses many requirements pertinent to ubiquitous computing. The strong context acquisition approach allows a flexible and extensible combination of augmentation processes. However, this approach does not support any acquisition of explicit user input and omits modelling characteristics of users. A major disadvantage is its relatively simple model of context. Furthermore, the model lacks formality and the explicit support for quality measures. In addition, the components do not maintain a persistent history of context information per default.

The context acquisition and the supply of context information are not separated by the SOLAR system. Applications are required to concern themselves with the context acquisition process because they need to know about the availability of sensors and operators. They are also responsible for the definition of the operator graph. Context Management issues like the provision of monitoring and recovery in case of component failures are addressed as well as the preservation of component states. End-user involvement by controlling the application's behaviour remains unsupported and the design support for building context-aware applications is weak.

3.2.8 Other Related Approaches

This subsection presents projects and approaches that depict a certain relation to the work and research conducted by this thesis. These projects only cover certain sub-aspects of the creation process of context-aware application and therefore show some general drawbacks in this regard. Furthermore, they restrict the group of actors involved in this creation process to a minority. The following paragraphs describe projects targeting the middleware-based creation of context-aware applications, tools for the design of such applications and contextualised information delivery.

Middleware-Based Creation of Context-Aware Applications

The middleware-based realisation of context-aware applications primarily focuses on software developers as the main actor in the software engineering process. The middleware-based approach comprises comprehensive software libraries and employs methods of encapsulation for the separation of business logic and functionality of context-aware computing. A context middleware introduces a layered architecture of context-aware applications with the intention of hiding low-level functionality for the acquisition, transformation, dissemination of context information. In addition, such a middleware provides its functionality via a standard application programming interface, monitors the context changes and sends events to interested applications.

Middleware-based approaches ease extensibility and simplify the reusability of software components. However, these approaches provide weak support of programming abstractions like the determination of situations or the access to historical context information. Furthermore, a context middleware mostly focuses on the acquisition of context information, and thus, disregard the input of users and insufficiently address control and actuation mechanisms of the targeted context-aware application. Hence, middleware-based approaches omit covering the end-to-end process chain of context-aware application ranging from the acquisition of context information to the realization of adaptations at multiple levels of abstraction. Furthermore, a context middleware places little emphasis on the involvement of other actors than developers in the development process of context-aware application and fail to support end-users in the adaptation of the behaviour or authors in the design of the appearance of the targeted application.

Due to these general restrictions and drawbacks, the middleware-based approach to the creation of context-aware application does not meet the requirements of this thesis. However, they are related to the research of this thesis, and this subsection enumerates a list of selected approaches.

The Contextor Infrastructure for Context-Aware Computing

The Human-Computer Interaction Group as a part of the CLIPS laboratory of the IMAG institute located in Grenoble (France) developed the Contextor Infrastructure for Context-Aware Computing (Rey and Coutaz, 2004a; Rey and Coutaz, 2004b). This infrastructure is a

result of top-down design approach informed by the theoretical foundations of a simple ontology and an abstract computational process model.

A Contextor constitutes a software abstraction that models a relation between several observables or variables of context information. A Contextor comprises a functional core, implementing the function of the Contextor and delivering results, and typed communication channels, transferring values between Contextors and controlling the Contextor behaviour. Depending on the type of functional core and communication channel, different types of Contextors emerge: elementary, history, threshold, translation, fusion, and abstraction Contextor. Furthermore, Contextors may be composed in two ways to form federations of Contextors: through the connection of data channels or encapsulation.

The assembly of the Contextor software components is ruled by an architecture style providing four layers of abstraction. The sensing layer encapsulates a diversity of sensors. The transformation layer determines the meaning of the values provided by the sensors. The situation and context identification layer constitutes the reasoning and inference layer, which detects conditions for moving between situations. The exploitation layer acts as an adaptor between the targeted application and the infrastructure. This layer responds to requests from the targeted application and has direct access to all three layers. Each level of abstraction provides mechanisms to support privacy, trust, and security as well as history management, and discovery and recovery. The implementation of the Contextor infrastructure for context-aware computing is restricted to the sensing and interpretation layers, and therefore, it fails to fine-tune and reconcile the conceptual principles of this work through practical considerations.

The Java Context Awareness Framework

The Java Context Awareness Framework (JCAF) elaborated by the Centre of Pervasive Healthcare at the University of Aarhus in Denmark provides a Java-based service-oriented runtime infrastructure and programming API for the creation of context-aware applications (Bardram, 2005b; Bardram, 2005c). Following the Context Toolkit (see Section 3.2.1), The JCAF represents only the second toolkit implemented in Java that aims at facilitating the experimental prototyping of context-aware applications for developers.

The runtime infrastructure emphasises security and privacy in an environment of distributed and cooperating services acquiring context information (through Context Monitors and Context Actuators) and enables interested application components to subscribe to relevant context events through an event-based mechanism (Bardram, 2005a). Context Transformers reside in a transformer repository and constitute application-specific implementations of aggregators of context information.

The JCAF application programming interface offers a semantic-free abstraction of modelling context based on entities that possess a set of context items covering relevant context information. Entities can themselves be context items of other entities, and this concept creates an elementary means for establishing relations among entities, which are difficult to

exploit and maintain. The JCAF has been the basis of three different context-aware applications providing a source of lessons learnt from using the JCAF. However, all applications of JCAF basically expose location-awareness.

The CORTEX Project

The COoperating Real-time senTient objects: architecture and EXperimental evaluation (CORTEX) project (Sørensen et al., 2004) supported by the Future and Emerging Technologies programme of the Commission of the European Union elaborated on the use of sentient objects to construct large-scale context-aware applications in an ad-hoc mobile environment. The term "Sentient Computing" was coined at the ORL research laboratories and AT&T Laboratories in Cambridge (AT&T, 2001; Hopper, 1999) and constitutes a conceptual framework, which formed the basis of many different projects rather than a single large project. This approach aims at transferring the human's perception of their environment to computer systems (Addlesee et al., 2001).

The middleware bases on sentient objects that constitute encapsulated entities consisting of three main parts: sensory capture, context hierarchy and inference engine. In the sentient object model, sensors are defined as entities that produce software events in reaction to a real world stimulus, whilst actuators are defined as entities which consume software events (Biegel and Cahill, 2004). The actuation is controlled based on sensor input according to internal control logic, consisting of event filtering, sensor fusion and intelligent inference. Sentient objects dynamically discover each other and share context information. The sentient object model incorporates the STEAM (Scalable Timed Events And Mobility) event service (Meier and Cahill, 2003) to provide communication among the sentient objects of the model. This event-based communication mechanism enables sentient objects to sense and view the behaviour of neighbouring objects, reason about them, and manipulate physical objects accordingly.

In order to support sentient objects, CORTEX provides a middleware based on a component framework, each of which provides a service to the sentient objects. This framework defines software abstractions for sensors and actuators, and allows the specification of a rule-driven behaviour. The resulting middleware is configured at deployment time and can be reconfigured at run-time through a reflective application programming interface to adapt to changes in the environment. Furthermore, a graphical development tool allows developers to specify relevant sensors and actuators, define fusion networks, specify context hierarchies and rules, without the need to write any code.

The ParcTab System

The ParcTab project conducted at Xerox PARC (Schilit, 1995) represents the first attempt to realise the Weiser's vision of ubiquitous computing (Weiser, 1991). The resulting ParcTab system bases on an architecture that supports mobile computing and addresses networking, device technology and human-computer interaction. The ParcTab system provides general purpose configurable mechanisms describing the utilisation of context and allowing the users

to tailor the behaviour of the system to their own needs (Schilit et al., 1993). The system is designed for indoor operation, in which users carry hand-held devices that use infrared as a communication and location tracking mechanism (Want et al., 1992).

One of the core characteristics of the ParcTab system consists in the sharing of contextual information among participants and offers four basic mechanisms of building context-aware applications: proximate selection, automatic contextual reconfiguration, contextual information and commands, and context-triggered actions. These mechanisms allow the system to modify the information presented to the user or the execution of programmes according to the physical location of the user or the proximity of other users.

The ParcTab system formed the basis of many different applications making use of the environment. These aware applications can retrieve context information synchronously using remote procedure calls, or asynchronously by subscribing for notifications. However, even though the ParcTab system bases on a very flexible architecture, all the mechanisms provided are coupled with location information. Therefore, the incorporation of new types of context information would require a re-implementation of the system. In addition, the simplicity of the context model used by this architecture makes it unsuitable for capturing historical context information.

Design Tools for Context-Aware Applications

The middleware approach of creating context-aware applications eases the burden on developers of context-aware applications but neglect opening up the space for designers and end-users of such applications. Since developers of context-aware applications design the context-aware behaviour of their applications usually with little user consultation, this behaviour is often hidden from users or difficult to override, which causes usability problems.

The paradigm of End-User Development (Fischer, 2002; Lieberman et al., 2006) addresses the active participation of end-users in the software development process. This paradigm transfers the tasks that are traditionally performed by professional software developers to the end-users in order to achieve a close match between user requirements and system behaviour. The active participation of end-users throughout an entire software development life-cycle ranges from the provision of information about the requirements, use cases and tasks, including participatory design, towards end-user programming. The projects covered by this subsection focus on end-user programming and empower end-users to configure and compose the context-aware application according to their diverse and changing needs. The Stick-e Notes framework covers various types of modifications an end-user may perform to a context-aware application in order to achieve a certain degree of end-user involvement in the realisation process. In order to enable end-users to handle the complexity they are confronted with when adapting and configuring context-aware applications, the iCAP and a CAPpella systems concentrate on developing appropriate and intuitive visual programming support.

End-user programming techniques generally provide better user control than traditional software engineering techniques, but in the decision to only support end-users they limit their

utility for developers. Such programming techniques require the flexibilisation of the software components and the underlying infrastructure, which leads to abstraction. All approaches presented in the following represent the behaviour of the targeted context-aware application with a rule-based or learnt connection between situations and actions. Furthermore, they are currently restricted to sensed context information and do not provide any mechanism to handle historical context information. Because of these restrictions and abstractions, the presented approaches can only handle and create context-aware applications that exhibit a low level of complexity and a straightforward behaviour. Furthermore, the end-user programming approaches address only a subset of the design tasks associated with context-aware computing. However, the realisation of context-aware applications for a variety of domains of different complexity requires the assistance of all aspects of a more comprehensive design process of context-aware applications. The following paragraphs provide a more detailed description of these projects.

The Stick-e Note Framework

The Computing Laboratory of the University of Kent at Canterbury developed the Stick-e Note framework (Brown, 1996) as a result of their research in mobile computing applications. Their work focuses on handheld mobile devices, which are used for information collection and information retrieval. The Stick-e Note framework aims at the facilitated creation of discrete context-aware applications for designers and authors (Brown, 1998). It provides general mechanisms for indicating what context an application designer wants to exploit and provides simple semantics for writing rules that specify what actions to execute if the current context information fulfils particular requirements.

The Stick-e Note framework comprises a model for the association of content with context information and a triggering engine. In order to determine the application behaviour, the author utilises a specific markup language to write individual rules in the form of Stick-e Notes. A Stick-e Note consists of a note, which characterises a context description, and a body, which contains information or a script. If the context described in the note matches the readings from sensors in the real world, the triggering engine presents the information or executes the script contained by the body. The whole targeted context-aware application consists of a set of Stick-e Notes bundled into a document and the Stick-e Note architecture.

The Stick-e Note framework has demonstrated its utility through various implemented context-aware applications that use mobile devices that are made aware of their location (Brown, 1996; Brown, 1998; Pascoe, 1997; Pascoe, 1998). The practical focus of most of these applications is on the location acquired by a GPS device. The Stick-e Note framework does not cover continuous context-aware applications and cannot handle rapidly changing discrete context information. In addition, the context information is not provided to other interested applications. Furthermore, the framework ignores the storing of historical context information and the interpretation of context information to gain semantically enriched information.

The Systems iCAP and a CAPpella

The joint research of the University of Berkeley, Intel Research in Berkeley and the College of Computing at Georgia Tech in Atlanta focussed prototyping environments for end-users to build context-aware applications without writing any code. The iCAP system (Sohn and Dey, 2003) provides end-users with a design tool that allows for the specification of a rule-based behaviour for the targeted context-aware application in a graphical user interface. The rules represent relationships between acquired context information and performed actions. This rule-based approach limits the design space of context-aware applications to such types whose context-aware behaviour can be determined by static and well-specified rules.

In the continuation of this research, the a CAPpella system represents a more physically-based prototyping environment for context-aware applications, which allows end-users to create recognition-based context-aware applications (Dey et al., 2004). A Cappella combines machine learning and user input in order to support the building of context-aware applications through end-user programming by demonstration or example (Lieberman, 2001). The machine learning techniques employed by the system learn from the user's demonstration of situations and associated actions. Thus, the end-user programmes the behaviour of the system by acting the rules or behaving naturally. The system requires a number of training examples in order to produce activity recognizers. Once the a CAPpella system is trained, it recognises a demonstrated situation and is able to carry out the associated actions automatically, without prompting the user. The drawback of this approach constitutes the alternation of a behaviour that the machine learning algorithm has already learnt, in case of a changing environment or activity of the user. Even minor changes in the behaviour could force the end-user to retrain the algorithm. Furthermore, this tool remains in an early prototype stage and no evaluation of exploiting this approach for design of a large-scale context-aware application has been conducted so far.

Contextualised Information Brokering

Information brokering addresses the directed collection, reorganisation and customer-oriented distribution of information and describes the value-adding process of mediation between information demands and information offers (Klemke, 2002). The consideration of the particular contexts of the information provider, information consumer and the information entity itself can additionally enhance this mediation process. The resulting type of context-aware application relies on the association of (relatively static) context information with information entities of the domain, because such an association enables a filtering of entities based on the similarity of the context of each entity and the context of the user (Oppermann and Specht, 1999).

The delivery of information to the user depending on her current context constitutes a major area of context-aware computing. Context-aware applications that perform contextualised information delivery cover domains such as museums, e-learning, or tourism (see Section 4.1). The context model of an appropriate context-aware application exhibits a strong

connection with the specific application domain. For authors or developers of such a context-aware application, the creation of an accordant knowledge base involves the allocation of context information to the set of domain entities. In order to facilitate this allocation process, many existing metadata standards determine the schemata of the incorporation of context information into the description of an entity such as the Learning Objects Metadata standard version 1.0 (LOMv1.0) of the e-learning domain (Cardinaels et al., 2006; Hodgins and Duval, 2002) and the Moving Picture Experts Group standard MPEG-21 of the multimedia domain (Bormans and Hill, 2002).

Some projects brokering information provide authoring tools for the generation of contextualized information. The following paragraphs present two projects from different research field that facilitate the creation of contextualised content. Because these projects represent a specific type of context-aware application, they are related to this thesis. However, these projects emphasise authors as the main actors in the development process of context-aware application and disregard other actors. In addition, they rather focus on situation-awareness, because the domain entities remain in a rather static context with marginally changing context information. Furthermore, these approaches rely on context information acquired from sensors and dispense with the derivation of higher-level context information and with the consideration of historical context information.

The Broker's Lounge

The Broker's Lounge (Jarke et al., 2001) developed at the Fraunhofer Institute for Applied Information Technology in Sankt Augustin (Germany) facilitates the development and management of information brokering services that deliver filtered information on demand. This tool allows a structuring and organisation of information into hierarchies of concepts and categories that form the ontology of the targeted domain. Concepts determine the type of information and categories describe properties of information types from the user's perspective. In contrast to categories, concepts can develop certain relations to other concepts of the concept hierarchy.

An ontology administration interface supports the information provider in annotating information entities with additional context information. In analogy to Semantic Web systems (see for example Berners-Lee et al. (2001)), this annotation bases on the reference of the information entities to the domain ontology. The combination of concepts and categories enables the multidimensional classification of information entities, which results in a description of properties of the respective information entity. A source administration user interface allows for the specification of access profiles for online sources of information entities and facilitates the configuration of aspects like access frequency or weighting. In addition, software robots scan documents of these online sources for occurrences of domain concepts (and possible synonyms) in order to gather further information about the information entities and calculate a domain score for each.

This semiautomatic annotation process of information entities delivers a description of the context information related to each information entity comprising its fundamental properties.

The context information associated with each information entity allows for an evaluation of its relevance for a given domain or situation, and affects the filtering triggered by a retrieval process. Based on the Broker's Lounge two major applications have been developed. The ELFI system constitutes an advisor that manages knowledge about research programs in Germany such that a proposer can identify appropriate funding schemes. The MarketMonitor represents a tool that helps companies monitor the web pages of competitors, suppliers, and customers for early detection of changes in the market situation.

The RAFT Project

The European project RAFT (Remotely Accessible Field Trips) creates learning tools for field trips in schools and aims at embedding of learning and teaching activities in an authentic real world context (RAFT, 2003). Besides the individual learner, her knowledge, interests, and preferences, the contextualized learning approach followed by the RAFT project takes into account additional context parameters in order to adapt the content selection, content presentation and navigation support (Specht and Kravcik, 2006). The principles of learning in context and understanding artefacts of the real world with having information available in context base on the underlying theoretical background of situated cognition and situated learning (Lave and Wenger, 1991).

The software development of the RAFT project produced the Mobile Collector (Kravcik et al., 2004), an authoring tool for creating and relaying contextualised learning materials. The Mobile Collector facilitates the collection of data (particularly photos) in the field, its storage in a repository and its annotation with additional information. The semiautomatic annotation process covers the manual selection of related concepts (keywords) from a list predefined by the teacher and recording of an audio comment, and the automatic addition of situation information acquired by sensors for time, location, temperature, etc. (Specht et al., 2006).

The Mobile Collector stores all the collected data in the system repository and allows for a later elaboration through the learner as well as an evaluation through the teacher. The additional context information enables the structuring, accessing, exploring and visualization of learning materials. Furthermore, this context information provides a flexible way for teachers to integrate these materials into their courses and for the learners to include them into their projects.

3.3 Key Requirements and Summary

This chapter outlined the research framework that is relevant for the provision of development support facilitating the construction of adaptable context-aware applications. In a first step, the research direction constrains the research area of context-aware computing to specific aspects of concern for this thesis and provides an assessment framework for the subsequent investigation of the state of the art. In a second step, this chapter surveyed the state of the art of tools, infrastructures and other development support facilitating the construction of

context-aware applications and involving other actors than developers in the software engineering process. This examination of related approaches has been guided by the research directions that substantiated the discussion and evaluation. The insights provided by these sections form the basis of the derivation and identification of key requirements that allow for a detailed assessment of the existing approaches. The following paragraphs present these key requirements and provide summary of the research framework.

3.3.1 Key Requirements

This section elaborates a set of key requirements that are essential for the realisation of appropriate development support for adaptable context-aware applications. Requirements are descriptions of how the system should function, constraints on the system's operation, or specifications of a system property or characteristic. The process of requirements engineering is a continuous iterative process and not a stage or phase in the way. Once identified and documented, the requirement's description tends to be subject to alternations because new requirements arise and old ones disappear or change. These key requirements guide the assessment of both the approaches surveyed in the state of the art as well as the results of this work. The key requirements mainly originate from the weighted condensation of the described research direction and comprise:

Holistic Context-Aware Application Architecture: The development support must represent a holistic, systematic, and application-oriented functional decomposition of context-aware applications. The fit criterion for this requirement is the provision of a software architecture and framework of context-aware applications that covers the end-to-end process chain ranging from the acquisition of context information to the realization of adaptations at multiple levels of abstraction.

Context- and User-Modelling Integration: The development support needs to provide architectural support for the construction of adaptive applications that exploit both information about the context and the user. This requirement is accomplished if the development support allows for the modelling and derivation of user characteristics, and the processing of information explicitly specified by the user.

End-User Involvement: The development support needs to enable end-users to modify and control the adaptive behaviour of their context-aware application through the integration of adaptivity and adaptability. This requirement is met if the development support covers transparent internals of the targeted context-aware application that are examinable by the end-user and if the behaviour of this application is adaptable.

Software Design Support: The development support must provide software design support at different expertise levels in order to address several actors involved in the design, development and operation of context-aware

applications. The fit criteria for this requirement are the depiction of the design process of context-aware applications and the provision of appropriate supportive tools for each actor involved this process.

Programming Support: The software framework of the development support has to provide programming abstractions and models to the developer. This requirement is fulfilled if these abstractions at least cover means of acquisition, transformation, comparison, querying and filtering of context information, management of historical context information, and event triggering.

Metadata Processing: The development support needs to augment context information with metadata expressing quality and user preferences in order to cope with ambiguous, uncertain and unknown data. This requirement is satisfied if the metadata is accessible by any architecture component and covers information such as quality and user preference.

The requirements described above form the basis of the recapitulating assessment of the current state of the art.

3.3.2 Summary

Recent research into context-aware computing has largely focused on the development of software infrastructures that primarily perform tasks such as the acquisition, storage, and dissemination of context information. Three approaches for the management of context can be identified (Winograd, 2001): Context widgets, infrastructure approach and blackboard approach.

Widgets (Dey, 2001; Dey et al., 2001; Dey et al., 1999b) separate applications form the context acquisition issues hiding the complexity of gathering and managing context information. A context widget is a reusable building block providing one-dimensional information that is communicated over a network based on messages and call-backs. They function independently of applications, and thus, permit multiple applications to subscribe to it simultaneously. The widget architecture depicts a tight coupling of system components making it efficient, but complex to configure.

The infrastructure approach such as Ebeling et al. (2001) is equivalent to service-based architectures, which are more flexible than the widget approach. This results in an increased complexity because each component needs to manage network connections and perform more required functionality. It also facilitates the management of components like sensors, services, and components. Applications either have direct access to the components or run discovery processes. This approach focuses more on configurability and robustness and less on efficiency and tight control.

The blackboard model such as Chen (2004) comprises a common shared message board, at which messages can be posted to and components can subscribe to receive messages

matching a specific pattern. In this architecture, all communications go through a centralised server performing a routing of messages to several components of the architecture. The blackboard approach consists of loosely-coupled components based on a general, non-optimised message structure. The model may suffer from efficiency, but is a robust and easily configurable model.

The discussed approaches share the common goal of shifting much of the complex functionality from applications onto a middleware or toolkit, and thereby simplifying the construction of context-aware applications. The evolution of these approaches started with the acquisition and dissemination of context information, continued with the management of context information and the support for decision making and disembogued in the control of adaptive behaviour. However, no approach seamlessly combines all of these functionalities into a single, comprehensive whole. An additional drawback of recent developments is the lack of control and actuator components that facilitate the adaptation of variable parameters of the targeted application based on context changes. Currently, context-aware computing lacks standards for exchanging information as well as a generic architecture and processing pipeline. Reusability of context-aware functionality across different domains has not received much attention. Reusable standard components need to be solidly founded upon a fundamental understanding of capabilities and limitations of context-aware computing, which is still not accomplished to date.

Many approaches found on oversimplified context models that lack a formal and flexible structure, and thus, are difficult to process consistently. Research into context-aware computing has gradually been extended to include all kinds of situational properties, but the variety of potential context attributes and their associated values is currently not considered in a comprehensive way. The same holds for rich types of context information like histories and relations between entities, which still remain the exception. Many approaches address context sensor integration and value abstraction to gain a semantically more enriched context model. With the exception of the work elaborated by Henricksen et al. (2002), most projects emphasise context information acquired by sensors and regard neither statically determined nor explicitly provided user characteristics. All of the investigated approaches and programming frameworks spare the integration of context-awareness and user modelling and do not consider learning user characteristics from changes in the context. However, research arises concerning the exploitation of learning algorithms to generate user models for the use in context-aware computing.

There is a growing recognition of common usability challenges associated with context-aware software. These challenges originate from the lack of transparency of and user control over the application's actions. As a consequence, there emerges a need for involving the end-user in several steps of the software engineering process of context-aware applications. For a flexible user-customisation and control of context-aware software, some architectures restrict the incorporation of end-users to the specification of user preferences and the alternation of preference policies (Henricksen, 2003b) that are hard to understand. However, there has been little research addressing these issues so far. The mediator approach proposed by Dey and

Mankoff (2005) for the mediation of ambiguous context information in dialogue with the user clearly involves the user in the operation of the context-aware application, but remains conceptual work so far.

The approaches and projects presented above only support the implementation phase of the context-aware software engineering process. Not even the roles of people involved in this software engineering process are identified and documented, yet. The work provided by Henricksen (2003a) investigates on the analysis and design tasks for such software and claim for superior modelling techniques and tools. Sohn (2003) and Dey (2004) presented the only visual tool assisting the design of context-aware applications by means of programming by example. Approaches to the understanding of the design process as a whole should result in a consistent and straightforward mapping to respective implementation tools.

The utility of high-level programming models for the reduction of complexity and effort involved in implementing context-aware applications has attracted the attention of many researchers so far. However, the few proposals emerged in this area remain rudimentary and are often limited to traditional programming methods such as abstraction, notification, interpretation and storage. Abstractions provided by some approaches like the situation abstraction (Dey, 2001), preference modelling (Henricksen and Indulska, 2006) and the enactor component (Newberger and Dey, 2003) already depict a trend towards more complex and problem-specific abstractions. There is an urgent need for new techniques for describing and programming with context in highly abstract terms as well as for interpreting and making decisions about context.

Many of the surveyed approaches and projects show a growing awareness of the accuracy of context information obtained from sensors, applications or users. Some allow for quality metadata to be expressed in association with context information. Henricksen (2003a) distinguishes between information that is absent from the context model because it is false, unknown or uncertain, and thus, form a basis of following up such low-quality information. However, such a comprehensive treatment of uncertainty remains unique and other approaches to augment acquired context information with metadata are restricted to very simple mechanisms.

All surveyed approaches lack important context management functionalities for an automated conflict detection and resolution. However, more recent approaches mention and enumerate context management functionalities, but neither give an operational definition nor make concrete implementations available. An additional remark needs to be emphasised: The definition of a reference architecture typically comprises a description of components, protocols of interoperation and a prototypical implementation. Many prototypical implementations of such architectures represent an incomplete mapping of the architecture. Additionally, most approaches claim to be universally applicable, but base this statement on one single example application, which is not sufficient evidence for a universally applicable framework. The detailed assessment of the current state of the art is illustrated in Table 1.

	Holistic Context-Aware Computing Architecture	Context- and User-Modelling Integration	End-User Involvement	Software Design Support	Programming Support	Metadata Processing
Context Toolkit	Acquisition Transformation Dissemination Service	No	Reflection Ambiguity Resolution	Process description	Interpreter (simple) History Situation abstraction Enactor Mediator (concept)	Quality (weak)
PACE	Acquisition Transformation Dissemination Control	Modelling of static and profiled user characteristics	Preferences	Process description Scheme creator	Situation abstraction History Triggering model Branching model Preference abstraction Query API	Quality Accuracy Freshness Ambiguity Uncertainty
Coordinated Adaptation Platform	Control Adaptation	No	Reflection Control policies	No	Event triggering	No
Context Service	Acquisition Aggregation Dissemination	No	No	No	Query API Event Triggering Privacy	Quality Freshness Confidence
Context Broker	Acquisition Aggregation Interpretation Dissemination	No	Privacy policies	No	Privacy	No
TEA	Acquisition Aggregation Interpretation Dissemination Control	Modelling of user characteristics	No	No	No	Confidence
Context Fusion Network	Acquisition Aggregation Interpretation	No	Context model configuration	No	Event triggering	No

Table 1 Detailed Assessment of Existing Approaches

The examination of existing approaches has shown that no solutions exist so far that satisfactorily meet all the requirements that a development support and universally applicable architecture of adaptable context-aware applications needs to fulfil. As a consequence, this observation motivates the development of new concepts and the generation of an innovative architecture for context-aware computing. The following chapters elaborate a functional decomposition of context-aware applications, a corresponding software framework comprising an integrated set of conceptual modelling techniques, and a tool suite that support the developer as well as the end-user throughout the software lifecycle of context-aware applications.

Chapter 4

Context-Aware Computing

This chapter extends the conceptual foundation elaborated in Chapter 2 towards context-aware computing, which is essential for the subsequent design and realisation of a tool suite facilitating the creation of adaptable context-aware applications. Designers and developers who create context-aware applications need to know how to deploy and apply context in their applications in order to achieve the intended behaviour. The provision of a broad development support requires the definition of a software architecture of context-aware applications because it guides the software engineering process for such applications. The design of the software architecture and the embracing development tool suite bases on the understanding of the functionalities and the mode of operation of context-aware applications abstracting from specialised solutions. The targeted software architecture needs to be universally applicable in several application domains in order to support the realization of as many types of context-aware applications as possible and holistic in order to cover all relevant aspects of the application. The sections of this chapter elaborate the systematic and application-oriented decomposition of context-aware applications into their major functional constituents and successively establish the derivation of the required software architecture.

4.1 Context-Aware Applications

The understanding of the mode of operation of context-aware applications starts with the examination and investigation of existing applications taken from different application domains. Chapter 2 has already introduced an example application scenario taken from a specific application domain for context-aware applications. The following table provides an extract of additional application domains and associated research work in these fields.

Museum: The visitor of a museum visually (Oppermann and Specht, 2000) or acoustically (Eckel, 2001) obtains information about exhibits.

Telephony: The user's location is exploited in order to forward calls to the nearest

	phone (Want et al., 1992). A mobile phone automatically selects ringing profiles based on the user's context (Schmidt et al., 1999).
e-Learning:	The learning content is automatically adapted to the knowledge of the student (Oppermann and Thomas, 1996; Specht and Oppermann, 1998; Wulf, 2000).
Tourism:	The context-aware application presents the user with information about sights she is passing (Long et al., 1996).
Facility Management:	Provide engineers with the location of the device to be managed and with an error report if available.
Shopping:	Assist the customer during shopping by providing details about items she is walking by, guiding her through the shop, helping her to locate items, etc. (Asthana et al., 1994).
Fieldwork:	Capture and collect information about the environment and attach it to photographs or recordings taken in the respective place (Kravcik et al., 2004; Specht et al., 2005).
Assisting:	Context-aware assistants manage the user's schedule at a conference (Dey et al., 1999a), in the office (Yan and Selker, 2000) or in everyday life (Rhodes, 1997).
Teleporting:	The context-aware application dynamically displays the user's current interface on the computing resources nearby while the user moves around (Bennett et al., 1994).

Table 2 Application Domains for Context-Aware Applications and Research Projects

Each project from this compilation emphasises certain functionalities of context-aware computing and exhibits specific properties of how context has influence on the behaviour of the application. A clustering of these properties enables the derivation of application classes for context-aware computing that further contribute to the understanding of the functioning of such applications and allow for their decomposition into functional building blocks. The following section describes these classes.

4.1.1 Classes of Context-Aware Applications

The previous section has shown that many sample uses for context-aware applications exist in different fields of everyday life. In order to advance the understanding of context-aware computing, several categorisations of context-aware applications into different classes were proposed in the literature. Schilit et al. (1994) identify four classes of context-aware applications that arise from a schema using the following two orthogonal dimensions: fetching information versus giving commands and manual versus automatic actions. Based on

the four extremes resulting from this two-dimensional matrix, Dey and Abowd (1999) formulate three universal characteristics of context-awareness, which consolidate information and services, and reduce the importance of proximity and the localisation of nearby resources compared to the former proposal. These two approaches combined lead to the following categorisation of context-aware applications:

Proximate Selection (manual information access) denominates an application that provides and emphasises information about input/output devices, non-physical objects and services, or locations in the near proximity of the user. For example, the graphical interface of the system highlights all printers nearby.

Automatic Contextual Reconfiguration (automatic information access) indicates applications that automatically integrate, remove or alter resources, components or information depending on the context. For example, the system automatically starts the screen saver when the external power supply is disconnected.

Contextual Information and Commands (manual action execution) refers to applications that offer actions to the user, which might produce different effects depending on the context in which they are issued. For example, the command "open" starts different applications depending on the file extension.

Context-Triggered Actions (automatic action execution) characterises applications that automatically execute an action, once a certain context occurs. This technique bases on if-then style rules triggered by contextual information. For example, the system plays a sound when the coffee is ready.

Metadata Tagging (automatic annotation) constitute applications that automatically attach relevant context information to physical or virtual objects of the system or the environment in order to ease a later retrieval and filtering. For example, the system attaches the user name, date, weather conditions and GPS coordinates to a picture taken with a camera.

These classes of context-aware applications base on the functionality or features such applications provide, i.e. the purpose of the adaptation a context-aware application performs. Further discriminations of context-aware applications can be proposed by virtue of what kind of information the application exploits for the adaptation, what adaptation methods are applied or how the entire process of adaptation is arranged. Besides the described functional classification, the role of the context in a context-aware application allows for a more natural classification based on the way how the context is used. The following section gives an overview on the four different roles of context.

4.1.2 Roles of Context in Context-Aware Applications

The abovementioned classes of context-aware applications provide a more functionality-based categorisation of such applications. Other approaches apply a more context-centred view to discriminate context-aware applications. Brown (1998) distinguishes between *discrete* and *continuous* applications depending on the rate, at which the context-aware application makes use of the context. Discrete context-aware applications trigger actions at certain well-defined points, whereas continuous context-aware applications always update parts of the application that depend on context. On the other hand, Chen and Kotz (2000) distinguish two ways to use context in a context-aware application: *Active context* automatically changes the application behaviour, whereas *passive context* is presented to an interested user or made persistent for the user to retrieve later.

The definition of context in Section 2.3 entails a different perspective on the classification of context-aware application. The way how such applications make use of context and the role context plays in these applications is important. Taking the different classes of context-aware applications into account, four roles of context can be identified:

Filter: The context serves as a basis of the preparation of a selection of choices. The usage of context for decision support configures queries carried out on possibly large data and information sets and generates a specific subset from this set.

Trigger: The context initiates an activity based on the occurrence of some significant event, i.e. changes in the context. The usage of context as a trigger can cause various actions, depending on the application. The context as a trigger either activates or deactivates an action.

Annotation: The context is used for the generation of metadata in order to tag this information to physical or virtual objects. The captured context can be matched against a filter context in a later retrieval.

Reflection: The context serves as a pure information source. This usage of context comprises the display and recording of context information as well as the exploitation of context information for the parameterised execution of actions (e.g. reconfiguration).

An obvious observation of these four roles of context clarifies that a single context-aware application can make use of context in more than one way. The context of such an application takes over several roles either in parallel, temporally interlinked or in a combination of both. Each role of the context might emphasise another part of the context description. For example, a context-aware museum guide uses changes in the location context of the user to *trigger* a process, which *filters* information about the approached exhibit based on the user's interests. Some context-aware applications allow the user to take over control and play the role of the context. Then, the user pretends to be in a certain context (Brown, 1998) and filters, triggers, annotates or reflects through the manual specification of context information.

A software architecture of context-aware applications needs to incorporate these different roles of context in order to be universally applicable in several application domains.

4.2 Knowledge Contained in Context-Aware Applications

The developer needs to reflect the different roles of context in her specific implementation of a context-aware application. Independently from any software architecture the developer needs to represent and implement the required application- and domain-specific knowledge in the context-aware application. As knowledge-based systems, context-aware applications organise and structure this knowledge in a systematic way such that it can be exploited for the system's proper operation and improvement over time. This section discusses basic questions concerning the knowledge representation in context-aware applications in order to further expand the understanding of how such applications work and to achieve an application-oriented decomposition of context-aware applications into major functional constituents.

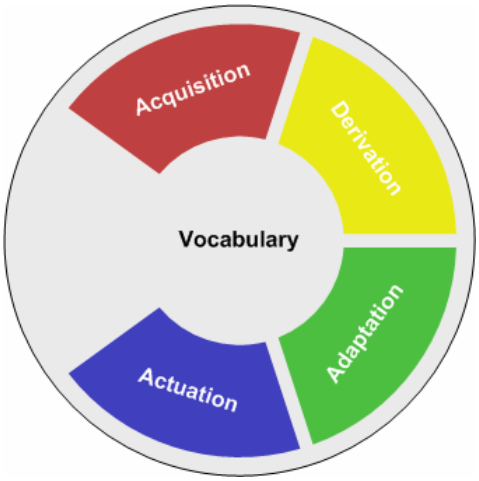


Figure 15 Knowledge Containers of Context-Aware Applications

The elements used for knowledge representation are certain data structures and operations manipulating these data structures. Clusters of such knowledge description elements of a similar type are called *knowledge containers*. Similar to computer programmes knowledge bases exhibit some structure and are organised in modules, each of which solves a specific task and can be accessed by other modules. Each context-aware application maintains different knowledge containers, in which it can store and access the knowledge required for its operation. These knowledge containers depend on each other because no single container is able to completely cover the entire task. Therefore, an inadequately filled container, which depicts knowledge deficiencies affecting quality aspects of the context-aware application, may become a burden to other containers. On the other hand, some knowledge may be

represented more efficiently or easily in one container and more complex in another. Thus, a reorganisation of knowledge containers leads to an improvement of the system over time.

The knowledge container view is relevant in many other areas (Richter, 1995) and provides a framework of organising knowledge related to a context-aware application. The available knowledge of a context-aware application is distributed over the containers *acquisition knowledge, derivation knowledge, adaptation knowledge*, *actuation knowledge*, and the *vocabulary* (cf. Figure 15). These five knowledge containers are the main knowledge sources in context-aware applications, but each of them can be split up into different subcontainers. The following sections provide detailed descriptions of the knowledge containers.

4.2.1 Acquisition Knowledge

The acquisition knowledge covers the knowledge required for observing the context of the users and other entities of the domain. Traditionally and factually, the user's interaction with the system is the most important dimension of adaptivity. In here lies the biggest inter-individual and intra-individual varieties and in here lies the biggest need for adaptation during the usage of the system. Generally, over a certain time period the system needs to detect and record relevant indicators regarding the five categories of context information mentioned in Section 2.4. The knowledge of how to acquire context includes the selection of the appropriate acquisition method. These methods supply information that is provided explicitly by the user, implicitly through observations or from external sources:

Questionnaires or Forms: Questionnaires and forms present an effective means to capture information explicitly provided by the user. The user fills in questionnaires and delivers a variety of valuable information to the system. This sort of input events oftentimes enables the system to learn about the user's properties, preferences, interests, task, etc. because they are explicitly specified by the user.

Tests: Like questionnaires and forms, tests are filled in by the users in order to make information available to the context-aware application. In general, tests serve as means of the determination of the user's knowledge level, competence, and expertise. In addition, third party evaluation, assessments and results of tests (e.g. through tutors in learning systems) can be supplied to the system.

Sensors: As additional sources of information and technical components, sensors measure physical or chemical properties of the users and their environments. Sensors quantify temperature, humidity, pressure, sound, lightness, magnetism, acceleration, force, and many more other properties of the environment. Furthermore, sensors measure the user's blood pressure, body temperature or eye movements as well as the location of the

user in- and outdoor. More recently, the sensors are equipped with built-in microprocessors and are increasingly capable of autonomously processing their signals (so-called *smart sensors* or *smart dust* (Satyanarayanan, 2003)). In self-organising networks (Culler and Mulder, 2004) sensor technologies builds ad-hoc sensor networks and deliver requested information on demand.

Queries: Furthermore, there exists a multiplicity of external sources like databases, pools or external applications that manage valuable information (e.g. user profiles, weather forecasts). Information cannot be gained exclusively through direct interaction of the user with the system or through observation of the user. Additional queries to external information sources may augment the system with a variety of valuable information.

The careful selection of appropriate acquisition methods requires a lot of knowledge because one acquisition method might deliver valuable information for one domain, but might be useless in another. In addition, possible restrictions given by the law have influence on the selection process. Knowledge about the capability and limitations is indispensable for almost any acquisition method, and decisions on possible methods must base upon the following properties:

- *Sample rate*, i.e. the time span between two acquired values
- *Value range*, i.e. the span between the extreme values
- *Scale*, i.e. the minimal difference between two values

In particular for the sensing technologies, the availability of infrastructural aspects, like suitable technology, network connectivity, computing power, storage capacity, and costs, plays an important role in selecting an appropriate context acquisition method.

The determination of the user's knowledge exemplifies possible difficulties in the selection process: using a questionnaire the user in fact specifies her knowledge explicitly. However, this input may be afflicted with several drawbacks because the user might have limited self-assessment capabilities and may overestimate her knowledge. In terms of temporal aspects, the system needs to offer the user sufficient time to submit her input and ensure that the resources are still available at the time of access. On the other hand, if the system uses sensors for the observation of the user, the determination of her knowledge might result from weak interpretation methods like "the longer the user accesses a webpage the more knowledge she will gain". Any implementation of the acquisition procedure will always be subjective and fail in covering the entire spectrum of user personalities.

Example: The example application scenario described by Section 2.1.1 provides a list of examples of acquisition knowledge required for the realisation of this context-aware car crash protection system. In particular, the developer of the alerting mechanism needs to know how to automatically acquire Carmen's stress level in a reliable and accurate way without any biosensors attached to her body.

4.2.2 Derivation Knowledge

The derivation knowledge encompasses knowledge of how to combine and process several types of information in order to enrich already available information. The explicit input provided by the user, the values measured by sensors and the data requested from external sources offer a broad range of information about the properties of the users and their environments. In order to fill the models with meaningful information and to perform an adequate adaptation, this information may not be sufficient, and thus, parts of this information need to be analysed, assessed and interpreted. Basically, the extraction of such higher-level information is a complicated process because boundary conditions apply, double entendres possibly emerge, and potentially undefined or uncertain results arise. The following non-exhaustive list describes techniques for derivation.

Statistical Models: The simplest way of data interpretation is based on statistical models, which are available as large software packages today. The use of statistical methods such as correlation, regression, clustering and time series analysis separates real effects from flukes, and thus, avoids misinterpretations of the user behaviour. Statistical coherences are depicted and backgrounds are uncovered that emerge form interdependencies between events.

Fuzzy Logic: Fuzzy logic provides logic operations to process fuzzy value sets (Zadeh, 1965), while available knowledge incorporates the processing. These sets emerge from the fact that discreet values only can be seen as discreet within a reachable precision of measurement. Fuzzy sets are based on vague definitions of sets and fuzzy logic enables the determination of membership in vaguely defined sets. Therefore, fuzzy logic constitutes a means of expressing uncertainty.

Data Mining: Data Mining goes one step further and refers to techniques for finding interesting and useful patterns and rules in huge data sets. The term data mining includes a number of technologies that allow for the analysis and prognosis of behaviour patterns and trends, and deliver insights and coherences that have been hidden so far. Very often, data mining technologies base on algorithms from the field of artificial intelligence, knowledge management and statistics. The appliance of these algorithms depends on the aim and the purpose of the desired analysis.

Nearest Neighbour: The nearest neighbour algorithm (Duda and Hart, 1973) operates by storing examples or experiences in a training set. A new and unseen instance is classified through the assignment to the class of the most similar example. A prominent example of this algorithm is case-based reasoning. The aim of case-based reasoning is to copy the human way of solving new problems on the basis of solutions for similar

problems of the past and to map this procedure to machine processes. Case-based reasoning (Aamodt and Plaza, 1994) may be suitable for problem areas, in which the knowledge of how a solution is created is poorly understood (Watson, 1998).

Probabilistic Procedures: Probabilistic procedures like hidden Markov models or Bayesian networks offer a method of the representation of uncertain knowledge and resultant possible reasoning. They enable the compilation and representation of coherences, dependencies and independencies of objects in a probability network. Such a network contains the qualitative effects existing among the objects. Following the Bayesian theorem, the relevancy of an (unknown) object can be calculated on the basis of the sum of all probabilities of the effecting objects.

Neuronal Networks: A neural network is an interconnected group of neurons that uses a mathematical model or computational model for information processing based on a connectionist approach to computation (Hertz et al., 1991). The weighted connections between the neurons change their structure based on external or internal information that flows through the network. During the training phase of the network the algorithm iteratively or recursively adapts the connection weights between the neurons based on the presentation of input/output data sets. These weights store the knowledge to the solution of specific problems.

Evolutionary Algorithms: Evolutionary algorithms (Bäck, 1996; Bäck et al., 1997) are approaches for solving difficult optimisation and search problems and are geared to the well-known evolutionary theory: characteristic descriptions of problem solutions are coded and constitute so-called individuals that are assessed by a fitness function. Through repeated selection, recombination and elimination new and improved individuals are created consistently. In the course of generations the fitness of individuals is improved and optimised problem solutions are reached.

Inference mechanisms derive conclusions based solely on information that is already known. Statisticians have developed formal rules for inference from quantitative data and artificial intelligence researchers develop automated inference systems. The selection of appropriate inference mechanisms requires experience and knowledge about their proper implementation, application, training and further processing of their results. Inference techniques address wide areas like search problems, planning, logical deduction, optimisation, and approximation methods. The adaptation process of these techniques to a new problem domain needs to take into account several aspects:

- *flexibility* of the applied algorithm,
- *robustness* against uncertain input,

- *performance* and response time,
- *learning aptitude*,
- *transparency,*
- *understandability.*

In addition, the information or knowledge that serves as a basis of initialising, training or improving the particular algorithms needs to be carefully selected. Decisions must be taken on what type of information can be accessed by the system and combined by the algorithm in order to derive higher level information. If the algorithm should be enabled for learning during runtime, further knowledge is required about how additional information influences the algorithm over time. Furthermore, the time (i.e. number of data sets presented) necessary for a complete training of the algorithms may be crucial. In some domains the interaction of the user with the system only lasts a short period of time, which might be insufficient for learning and producing appropriate results. In any case, the interpretation and further processing of the outcomes of such inference algorithms requires a lot of experience and knowledge because the result may base on imprecise input or assumptions, or it may be a prediction and uncertain per se.

A very popular subcontainer of the derivation knowledge is the context history. The context history contains situations or experiences that the context-aware application recorded during the course of the interaction with the users. As a valuable source of historical knowledge a context history allows for the enhancement of the current context or the restoration of uncertain or faulty context information through interpolation. Through extrapolation trends can be established and future context values predicted. The context history should only contain situations that show a certain minimal utility for the task of the application. Besides the traditional methods of removing situations from the context history that do not contribute to the application's knowledge, generalisation methods can be used to compress the size of the history. Generalisation means summarising several similar situations and removing some of the details, so that it will represent several situations. A theoretical model of learning such key situations can be found in Padovitz et al. (2005).

An example of the crucial application of inference mechanisms is the derivation of the user's mood based on biological sensors. Physiological data such as heart rate, blood pressure and skin conductance may potentially give information about the elevated concentration or excitement of the user.

Example: The derivation knowledge required for the realization of Carmen's car crash protection system comprises algorithms that convert the acquired stress level into a significant value for Carmen's fatigue level. The result of this derivation process indicates the fatigue level by an interval between 0 and 100.

4.2.3 Adaptation Knowledge

The adaptation knowledge container comprises knowledge about how to adapt the system behaviour according to changes in the context. The way the system reacts or adapts its behaviour is based on model assumptions on user needs, heuristics or ontological models of the application domain. The algorithms taking the decision on this behaviour represent the intended behaviour of the context-aware application and contain knowledge about the identification of the most appropriate adaptation the system should offer. Inside this container, the knowledge describes the "method of application" of the domain knowledge and concerns utility issues in order to provide the most useful adaptation and a usable context-aware application. The quality of the adaptation knowledge is a crucial factor for making the context-aware application a benefit rather than a hindrance (Bellotti and Edwards, 2001).

Dieterich et al. (1993) assume four phases of one cycle of the adaptation process: Initiating the process cycle, proposal of alternatives, deciding on one alternative and the execution of an adaptation. In addition, this four-phase process needs to be framed by two extensions: Identification of the need for adaptation and retaining the adaptation after it has been executed. In summary, the six phases of the adaptation process cycle can be described as follows:

Identify: Before an adaptation process cycle can be initiated, the need for an adaptation has to be identified based on changes in the context.

Initiate: If the need for adaptation is sufficient either the user or the system initiates the adaptation process cycle.

Propose: After the process cycle is initiated, one or more alternatives for the adaptation method need to be proposed.

Decide: During the decision phase one alternative of the set of adaptation methods proposed in the preceding step is selected.

Execute: Then, the selected adaptation method is performed using a set of adaptation operations that are conducted by either the user or the system.

Retain: After the execution of the selected adaptation method, the adaptation process cycle finishes with the retaining of the adaptation for measuring the success of the process cycle or for a later inspection.

As basic knowledge-based systems, rule-based systems present a popular way to deterministically express adaptation knowledge because of their simple implementation. The interpretation of rules leads to decisions that determine the role of the context during the course of the adaptation process cycle. Several different subcontainers of the adaptation

knowledge contribute to the execution of each of the six phases of the adaptation process cycle. The following paragraphs describe these subcontainers in more detail.

Adaptation Information

The adaptation knowledge covers knowledge about the need for an adaptation, which is identified and determined by a change in the context, i.e. entering, staying in and leaving a specific context (Schmidt, 2002). In addition, some decisions encoded in the adaptation knowledge affect the awareness of entities that share specific features with each other. Therefore, matching algorithms contain the adaptation knowledge of how to compare the descriptions of the two considered contexts of the entities with each other. The adaptation knowledge container comprises knowledge about what changes in the context are relevant. In this regard, adaptation knowledge is necessary to decide on the extent and the frequency of the context change that are essential for the selection of an appropriate adaptation method and associated adaptation operations. The determination of the extent and frequency usually bases on assumptions and experiences, which may not be optimal in specific single cases. Further adaptation knowledge is necessary to decide how the context information may influence the character of a selected adaptation operation.

Adaptation Goal

Empirical studies conducted by Karger and Oppermann (1991) show that many users have problems applying adaptable properties of an application or do not use them at all. The challenge is to provide enough motivation for users to make use of adaptation operations. Users must trust the application performing the adaptation on their behalf. Therefore, this subcontainer of the adaptation knowledge addresses the trade-off between prescription and freedom, and reflects the degree of user involvement in the adaptation process cycle. Depending on the purpose and task of the context-aware application, a specific (more or less predefined) strategy of adaptation is pursued. Constraint by this adaptation strategy, the adaptation methods align with the (presumably) pursued aims and plans of the user or they misalign. Furthermore, the adaptation knowledge container may consult psychological strategies as well that base on pedagogical or motivational strategies (In particular, this question is addressed in adaptive learning systems, e.g. (Dagger et al., 2004; Felder and Spurlin, 2005)). The knowledge about the goal of the adaptation affects the strategy of initiative of the adaptation that prescribes who is in control of the application in each phase of the adaptation process. Dieterich et al., (1993) distinguish characteristics of adaptive systems by virtue of the distribution of the tasks among the user and the system during the adaptation process cycle. In addition, this subcontainer covers knowledge about the strategy of the confirmation subsequent to an adaptation. This knowledge covers the determination how the user is told about what the system has understood or automatically adapted. Such confirmation can be explicit, implicit, or omitted.

Adaptation Operation

A further subcontainer of the adaptation knowledge comprises knowledge about whether the respective phase of the adaptation process cycle is to be operated automatically by the system, manually by the user or in a mixed initiative manner. An adaptation method abstracts from manual and automatic adaptation, and needs to be instantiated for each of the two concepts separately, i.e. visually for manual and algorithmically for automatic adaptation. In principle, for each of the automatic adaptation methods a manual adaptation method can be constructed. The opposite does not necessarily hold because the more complex manual adaptation methods become the more sophisticated corresponding automatic adaptation methods become (e.g. self-modifying code). The knowledge required for automatic adaptation may vary in complexity, and range from simple corrections of uncertain values to complex composite adaptations affecting various parts of an application. Enabling the user to manually perform modifications to the application means that the chosen adaptation method is made accessible and visible to the user. A finite set of applicable dialogue principles accomplish the manual realisation of an adaptation method. In this regard, adaptation knowledge also contains efficiency knowledge: the utility of one automatic and one manual adaptation procedure, which need different computational effort, may be sufficiently similar. In such a case the two approaches contain different efficiency knowledge. A distinction between design and rendering of an adaptation method allows a clear separation of concerns and provides a rich source of possible combinations.

Example: The adaptation knowledge required for the realization of the car crash protection system covers various aspects. The adaptation goal of the alerting system consists in the prevention of a car accident. The accordant adaptation operation causes a temporally increased volume of the car radio. The adaptation information comprises a change in the fatigue level of the user for longer than two seconds. Thus, this specific context-aware application uses context in the role of a trigger. The required adaptation knowledge could be encoded in a rule that fires if the user's fatigue level exceeds the threshold value "70" for longer than two seconds. If this precondition is fulfilled, the rule activates an adjustment of the volume of the car radio for five second time span.

4.2.4 Actuation Knowledge

The construction of a context-aware application requires actuation knowledge for the realisation of the selected adaptation method or of the entire adaptation process. The methodology of the way how adaptation is performed constitutes an important aspect of the adaptation process. Not only has this methodology an effect on the content of the potential adaptation, but also the necessary skills of the user are determined that are required for the execution of the adaptation in the case of manual adaptation and for the intelligibility of the adaptation in the case of automatic adaptation. The tighter the methodology of the adaptation is coupled with the actual usage of the system the easier the methodology can be applied and learnt. The chosen rendering of the adaptation

immediately affects the methodology. The actuation knowledge container covers the knowledge about effective modifications of the system and comprises two subcontainers for the selection of the appropriate adaptation target and the associated adaptation method. The following sections describe these two subcontainers in more detail.

Adaptation Targets

The algorithms rendering the adaptation method contain actuation knowledge about targets available for adaptation. An adaptation of these system parts may be immediately visible or noticeable to the user or the adaptation effects may be hidden from the user and display their impact at a later point in time. Context-aware applications consider five properties or parts of the system that can be tailored to the context and the user:

Human-Computer Interaction, i.e. the modality needed to enter commands or data, and receive information and services.

Information Presentation, i.e. the methods and coding required for receiving and displaying content (front-end).

Functionality, i.e. the features needed to perform tasks (back-end) and the ability of the application to solve a single task or a set of tasks.

Information and Service Selection, i.e. the information content, density and depth as well as the functionality and complexity of necessary services.

Knowledge Base, i.e. the collection, organisation, and retrieval of the knowledge about and the model of the user, the context and the application.

In a ubiquitous computing system, traditional modalities for data input (e.g. a keyboard) are expanded by other information acquisition methods such as sensors. The same holds for traditional information presentation displays (e.g. a monitor), which can be extended by every possible actuator that may have influence on the environment like motors or Light Emitting Diodes (LEDs). For example, a warning for the user may be rendered as a “Warning” writing on a monitor, a flashing red light or an alerting noise coming through the loudspeaker.

Continuum of Adaptation Methods

Independently from the adaptation targets mentioned above an array of basic methods of adaptation can be specified. The rendering and execution of an appropriate adaptation method is part of the actuation knowledge. Additionally, these adaptation methods are independent from the way an adaptation is performed, i.e. manually or automatically. An overview of several basic adaptation approaches is given in Figure 16.

There exists an increasing complexity from null adaptation on the left, to generative approaches on the right-hand side. The following sections will describe these different approaches and illustrate them. Furthermore, these sections provide details on what knowledge is required for the respective method.

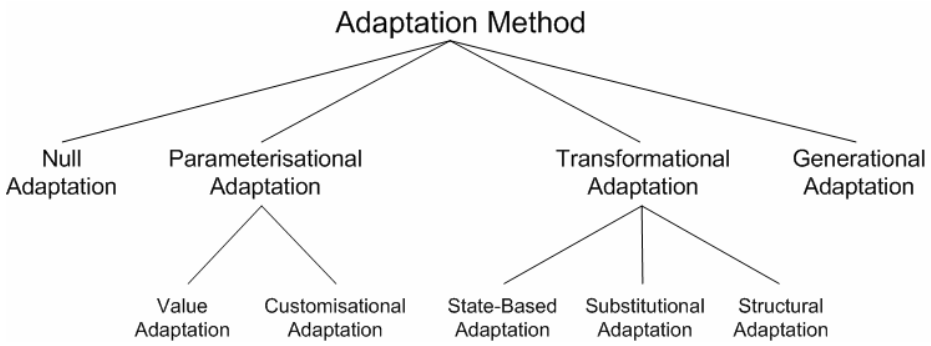


Figure 16 Continuum of Adaptation Methods

Null Adaptation

The simplest way of adaptation is the null adaptation if there is no adaptation necessary. Null adaptation can also mean that an adaptive system leaves the adaptation entirely to the user.

Parameterisational Adaptation

The term parameterisational adaptation refers to changes affecting the internals of elements or components of the adaptation target. This adaptation method changes accessible values of these components and alters parameters in order to reconfigure their behaviour. The parameterisational adaptation can be subdivided into *value* adaptation and *customisational* adaptation.

Value Adaptation

A basic way of adaptation exhibits the value adaptation as the first form of parameterisational adaptation. Value adaptation specifically affects elements of the adaptation target that serve as containers for *values* and addresses the alternation of values contained by these components. The functionality of the concerned component persists as well as the structure of the adaptation target. The knowledge necessary for this adaptation method comprises the awareness of possible value ranges and their type. An example of a value adaptation is the continuation of the location determination through extrapolation (e.g. history analysis) during signal loss of a GPS device in a tunnel.

Customisational Adaptation

Customisational adaptation particularly influences the *functional* behaviour of elements of the adaptation target. Selective alternations of specific parameters of a component lead to changes in its behaviour or view. Independently developed adaptive applications are constructed under a set of assumptions that the developer had to make about the targeted operating environment. Functional configuration allows existing applications to extend their behaviour without the need for reimplementation. Customisation and reconfiguration are the prerequisites for the universal applicability of components. Customisational adaptation does not affect the

interfaces to other components. The knowledge required for this kind of adaptation has to cover the functionality of affected components and means of their customisation. The reduction of the event triggering rate of a GPS tracking component to cope with a low bandwidth is indicative of customisational adaptation.

Transformational Adaptation

Transformational adaptation means that the old structure or composition of the adaptation target is transformed into a new one. This kind of adaptation supports the reorganisation of parts of the adaptation target and permits modification, addition and removal of these elements under certain conditions. Typically, systems performing transformational adaptation employ a fixed set of adaptation operators or transformation rules. Transformational adaptation requires domain knowledge on how certain changes in the structure of the adaptation target lead to differences in its behaviour. Depending on the degree of modification state-based, substitutional and structural adaptation can be distinguished.

State-Based Adaptation

The state-based adaptation method covers all changes to the states of components. Possible switches between component states correspond to activating or deactivating components. As a result of this adaptation method, the (de-)activated component is still present in the structure of the adaptation target, and thus, the structure remains unmodified. State-based adaptation complies with a controlled intervention into the structure of the adaptation target. The application of this adaptation method requires knowledge about dependencies of the affected component with other components within the structure. A simple example of state-based adaptation would be the deactivation of a GPS tracking component because of a sensor malfunction.

Substitutional Adaptation

The substitutional adaptation addresses the replacement of one element of the adaptation target with another. The result of an adaptation method will typically be very close to the initial situation. The structure of the adaptation target remains unchanged. This adaptation method requires similar components to be replaced by each other depicting an approximately similar behaviour. Additionally, the exchanged components should provide the same interfaces for accessing their functionality. Taking the selection of an element from a list as an example, a substitutional adaptation occurs if a list displaying twenty possible selections and a list showing only three possible selections are exchanged because the first one does not fit the screen size of a Personal Digital Assistant. Taking the determination of the user's location as an example, a substitutional adaptation occurs if two tracking technologies based for instance on GPS and WiFi are exchanged with one another because one of them delivers a low quality of service.

Structural Adaptation

More substantial modifications to the adaptation target are performed during the structural adaptation method. Structural adaptation supports the reorganization of elements of the adaptation target and permits the addition and removal of complete elements under certain conditions. Adaptive systems utilizing this adaptation method employ a set of transformation operators, which modify the structure of the adaptation target, depending on the relation between the initial situation and the desired situation. An example for a structural adaptation is the dynamic extension of the context model with an attribute for the user's position, after the GPS device finished its satellite discovery and starts tracking.

Generative Adaptation

Generative adaptation is radically different from transformational adaptation. This adaptation method requires a generative from-scratch (re-)programming or change of the functionality of an element of the adaptation target. Such generators need to be tightly integrated with the adaptive system and might perform a generation automatically or in correspondence with the user. In practice, a pure automatic generative approach is mostly insufficient because of the computational complexity of the generation process or because of the insufficient quality of the results it produces. Such an automatic generator should only generate those small parts of the adaptation target that are inadequate regarding the desired situation. As a consequence, the use of a generative adaptation method requires a different kind of knowledge than the transformational adaptation. Instead of exploiting knowledge that describes how changes of the current situation leads to differences in a potential desired situation, knowledge for constructing (parts of) a solution from scratch is required. This adaptation method is the best example of the end-user programming concept in order to change the code base of a service.

Example: The car crash protection system illustrated by Section 2.1.1 requires actuation knowledge about the adaptation target and an associated adaptation method. The adaptation process of this context-aware application targets the radio built into the car. The particular property of the car radio that needs to be adapted constitutes the volume of the device. The type of adaptation method that needs to be applied for this process is a customizational adaptation.

4.2.5 Vocabulary

As knowledge representation systems, context-aware applications need to make use of data structures and elements of such structures in order to represent primitive notions. Many types of context information exist and for this reason many different types of representation and modelling of this information exist as well. The vocabulary used to describe the domain needs to guarantee that its data structures efficiently respond to queries and requests, and that these date structures are able to capture potentially fast changing values. Thus, context-aware computing can be simplified by an appropriate choice of knowledge representation. The same concepts and their interrelations can be represented

using different notations. The quality of the knowledge within the vocabulary and representation container can be judged by criteria such as:

- *scalability*, i.e. a growing amount of knowledge can be handled.
- *abstraction capability*, i.e. the information content of a concept can be reduced.
- *completeness*, i.e. all relevant properties can be formulated.
- *efficiency*, i.e. interrelated concepts can easily represented.

This knowledge container can be subdivided into two subcontainers for representing the intended knowledge and for the modelling of the domain. The following paragraphs provide an overview on these two subcontainers.

Representation

The representation of knowledge in one way may make the adaptation process simple whereas an unfortunate choice of representation may make the adaptation process difficult or obscure. There is no representation that can serve all purposes or make every problem equally approachable. Common constructs used in context-aware computing are:

Attribute-Value Pairs: Context information is modelled through the context type as a key and the measured data as the value.

Modelling Languages: Based on the Standard Generic Markup Language (SGML) modelling languages are developed and context information is represented as tags and associated fields. This approach enables recursive declarations. An example is the XML-based ConteXtML modelling language (Ryan, 1999).

Object-Oriented Models: The context information is represented by objects that capture and store the acquired data as attributes and offer methods of the manipulation and access of this data. Henricksen et al. (2002) proposed an object-oriented context modelling in which context information is structured around a set of entities.

Logic-Oriented Models: Within a database that bases on an entity-relationship-model context information is expressed as statements in a rule-based system. New rules can be added and queries can be sent to a database.

Other structures like for example taxonomic ones can be build up from attribute-value pairs. The vocabulary serves as a basis of all other containers and it can be used for various types of descriptions, ranging from logical expressions to free text.

Domain Models

Further knowledge required by a context-aware application comprises knowledge about the domain, the system, the users, their tasks, etc. (cf. Section 2.3.2). The accordant information

needs to be mapped onto an adequate *domain model* making data and its meaning accessible to the system. Indirect domain models describe the domain through associated artefacts and comprise descriptions of the domain content like texts, images, videos, etc. Direct descriptions of the domain dispense with third party instances like documents in order to characterise a domain, and can be directly processed by an application.

Complex application domains require knowledge about the system represented by a *system model* in order to better understand, design and control complex phenomena. The system is delimited to the environment and comprises interconnected components that form the system structure: System functions, interaction means, error possibilities, platform, help dialogues, etc. The system model serves as a basis of the definition of a formal semantic of these components and of the abstract representation of the system behaviour. In connection with the task model (cf. Section 2.3.2) the system model allows for the determination of the user's role regarding the usage of the system, and therefore, serves as a basis of adaptations.

Note that the context model holds associations with the domain model as well: a context model consists of domain-independent and domain-dependent parts. On the one hand, the context model contains properties of the respective entity, like for example the age of a user or the dimensions of a car, which can be transferred to other domains. On the other hand, characteristics such as a person's interest in arts strongly need to be interpreted with respect to a specific domain, and therefore, are inapplicable in all domains.

Example: The example application scenario illustrated by Section 2.1.1 describes a context-aware application that filters music according to Carmen's listening history, interests and mood. The vocabulary chosen for this application needs to exhibit such expressiveness that allows for a full description of the music songs and enables a comparison of the characteristics of these songs with the characteristics of the user.

4.3 Maintaining Knowledge Containers

The organisation of knowledge contained in context-aware applications is independent from the architecture of the system. The knowledge containers structure the intended knowledge in such a way that it can be used for a proper operation of the application. Some part of this knowledge is represented directly, e.g. expressed by specific values, and another part is represented indirectly, e.g. hidden in algorithms. Most systems operate even if the knowledge is incomplete, but these systems are not totally exact and not very efficiently to use. Furthermore, an ideal knowledge container does not exist. Such an ideal container would hold all the knowledge that is essential for the context-aware application to operate and to produce results while the other containers cover trivial knowledge.

The five knowledge containers of context-aware applications are not independent from each other. As an example, the generation of context acquisition knowledge already requires an idea about what kind of potentially executable adaptation methods are available and what kind of indicators need to be derived by suitable inference algorithms. This section examines

some of these container interdependencies. It investigates on how the knowledge containers are filled at which time of the application development. Furthermore, this section studies how this initial knowledge of the application can be improved through maintaining the knowledge of individual containers or shifting knowledge between containers. The interplay between the knowledge containers is the reason why such containers play a major role in context-aware applications.

4.3.1 Filling the Containers

The descriptions of the particular knowledge containers of a context-aware application already outline which kind of knowledge needs to be filled into the respective container. The way these knowledge containers are filled depends to a large extend on the way how the knowledge is presented to the system. Therefore, the moment of the filling procedure is splitted into compile-time and runtime. The compile-time corresponds to the time of system design and development, prior to its actual operation, including human knowledge engineering activities. The runtime complies the operational time of the system, when the first sensor values are acquired and the first adaptation methods have been executed and realised.

The first observation leads to the conclusion that all knowledge containers contain compiled knowledge and need to be filled at compile-time. System designers and developers need to understand the required knowledge and compile it into the desired application. They need to identify and model relevant domain entities, describe their interrelations and exchanged messages. In addition, they have to collect suitable acquisition and actuation methods, arrange appropriate derivation algorithms, and define useful and correct adaptation rules. The methods applied to this compilation process are observation, interviews and other empirical approaches, which investigate on a possible or respectively the planned embedding of an adaptive system.

An example of a knowledge (sub-)container that is filled at runtime is the history in which past context information is stored for a later processing. This knowledge container is filled incrementally and the situations can be stored in the history without understanding them at all. The history knowledge is interpreted and understood only at runtime, i.e. when it is required and beneficial for the process of adaptation. Storing knowledge is easier than compiling it; however, more derivation knowledge is required for the interpretation of this knowledge. For compiled knowledge, especially if manually compiled by designers or developers, the acquisition and maintenance task is as difficult. However, for knowledge interpreted at runtime the acquisition and maintenance task is potentially easier because it requires updating of the stored information only.

4.3.2 Knowledge Improvement of Individual Containers

The knowledge containers provide the basis of efficiently development and maintaining context-aware applications. This is due to the possibility of changing the content of a single container in order to improve the efficiency, amount, comprehensibility, and other

performance properties of this container. Basically, for the individual container the knowledge can be improved by human engineers or by applying machine learning techniques. This further compilation of knowledge equals an optimisation which bases on the three principal methods of adding, removing and modifying the knowledge base.

Removal is triggered by redundancy of knowledge that does not contribute at all, or contributes little to the quality of the system. The deletion of knowledge is delicate because it can mean implicit consequences: the deleted fact may be dispensable in one situation, but useful in a later situation. The generalisation and abstraction of knowledge exhibits safer methods of reducing knowledge. Addition is triggered by the detection of useful knowledge that can improve the system quality. Addition can also contribute to the user's understanding of the system. Through semantic enrichment the knowledge becomes more intelligible for the user, e.g. instead of providing the location by GPS coordinates it could be converted into a symbolic representation like street names.

An important subject for improvement constitutes the vocabulary container. The vocabulary is used for encoding the knowledge of the context-aware application and should allow the expression of the expected or desired phenomena. The choice of the vocabulary can therefore exclude certain phenomena from consideration assuming that they might be not important or they simply do not exist. In addition, it is almost inevitable that certain parts of the knowledge will be overlooked in the representation process. The vocabulary could be further refined to cover missing parts, but such a process will also make the vocabulary more awkward to use. Here, a trade-off between the coverage and the usability of the vocabulary seems to be desirable. At some point the vocabulary needs to be finalised because there will always exist an inexpressible phenomenon. Such excluded phenomena will be increasingly strange and unlikely to be encountered, and therefore, it is acceptable to ignore them.

4.3.3 Knowledge Shifts

In fact, the techniques mentioned in the last section for improving one container have an influence on the other containers too. In case there is not enough knowledge available to fill one container as requested, there is a need for knowledge transformation from some containers to others. Knowledge that is transferred into another container does not need to be newly generated. It only needs to be interpreted differently because the knowledge used is already available and coded in some of the other containers. Such shifts between the knowledge containers make implicit use of relationships between these containers. Shifting knowledge from one or more containers to another constitutes an important concept because the improvement of the containers is a major aspect from the viewpoint of software engineering. The improvements affect for example the efficiency, the size of the knowledge base or the comprehensibility. Taking the car crash protection system described in Section 2.1.1 as an example, Table 3 illustrates possible knowledge shifts between the knowledge containers of this simple context-aware application.

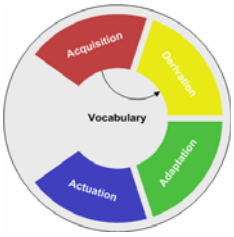
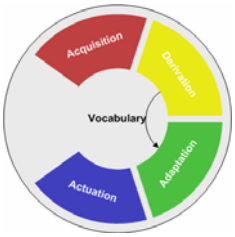
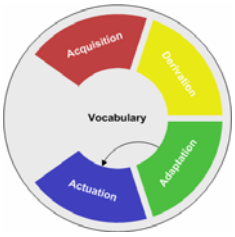
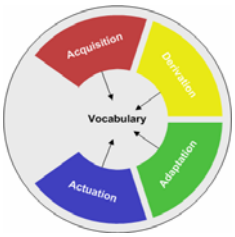
	The stress acquisition mechanism is replaced by one sensor for the heart beat and one sensor for the muscle tension. The knowledge shift occurs if the derivation knowledge container gains knowledge about how to derive the fatigue level from these two sensors.
	Parts of the derivation knowledge can be shifted to the adaptation knowledge container. The algorithm mapping the user's stress factor to the interval [0, 100] can be formulated as a rule in the adaptation knowledge container.
	The knowledge shift from the adaptation container to the actuation container occurs if a new actuation mechanism can be integrated that is able to turn up the radio volume and maintain this state for five seconds. The corresponding rule specified inside the adaptation knowledge container can be simplified afterwards.
	The indicators about a possible success or failure of turning up the radio volume that need to be available to the system are contained in the actuation knowledge. This knowledge can be transferred to the acquisition knowledge by employing an acquisition mechanism of measuring the noise level of the environment (e.g. a microphone).
	All knowledge containers can shift parts of their knowledge to the vocabulary. The knowledge contained in the vocabulary can be extended through the introduction of a new attribute for the quality. For example, this attribute reflects the quality of sensor measurements or a neuronal network.

Table 3 Examples for Knowledge Shifts in Context-Aware Applications

These examples form a circular argument for shifting knowledge between several knowledge containers. Similar examples can be provided through an inversion of the direction of the knowledge shift. These examples clearly show that moving knowledge from one container to the other does not affect or change the total amount of knowledge (at a given time). Hence, functionality can potentially be implemented in one container rather than the other without changing the overall functionality of the system. Additionally, shifting knowledge from an origin container does not always imply that the knowledge has to be deleted from the source.

Context-aware computing currently fails at providing a formalism for knowledge shifts between the knowledge containers of context-aware applications because the explicit interplay between containers is not yet fully understood. Research topics such as knowledge shifts performed during runtime, strategy of knowledge shifting and of the improvement of individual containers as well as the examination of quality concepts require deeper investigation and are out of the scope of this thesis.

4.4 Knowledge Processors: A Layered Architecture

The types of domains of context-aware applications are manifold as the examples presented in Section 4.1 have illustrated. The knowledge containers introduced by the previous sections provide a systematic concept of organising the knowledge base exploited by a context-aware application. Since the knowledge containers are independent from any architecture, they convey a broader understanding of the operation of such application. The provision of support for the design and development of context-aware applications requires the definition of an architecture guiding the software engineering process of such applications.

The software architecture described in this thesis is an abstract representation of the software part of context-aware applications. Because software architectures equate partitioning schemes, describing components and their interaction with each other, the composition of the software architecture of context-aware applications builds upon the foundations provided by the knowledge container view. Each context-aware application bases on functionality that operates on the knowledge containers and processes the covered knowledge in order to realise the intended adaptive behaviour of the application. This functionality can be organised and clustered into several layers of *knowledge processors*. Such a layered approach achieves a loose coupling of components and clean separation of concerns.

Figure 17 illustrates the discrimination into four main layers in a transparent way: *sensor*, *semantic*, *control* and *actuator layer*. Any context-aware application represents a specific instantiation of this four-layer software architecture. Depending on the level of integration with a potentially existing, non-context-aware application, parts or components of the software architecture may already be available. The architecture encloses optional sublayers located on each layer, which a developer needs to instantiate if required, and therefore, some sublayers may not exist in the deployed context-aware application and functions may shift between the layers (Rey and Coutaz, 2004a). In order to achieve a further structuring of the implementation tasks, the following subsections provide a detailed and complete specification of these layers and the information transfer between them.

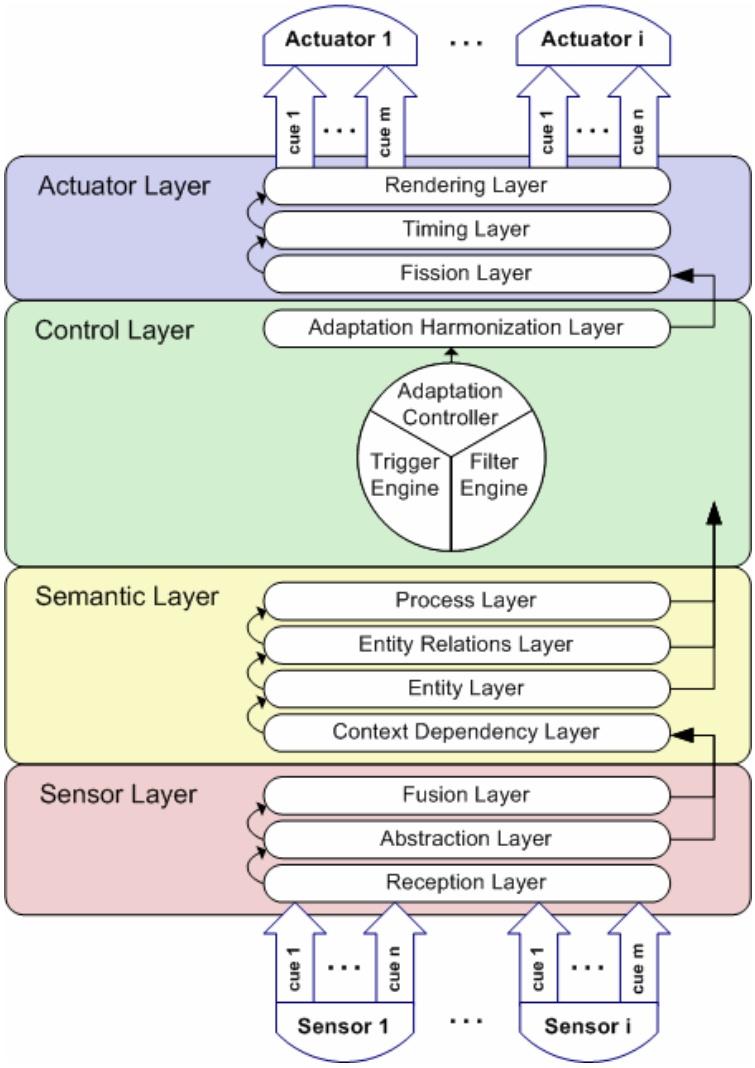


Figure 17 Software Architecture for Context-Aware Applications

4.4.1 Sensor Layer

The first functional layer processes context acquisition knowledge and serves as an information collector. This layer collects data from all context acquisition components attached to the system and presents respective data in a unified way to the higher layers of the architecture. Sensors are components that can provide information about the environment. Each context-aware application relies on a network of such sensors placed in the physical environment and delivering an image of the current situation that the user is acting in. Watching indicators for changing situations, sensor filters detect every change within the environment and perform an observation of the user's behaviour and interaction with the system.

Since the sources from which context information can be acquired vary significantly (cf. Section 2.5.1), the types of sensors vary as well. A rough discrimination can be made between physical and logical sensors. Physical sensors are hardware components that measure parameters in the environment and provide the information on electronic level (e.g. temperature, weight, etc.). Logical sensors are software components that provide information about the context, which is not directly taken from the environment (e.g. system clock). In addition, logical sensors access the Web or databases in order to supply context information to the application. Furthermore, interaction dialogues accepting user input are regarded as sensors because they provide the application with relevant context information as well.

The sensor layer constitutes the sensing infrastructure of a context-aware application, which involves sensor hardware and software drivers as well as low-level communication protocols. The system's sensitivity, speed and accuracy depend on the technology used for sensing context information. Sensors need to be configured dependent on the domain, however, the respective software objects can be implemented in an abstract manner. Abstraction is important to generalise the context acquisition process through offering a framework that allows to plug-in any kind of sensor that provides relevant information. For this reason the sensor layer is established by three sublayers *reception*, *abstraction* and *fusion*.

Reception Layer

The reception layer supports the access to the sensors or sensor cues that can be stationary or potentially distributed in space. Schmidt (2002) formulates the concepts of cues that provide an abstraction of physical and logical sensors, and addresses the fact that many sensors are able to deliver several values (e.g. a clock sensor supplies seconds, minutes, hours, etc.). Each cue depends on one single sensor, but through further calculation based on raw sensor data multiple cues can emerge. Thus, an intermediate layer of cues provides the reception layer with all required sensor values.

Sensors transmit their values about the context via push or pull. If the data is pulled from a sensor, the reception layer as the consumer actively requests the required data and controls the time when this data is requested and used. If the data is pushed by a sensor, the reception layer passively receives the sensor data because the information provider controls the distribution of its data. Whether the sensor values are pushed or pulled immediately affects the implementation of the reception components. The push mode requires the reception component to subscribe to specific sensor cues and listen to changes of the sensor value. For the pull mode the reception component needs to implement specific protocols for querying the sensor.

The software components hosted by the reception sublayer hide the implementation details for accessing sensor hardware. This layer feeds transformed and processible raw sensor data into the abstraction layer.

Abstraction Layer

The main purpose of the abstraction layer is to unify the raw sensor data delivered by the reception layer. The transmission of sensor values via push and pull means that the inquiry of sensors of a context-aware application happens either synchronously, triggered by time intervals, asynchronously, triggered by events, or on demand. Hence, they deliver a temporally diffuse stream of data to the sensor layer. Prior to forwarding the sensor data to the fusion or semantic layer, and prior to a subsequent processing, the abstraction layer discretises this temporally diffuse data stream and compiles it into a unified vector of values. This discretisation includes the analysis and tailoring of data streams, e.g. detecting motion from a video stream.

The two modes push and pull can introduce a considerable timing problem to a context-aware application. Using the push mode, the sensor data can potentially arrive at the sensor layer at any time. Sensors might even push their values continuously. Using the pull mode the sensor data needs to be requested in an interval, in which it could affect the application. If the requested sensor data changes less frequently, pulling the same data all the time means a waste of communication resource.

The components in the abstraction layer take these problems into account and provide means for a straightforward distribution and usage of sensor values to the upper layers. In order to gain an optimization of the information flow, these components allow for a discretization regarding time and value range. They enable the configuration of the desired update rate and the specification of the required granularity of the change in value, which have immediate impact on the abstraction and resolution of the original sensor data stream. The abstraction layer notifies potential receivers on the fusion or semantic layer about a changed value and delivers this value to them.

Fusion Layer

The fusion layer correlates the values belonging to the same type of context information from various sensors. Since the data provided by the abstraction layer already has a homogeneous representation, the main issue of fusion is to exploit overlap and to detect inconsistencies. Particular fusion algorithms combine the data supplied by different sensors to achieve more accurate values. These algorithms base on measures of the quality characteristics of the sensor values such as accuracy, reliability, coverage, etc. The sensor fusion components consider homogeneous or heterogeneous associations of the similar sensor types that can be handled on microcontroller level.

Arranging a number of sensors of the same type into an array often provides additional information that is hard to get from just one sensor. Sensor arrays make a acquisition task easier or even feasible because some types of context information can not be obtained or derived from single sensors (e.g. locating a user in- and outdoors requires different positioning technologies). In addition, in a basic way sensor fusion algorithms take into account the placement of sensors with regard to the observer, i.e. whether the sensor is always

attached with an entity identifier, or the sensor has a fixed position. These algorithms provide basic means of resolving entity identifiers associated with the acquired sensor information.

In ubiquitous computing environments several options exist for creating a sensing infrastructure: direct access to hardware sensors, facilitated by a middleware or acquisition from a sensor server (Chen, 2004). The direct sensor access gives high control over the operation and readout of the sensors. On the other hand, the increased control entails a higher maintenance effort in particular when the number of sensors increases. Middleware approaches facilitate direct sensing through hiding complex implementation details, but trade computation resources of the hosting device for development convenience. As a resource-rich device, a sensor server provides sensor information to several context-aware applications, even if they do not have built-in sensing capability.

4.4.2 Semantic Layer

The semantic layer exploits the derivation knowledge of the context-aware application in order to semantically enrich the value vectors provided by the sensor layer. In addition, the semantic layer defines the context model of the context-aware application because it assigns the sensor values to the respective software objects. This layer holds all possible descriptions of the context of every entity that is relevant and modelled for a given scenario or a particular application. Starting from that, the sublayers *context dependency*, *entity context*, *entity relations* and *process* further utilise the context information.

Context Dependency Layer

The context dependency layer derives and infers semantically enriched context data from sensor data through the consideration of dependencies among contextual information. The derivation knowledge required for this task comprises computations that outreach sensor fusion, take values of various dependent types into account and access (domain-specific) knowledge expressed by overlay models (e.g. a location model mapping GPS data to city and street names). The context dependency layer contains ready-made software objects that are instantiated through a mapping of information provided by the sensor layer to correspondent attributes of these software objects. All context information derived from the values provided by the sensor layers can be classified into one of the categories described in Section 2.4: individuality, time, location, activity, and relations. The description of the context does not necessarily cover all possible information, but all the relevant information the context-aware application requires. These categories introduce a formal structure of context information and determine the design space of context models.

Entity Context Layer

The entity context layer defines all the entities of the domain and associates an appropriate description of the context to each of them. In addition, the components of the entity context layer allow for a further structuring and organising of the context information associated with one entity in order to package this context information regarding specific characteristics such as mentioned in Section 2.4. This enables a special treatment and processing of clusters like static and dynamic (Zimmermann et al., 2002), or public and private (Bulander et al., 2005) context information.

Entity Relations Layer

All entities instantiated in the entity sublayer can potentially relate to each other in different ways. Some entities are part of an interaction, others are spatially or socially related to another, and others are nested (cf. Section 2.4.2). On the entity relations layer, such relations and dependencies between entities are identified and modelled. This layer provides the functionality to dynamically add and remove relations among entities and to assign weights (indicating the strength) and types to such relations. One of the major tasks of the entity relations layer constitutes the allocation of entity-related sensor information to the context of the correct entity. In particular, settings, in which sensors and entities are distributed and mobile, make a correct correlation of derived context information and the associated entity necessary. The identification of relations among entities and resulting shared contexts enable such a correlation because the exploitation of entity relations allow for transferring contextual knowledge from one entity to the other (cf. Section 2.5.3).

Process Layer

The process layer observes the evolution of the abovementioned contexts or parts of the context over time. At this level, complex learning mechanisms derive comprehensive knowledge about the entities and beliefs about future entity relations. For a user, the components in the process layer deduce beliefs about the user's behaviour, preferences, interests, plans, intentions etc. and generate a profile of each one. The application of time series and history modules, statistical models, and intelligent algorithms supports this analysis. The incorporation of learning functionality into a context-aware application can occur prior to the system usage during the design-time or during the usage of the system in a purposive learning phase or continuously (Schmidt, 2002). Since different learning and inference mechanisms may derive different beliefs about the context, a system for the resolution of contradictory evidence and inconsistent beliefs should be employed if necessary.

The semantic layer serves for the flexible knowledge representation of an adaptive system and always represents all available up-to-date information describing the current context. The semantic layer completely covers the profiling task of personalisation engines and provides

different views on the data captured about context enabling a subsequent processing of this information. The semantic layer supplies the control layer with an accurate image of the current and past interaction situations between the system and its users. Controlling components that have registered for the notification about changes in the context, receive context change events associated with the altered context information. The events that are sent to the control layer represent the relevant transitions between situation states.

4.4.3 Control Layer

The control layer primarily utilises adaptation knowledge and controls the adaptive behaviour of the context-aware application by using the context as a trigger, filter, annotation, reflection or as a combination of these four (cf. Section 4.1.2). The control layer provides the link between the two questions, what information is taken into account for adaptation and which part or functionality of the application is adapted and how? Based on the available knowledge about the context model and the pre-processed data provided by the semantic layer, the control layer decides what actions should be triggered if particular conditions in the model become true. Decision support for this task can be constructed from a simple set of rules, but can exhibit more complex structures as well.

The adaptation process activated by the triggering engine is coded into a set of commands assembled by the control layer. These commands may vary in their level of abstraction: Basic *commands* and more complex *strategies*. Commands realise low-level adaptive behaviour, respond to requests or queries received from external applications, or invoke targeted feedback loops. On the other hand, strategies represent more complex macros or plans that can involve a series of consecutive basic commands, whose timing the execution engine needs to take care of. On this strategic layer, the system decides on the highest level whether to behave pro-actively or reactively in the interaction. Altogether, the functioning of the control layer relies upon interplay between a *triggering engine*, a *filtering engine* and an *adaptation controller*, and an *adaptation harmonisation layer*.

Triggering Engine

The triggering engine accounts for detecting the occurrence of significant events through processing adaptation information (cf. Section 4.2.3) and then invoking an adaptation process. The triggering engine accesses a set of triggers from a repository, each of which consists of an event and a corresponding action. Trigger conditions are represented by logical expressions that must be fully satisfied following the corresponding trigger event in order to have the action executed. The context change events sent by the semantic layer activate the triggers as a response to relevant context changes. Therefore, the triggering engine compares the current situation provided by the semantic layer with the abstract situation description represented by the trigger conditions and triggers those that match, which means that a context meets a specific situation. For example, Schmidt (2002) provides three basic triggers: entering a context, leaving a context and being in a context. Since the evaluation of a trigger condition

involves matching of context information, the matching process can be supported by the filtering engine. A trigger either activates (positive trigger) or aborts (negative trigger) the execution of an adaptation process.

Filtering Engine

The filtering of entities regarding specific shared features constitutes a major task in context-aware computing. The filtering engine of a context-aware application as part of the control layer generates a special subset from an arbitrary set of entities (e.g. filtering documents that match the user's interests). The filtering engine implements the two filtering mechanisms *find* and *sort*, and applies matching algorithms guiding and controlling the retrieval process (cf. Brown (1998)). The matching algorithms exploit adaptation knowledge of how to compare the information of the two considered contexts with each other. The matching process accesses three types of information: internal information about one entity context, external information about contexts of several entities as well as existing relations between entities.

Adaptation Controller

The adaptation controller supervises each step of the adaptation process (cf. Section 4.2.3) and encapsulates knowledge about how to achieve an intended adaptation of the context-aware application. The steps of an adaptation process always address a specific adaptation target and perform a specific method of adaptation as described in Section 4.2.4 to this adaptation target. Along these two dimensions the adaptation controller might cover very simple adaptation functionality without any user interaction or highly complex adaptation tasks comprising multiple switches between automatically and manually executed adaptation steps. Whether or not the adaptation process should involve the user depends on the context with special emphasis on the application state, the user's skills and preferences. Additionally, it needs to consider the capabilities of the available adaptation dialogues for manual adaptation that generally fall into the five major categories introduced in Section 4.2.3 and hybrid variations of them. After the adaptation process is finished, the adaptation controller retains the adaptation process and becomes idle until the next activity needs to be executed.

Adaptation Harmonization Layer

The adaptation harmonisation layer resolves conflicting adaptations and coordinates the execution and the rendering of actuators (cf. Efstratiou (2004)). Particularly in the case of system-wide adaptations, an adaptation harmonisation guarantees a coordinated access to resources and services shared by several applications running on the same system (e.g. hard disk or back light of the screen). In addition, in the case of two context-aware applications trying to perform contradictory adaptations, this layer harmonises the effects of the adaptation. The adaptation harmonisation layer processes cross-application knowledge and bases its decisions on system-wide (potentially user-specified) adaptation policies, specifying the mode of operation of applications and defining priorities.

With the assembled command sequences the control layer selects, instantiates and controls the components of the succeeding actuation layer. In addition, the control layer needs to account for the information to be considered during the adaptation process, i.e. for parameterised adaptations. The sublayers of the semantic layer serve as a basis of the required information. The other central parameter of the adaptive method is the way in which the adaptation is displayed or realised in the application, which is strongly dependent on the concrete rendering methods implemented on the actuator layer. The control layer also offers a direct communication link to other external applications or systems and responds to simple requests and queries.

4.4.4 Actuator Layer

The actuator layer represents the executive body of a context-aware application and processes the actuation knowledge of application. This layer handles the connection back to the environment by mapping the decisions taken by the control layer to real-world actions. Speaking of ubiquitous computing, the entire environment and everything the system can have influence on can be the "display" of a context-aware application. The sublayers of this layer implement domain-dependent methods that directly change variable parameters of the environment and the application. Depending on the level of integration with the application, these methods may be part of the application or the commands assembled by the control layer may have to be transformed into appropriate actions. As a feedback for the control layer, messages indicating the success or failure of actions are sent back.

As specialised software components located in the actuator layer, actuators process the adaptation method proposed by the control layer. In analogy to the sensor layer, each context-aware application relies on a network of such actuators that can be discriminated into *physical* and *logical* actuators depending on whether they are placed in the physical environment or in the application itself. The actuator layer comprises the three sublayers *fission*, *timing* and *rendering*.

Fission Layer

The fission layer disassembles the commands received from the control layer and distributes the adaptation operation among the respective actuators and actuator cues realising the adaptation operation. Actuator cues resemble the sensor cues mentioned in Section 4.4.1. They abstract from physical and logical actuators, and address the fact that many actuators enable the rendering of several activities (e.g. a video actuator renders video and sound). Hence, each cue is dependent on one single actuator. The fission layer assigns new values, content, change requests, etc. to the respective actuators cues and accordant fission algorithms base on measures of the quality characteristics of the actuator performance such as availability, resolution, etc. Furthermore, accordant fission algorithms provide basic means for resolving adaptation activities associated with specific entity identifiers in order to ensure that the activity reaches the respective target entity.

Timing Layer

The timing layer is responsible for unification of the execution of the adaptation operation regarding timing issues in order to obtain a homogeneous adaptation (e.g. receiving information about a place the user just passed). This layer intercepts the delay introduced by diverse computations of the context-aware application and harmonises the adaptation process. Furthermore, the timing layer considers the period an actuator needs for the execution of an adaptation. In particular when the adaptation methodology requires input from the user, other adaptation activities might be pushed to the background and pause for a certain time period. Furthermore, the timing layer takes potential delays in the communication between the context-aware application and its actuators into consideration (e.g. streaming video requires more bandwidth than transferring an email).

Rendering Layer

The rendering layer processes the delivery of instructions and content to the respective actuator cue. The rendering components constitute the actuating infrastructure of a context-aware application, which involves actuator hardware and software drivers as well as low-level communication protocols. In addition, rendering components address the problem of pushing and pulling values, actions and information to the actuators, which holds for sensors just like for actuators. The push mode requires the actuator cue to subscribe to specific rendering components that push changes of the component to the actuator. For the pull mode the actuator needs to have specific protocols implemented for querying the rendering component.

4.5 Actors in the Adaptation Process

In context-aware computing, several actors can be identified, who are part of the realisation of the adaptation process. This identification of actors is essential to better understand the creation, usage and management of context-aware applications. Such an analysis is necessary provide an accordant tool suite that supports each actor in its specific role in the software engineering process. During the entire software engineering process of context-aware applications the same human actor may assume different roles (cf. Section 6.1.9 for examples).

Table 4 summarises the five main actors in the adaptation process of a context-aware application: the *application*, the *user*, the *developer*, the *expert* and the *author*. This distinction can be more fine-grained, but the additional actors can be complemented by activities dedicated to the five basic groups of actors. In the following, these five types of actors are described in more detail.

The two obvious actors playing the main part in the adaptation process are the *context-aware application* for automatic adaptation methods and the *user* for manual adaptation methods. Traditionally, the context-aware application initiates and executes the adaptation process. As

illustrated in Section 2.1.2, the user needs to be actively involved in the adaptation process, and thus, the user participates even in the application of automatic adaptation methods. Dieterich et al. (1993) show the interplay between the user and the application in the different phases of the adaptation process.

Actor	**Activity**
Application	identifies the need for an adaptation proposes, initiates and/or performs implemented adaptation methods provides tools and methods
User	receives automatic adaptations accepts, rejects or modifies proposed automatic adaptations executes manual adaptations selects, performs and uses adaptation tools and methods
Developer	encodes acquisition, derivation, adaptation, and actuation knowledge follows a structured software engineering process integrates adaptive and adaptable methods
Domain Expert	assembles and configures the targeted application tailors the existing applications through reconfiguration extends the application implementation informs users about adaptation effects, methods and tools
Author	creates and composes the appearance of the application constructs and administrates content and services the application provides keeps provided content and services up-to-date conducts test runs

Table 4 Activities of the Five Actors in the Creation of a Context-Aware Application

Prior to the delivery and operational use of a context-aware application, *developers* implement such a complex system following a structured software engineering process. After the requirement engineering and planning phase this third group of actors complete the system comprising software and hardware step by step. The developers integrate the targeted adaptive and adaptable methods into the application in order to broaden the usage of the system. Additionally, the developers implement mechanisms that allow the user to accept or reject automatic adaptations proposed by the application or have influence on an automatic adaptation process. Developers play an important role as an actor at adaptive systems because they decide on and design the algorithms and conditions for adaptation activities.

The developers should build systems that are usable and suit the needs of many users. For the reasons mentioned in Section 2.1.3 and 2.6, certain dynamics limit the fit of the application characteristics to the user requirements after system rollout. Therefore, tailoring of the system or adding new features might become necessary after the development phase during the operational use of the system. Since the time between the emergence of new user needs and a professional development might be too short, *domain experts* perform the required adjustments and configurations to the system. Such an expert knows more about the adaptation possibilities than the user and has the better ability to execute them. As an interface or adapter between the system (or the developer) and the user, the domain expert might propose adaptation alternatives, which are neither automatically uncovered by the system nor investigated by the user. The expert's knowledge about the essential methods and tools links to approaches like end-user development (Fischer, 2002; Lieberman et al., 2006) addressing the configuration and the further development of applications.

A domain expert can accompany the adaptation process from the initial phase on and cooperate with the users, which all contributes to a better fit of the context-aware application with the requirements. Depending on the usage context of the application, *authors* administrate the content and services context-aware applications provide. In their role as domain experts, authors mange the information and service repository of context-aware applications. They care for the addition of new information and the removal of outdated information, and keep the provided content and services up-to-date. The author creates or composes appearance of the context-aware application without actually knowing the entire internals of the application. Thus, authorship bases on skills regarding creativity rather than programming. Furthermore, the author conducts test runs of the application without any user to check whether or not the application presents the desired content and services correctly.

4.6 Summary

This thesis targets the extension of the spectrum of actors involved in design, implementation, authoring and configuration of context-aware applications beyond developers in order to reduce the usability problems introduced by context-aware computing. The extension of this spectrum of actors bases on a common and comprehensive understanding of the field context-aware computing and a shared perception of the adherent concepts. Chapter 2 already provided the conceptual foundation of the terms "context" and "context-awareness". This chapter extends this conceptual foundation towards context-aware computing and introduces a software architecture of context-aware applications. The provision of development support of adaptable context-aware applications requires the definition of such a software architecture because it guides the software engineering process for such applications. The programming and architectural-level support for adaptable context-aware applications needs to follow a holistic approach covering all relevant aspects of the application (see the key requirements listed in Section 3.3.1). In addition, the targeted software architecture of context-aware

applications needs to be universally applicable in several application domains in order to support the realization of a broad range of types of context-aware applications.

The derivation of the intended software architecture demands the systematic and application-oriented decomposition of context-aware applications into their major functional constituents. A first decomposition approach constitutes the analysis of existing context-aware applications taken from several heterogeneous domains and their subsequent clustering into classes based on conjointly emphasised functionalities and exhibited features. However, the accompanied study of clustering approaches documented in the literature revealed different and disparate perspectives on the classification of context-aware applications (cf. Section 4.1.1). Consulting the definitions of terms related to context-aware computing elaborated by Chapter 2, which focus the operational aspect of context, entails a more context-centred view on the classification of context-aware applications. Context may play four roles in the adaptation process of context-aware applications (cf. Section 4.1.2): filter, trigger, annotation and reflection. Any universally applicable software architecture of context-aware applications needs to incorporate these different roles of context.

These investigations form the basis of an analysis of the knowledge that a developer requires to reflect and utilise different roles of context in her specific implementation of a context-aware application. Independently from any architecture the knowledge base of a context-aware application can be systematically organised and structured into five knowledge containers: acquisition knowledge, derivation knowledge, adaptation knowledge, actuation knowledge and vocabulary. Each context-aware application maintains these five knowledge containers, in which it can store and access the knowledge required for its operation and improvement over time (cf. Section 4.2). Dependencies among these knowledge containers allow for a reorganisation, maintenance and improvement of the overall application knowledge over time. These dependencies inspired the creation of a modification framework of the knowledge contained in context-aware applications (cf. Section 4.3). This framework explicates how the knowledge containers are filled at which step of the development process and how this initial application knowledge can be improved through maintaining the knowledge of individual containers or shifting knowledge between containers.

The knowledge container view results in an application-oriented decomposition of context-aware applications into major functional constituents and incited the design of the four-layer software architecture of context-aware applications (cf. Section 4.4). The assembly of this architecture arises from the clustering of the functionalities operating on the knowledge containers and processing the contained knowledge in order to realise the adaptive behaviour of a context-aware application. The software architecture based on the knowledge container view fosters the reusability of context-aware functionality across different domains, which reduces the high application development overheads for inexperienced developers (Chapter 6 provides a validation of the universal applicability of the elaborated software architecture).

Current research into context-aware applications suffers from putting too much focus on input and too little focus on output (Salber, 2000). The elaborated software architecture of context-

aware applications copes with the limitation of recent approaches through a seamless combination of all context-aware functionality into a single, comprehensive whole. At multiple levels of abstraction, the elaborated software architecture covers the end-to-end process chain ranging from the acquisition of context information to the realization of adaptations.

The software architecture equates a partitioning scheme, which describes the constituents of a context-aware application and their interaction with each other. The fine-granular compilation of the layers and sublayers enables a flexible coupling with a potentially pre-existing core system. Any development support that bases on this software architecture can complement pre-existing functionality with required context-aware functionality. Depending on the level of integration implementations of unneeded sublayers may simply be bypassed or instantiated with trivial functionality.

The software architecture and the accompanied knowledge container view on context-aware applications communicate a common understanding of the functioning of such applications. The presented software architecture forms the basis of the practical realization of a development support that aims at a facilitated creation of adaptable context-aware applications. However, such development support needs to address the demands of all actors involved in the software engineering process of context-aware applications. Currently existing approaches to development support only support the implementation phase of the software engineering process and not even the roles of people involved in this software engineering process are identified and documented, yet. Therefore, an investigation on different actors and their roles in the design, implementation, authoring and configuration of context-aware applications resulted in the identification of four key actors (cf. Section 4.5): the developer, the domain expert, the author and the end-user. The results and findings achieved by this chapter establish the transition to the succeeding chapter addressing the realisation of an appropriate suite of tools supporting each actor in her specific role in the software engineering process of context-aware applications.

Chapter 5

The Context Management System

The creation, usage and management of context-aware applications involve several human actors as identified and introduced in the last chapter: the *end-user*, the *developer*, the *domain expert* and the *author*. An accordant tool suite for the facilitated realisation of context-aware applications needs to support each actor in her specific role. Developers need to encode acquisition, derivation, adaptation, and actuation knowledge in their specific implementation of a context-aware application. Therefore, developers need support in building an application that is usable and suits the needs of many users. Experts establish the link between the developers and the end-users because they know about the adaptation possibilities of the implemented context-aware application. Therefore, experts need support in taking tailoring measures and performing fast configurations to the operational application. Authors administrate the content and the services of the implemented context-aware application. Therefore, authors need support in the composition of the appearance of the context-aware application. Finally, end-users need support in the adaptation of the implemented context-aware application aligned with their programming skills.

The demands of these four actors and the seamless support of all tasks involved in the realisation process of a context-aware application have considerable implications on the design of the *Context Management System* introduced in this chapter. This Context Management System facilitates the development of adaptable context-aware applications, which unite adaptivity and adaptability (cf. Section 2.1.2). In this sense, the term *Context Management* addresses the different actor's tasks and summarises the construction, integration, authoring, administration and tailoring of context-aware behaviour. This chapter discusses the design of the Context Management System and describes all its constituents and development support in detail.

5.1 Design of the Context Management System

This section introduces a Context Management System that facilitates the development and maintenance of adaptable context-aware applications through the hiding of complex technical details from developers and end-users. Additionally, this system allows the flexible enhancement of existing applications and services that lack adaptive and context-aware functionality. The Context Management System comprises two main parts as shown in the centre of Figure 18: The Context Toolkit (left) and a content management system (right).

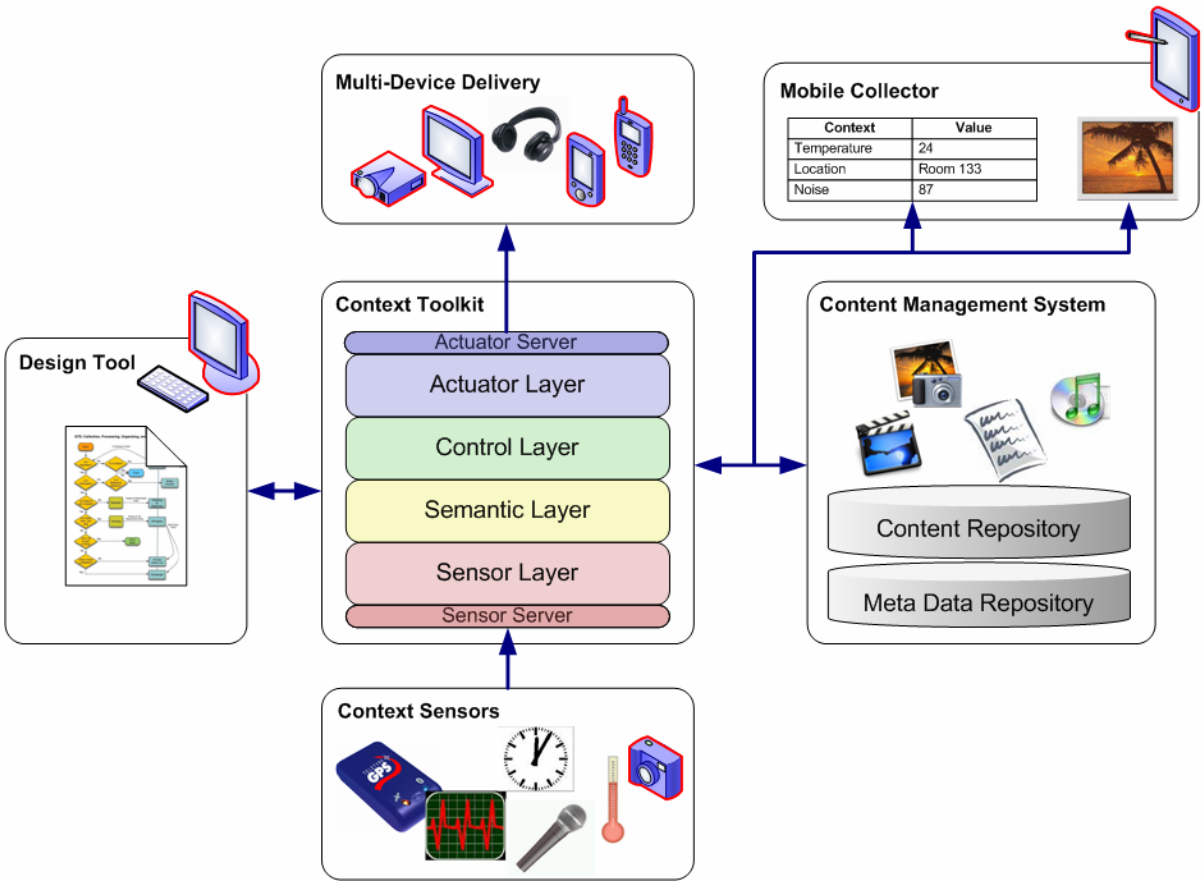


Figure 18 Constituents of the Context Management System

The content management system represents the information and service repository of the targeted context-aware application. Many context-aware applications base on such a repository of contextualised information because it can be triggered if the user enters, leaves or stays in a specific context. Depending on the type of context-aware application, the triggering can cause various actions ranging from presenting the information to the user to running a programme. The content management system enables the administration of this content through application authors, which involves creation, deletion and updating. Content in this sense may be any digital information such as text in the form of documents, audio or video files, multimedia files, or any other file type, which requires management.

As the core constituent of the Context Management System, the Context Toolkit (cf. Section 5.2 and 5.3) offers a software framework and a library of ready-to-use software components, which developers can employ and exploit during the software engineering process of adaptable context-aware applications. A suite of tools encompasses the Context Toolkit and the content management system, which involve experts, authors and end-users in the development process. The Design Tool (cf. Section 5.4.1) enables a modification of the design view of the adaptable context-aware application. This tool exploits the configurability of the Context Toolkit and allows a tailoring of the operational application. The Mobile Collector (cf. Section 5.4.2) facilitates the creation of contextualised content. This tool associates an element selected from the content management system with the context of an entity represented by software components of the Context Toolkit. The Content Player (cf. Section 5.4.3) enables the context-aware presentation of content from the content management system. This tool retrieves content from the application's repository that suits its current context and serves as a display of the operational context-aware application.

The remainder of this section enumerates and describes an array of general design considerations that forms the character and affects the appearance of the Context Management System. Note that while some of these considerations may seem obvious in retrospect, the contribution of this section lies not only in identifying them, but also in illustrating how the design of the Context Management System addresses these issues.

5.1.1 Simplicity versus Complexity

The paradigm of End-User Development (Fischer, 2002; Lieberman et al., 2006) aims at making systems that are easy to develop and empowers end-users to configure and compose the information technology according to their diverse and changing needs. At the core of End-User Development research is the question, how to reduce the complexity the user is confronted with when adapting and configuring technology? Therefore, Henderson and Kyng (1992) and Mørch (1997) introduced three levels of complexity that avoid big leaps in complexity and address users at different stages of expertise and development skill. These levels allow users to

- select between predefined behaviours,
- compose a desired application out of existing modules, and
- fully access the code base of an application.

This property of avoiding big leaps in complexity to attain a reasonable trade-off is called the *gentle slope of complexity* (Beringer, 2004; Wulf and Golombek, 2001). Users have to be able to make small changes in a simple way, while more complicated ones should only involve a proportional increase in complexity the user is confronted with.

The Context Management System achieves this gentle slope of complexity through the increase of the flexibility of the underlying technology. Object-oriented and component-based software paradigms allow for the introduction of different levels of complexity that address

several expertise levels of a variety of users according to the actors and roles in the adaptation process introduced in Section 4.5. The Context Management System introduces four complexity levels that can be exploited complementary:

Code Base: The code base offers a large collection of reusable core software components to experienced developers.

Programming Abstractions: Based on the software components provided by the code base, programming abstractions allow for structured programming with well-known concepts from the field of context-aware computing through reducing the details of the underlying implementation by experts.

Configuration: The configuration of the context-aware application enables designers of context-aware applications to tailor the arrangement of functional units of the generic architecture described in Section 4.4 to a concrete project. Changes in the configuration affect the function and performance of the context-aware application.

Tools: A set of tools equip authors of context-aware applications with instruments and user interfaces for the contextualisation of their content and means of the deployment and testing of their application.

The configurability of the context-aware application makes this application an adaptable context-aware application because the configuration can happen at design-time and at runtime. For end-users the combination of the configurability and the tool set provides the basis of the balance between full user control over the application behaviour and application autonomy. The appropriate position along the continuum between full user control and application autonomy will be dictated by the user's needs, situation and expertise.

5.1.2 Toolkit versus Infrastructure

The primary goal of the Context Management System is to reduce the complexity of designing, developing and deploying context-aware applications through the provision of a common understanding of the functioning of such applications and instruments for their realisation. A support for the tasks involved with the generation of context-aware applications can be guaranteed by software libraries, frameworks, toolkits, service infrastructures or other approaches.

A library comprises a generalised collection of algorithms and software components that are related to application-specific tasks. Libraries hide complex details from the developers and offer abstract access to potentially extensive functionality. Hereby, libraries exclusively focus on code reuse.

Frameworks emphasis design reuse by providing a basic structure of a certain class of application (Hong and Landay, 2001) and define a support structure in which a software project can be organised and developed. Frameworks offer fundamental functionality for

assembling components of one specific application class and provide means of customisations addressing specific needs.

Toolkits unite frameworks with libraries through providing a customisable reference or skeletal implementation while offering a set of basic building units. Some toolkits decouple the functionality of these building blocks from their definition and allow defining them at the initialisation of the application or even at runtime. A toolkit manages the creation and behaviour of the reusable components, but leaves some of the responsibility to the targeted application as well.

An infrastructure offers a well-established and publicly accessible collection of technologies that act as a foundation of other systems (Hong and Landay, 2001). Any application running on any device can access services from an infrastructure over a network through specific protocols and data formats. Thus, the interoperability problem can be eliminated and developers only need to know how to access the provided services. A functionality update of such services can be performed without any changing of the accessing application.

5.1.3 Distributed versus Centralised

A multitude of context-aware applications are deployed in environments, in which parts of the application such as sensors and actuators need to be distributed. The distribution occurs on two different levels: on a conceptual level where information is distributed and on an implementation level where system components are distributed. Typically, context-aware applications decouple the context acquisition functionality from the actual application. Thereby, the development effort can be decreased because of the large development overhead of interacting with a variety of sensors to capture the context, interpreting it into the desired format and disseminating it to interested applications. Such a separation of low-level functionality from high-level applications requires the introduction of a middleware layer, which typically can be achieved through centralised and distributed approaches.

A centralised approach bases upon a centralised context server, which provides contextual information to the applications and decouples context acquisition from context processing. This server collects all context information from context acquisition components and provides it to interested applications. These applications can actively request the desired information from the server or passively be notified about current values. Instead of maintaining all context information in one centralised place, a distributed approach holds the information at several places to avoid a potential bottleneck. Small devices acquire the context information required by the application themselves and process it directly. This approach requires the device to have the capability to sense and process all of the necessary raw contextual information, which may not be efficiently achieved for a simple device with restrictions concerning space, weight, or energy consumption.

An important characteristic of context information is that it is shared between entities. Some sensors acquire public context information that needs to be forwarded to interested components. In the same manner, context-aware applications may potentially address groups

of entities with the execution of specific public actuators. Therefore, an appropriate structuring of such a context-aware application through a system architecture is essential. The implementation of the architecture introduced in Section 4.4 offers one server in the sensor layer receiving raw sensor data and one server in the actuator layer managing to address the attached actuators. The sensor server implements a publish-subscribe model, which notifies registered components about changes in the received ray sensor data. The actuator server applies remote procedure calls for controlling remote actuator components. In principle, all four layers of the architecture can be separated and distributed because the set of messages exchanged between these layers can be transformed into messages that can be processed by the instant messaging server Jabber (see Jabber (2007) for more detailed information). Jabber is a set of streaming protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and other structured information in close to real-time.

5.1.4 Discrete versus Continuous

Context-aware applications can roughly be distinguished into continuous and discrete applications (Brown, 1998). Continuous context-aware applications continuously change their behaviour as a response to the continuously changing context of the user. Examples of such application would be one that adjusts the volume of a sound as the user gets closer to an object or one that displays the user's current GPS position on a map. Since such applications always take the user's current context into account, they provide an immediate and high-resolution response to the user, but need to handle a large amount of real-time information at the same time. The realisation of a proper behaviour of a usable continuous context-aware application requires serious programming effort.

In contrast, discrete context-aware applications associate one discrete piece of information with one specific situation the user might be in. This subset of context-aware applications trigger the discrete piece of information when the user enters, stays in, or leaves the situation attached to the specific information. The determination of a set of rules for triggering the information constitutes an important part of the creation of discrete context-aware applications. Thus, authoring these applications can solely be a creative process rather than an implementation task, which makes them accessible even for end-users.

The Context Management System supports the development of both types of context-aware applications. Preserving the gentle slope of complexity (cf. Section 5.1.1) the least complex way to create a context-aware application consists in the attachment of a piece of information to a specific situation and the application of the built-in automatic triggering functionality. Since continuous applications cannot be modelled in terms of retrieving discrete pieces of information from a repository, the Context Management System offers a programming toolkit which reduces the implementation effort, but which represents the most complex way to create a context-aware application as well.

5.1.5 Design- versus Runtime

In discussing the design of context-aware applications, it is important to keep the distinction between the design-time view (i.e., the markup) and the runtime view (the software that executes the markup) in mind. Both views base on individual software components that are responsible for specific tasks within the context-aware application. These components can be perceived as black boxes, required only to implement the application programming interface for each type of component. This interface allows the components to communicate with each other through the exchange of events or messages.

At the design level, the context-aware application takes the form of a set of markup documents that serve as a description of the components of the context-aware application and partially of the procedure of how these components process data. The markup language describes properties, memberships and operations and allocates them to basic components of the application. In addition, the markup language prescribes the sharing of information (i.e. the interaction) between these basic components and allows for a nesting. The repository containing the markup language and defining the components of the application can be one single document or distributed across multiple documents. Thus, through these markup documents the design view on context-aware applications facilitates the construction of such an application that would require far-reaching programming skills otherwise. Taking the principle of encapsulation into account, markup languages are not required to reflect directly a defined architecture and application programming interfaces.

At runtime, the context-aware application architecture features loosely-coupled software components that may be either co-resident on a device or distributed across a network. Since the design view forms a prescription of the assembly and the behaviour of the context-aware application, the runtime view needs to reflect this prescription. Therefore, the runtime view employs designated software containers for the markup documents, which interpret the markup language and instantiate, initialise and couples corresponding software components. The runtime view on context-aware applications distinguishes between the runtime framework and these software components. The runtime framework provides the basic infrastructure, which various software components plug into, and manages the combination and coordination of multiple processing components. In keeping with the loosely-coupled nature of the architecture presented in Section 4.4, the components communicate only by exchanging events, which may be commands, replies to commands or notifications about changes.

The Context Management System applies the eXtensible Markup Language (XML) as the language for the markup documents. Although nothing in the context-aware application architecture requires any particular correspondence between the design-time and runtime views, many dedicated software components exist, which are responsible for each different type of markup document. The Context Management System employs one designated manager and reserves one separate namespace for each type of software component that owns an accordant design view. At the markup level, a top level document for all markup

components, which provides the root of a document tree, does not exist. The documents can be linked either by *reference*, which uses a linking construct to reference one document from another, or by direct *inclusion* of a document from one namespace within a document from another namespace. The runtime framework of the Context Management System handles all communication by events and links all components through a default event-response connection. This link needs to be modified if additional facilities for sharing specific information or a specific data model need to be provided that allow for a closer coupling than that provided by the base interfaces. The following section discusses the nature of the runtime software components and the application programming interfaces between them in more detail. Appendix A covers the complete notation of the design view in Extended Backus-Naur Form (EBNF).

5.2 Context Toolkit

The core component of the Context Management System consists in the Context Toolkit, which provides software developers with an extension to the JAVA programming language for the implementation of context-aware applications. The libraries consist of ready-to-use software components that hide complex technical details from the developer. The implementation of the Context Toolkit realises the layered architecture presented in Chapter 4, and follows a component-based approach. Such an approach facilitates the independent addition, removal, and replacement of components on each layer and allows for a focused allocation of resources to the several layers (e.g. experts in context acquisition may work independently from experts in machine learning or decision making). Furthermore, the Context Toolkit provides the basis of adaptable context-aware applications through an accordant design view. Therefore, XML configuration documents allow for an instantiation of the software components as well as a subscription of these components to the event flow of the application. For each of the four layers the following sections describe the maturation of the implementation and document the usage of its configuration ability. The figures illustrating the modelling of the software system base on the standardised Unified Modelling Language (UML).

5.2.1 Sensor Layer

Sensors are specialised software components that process small tasks like qualitatively or quantitatively measuring physical or chemical properties. *Smart* sensors are equipped with small processors that enable a more intelligent information acquisition (i.e. “smart dust”, further reading in Satyanarayanan (2003)). The sensory function of a context-aware application is mostly distributed over different devices or embedded in the user’s environment. Thus, sensors are organised in a network and a sensor server can receive all data, processes the information and delivers inferred knowledge towards desired applications.

The implementation of the sensor layer in library `toolkit.sensor` of the Context Toolkit consists of software objects that receive incoming data, perform a cleaning of this data and accomplish a fusion of the sensor values. The access to current sensor values occurs on demand, through the sensor interface method `getValue()` or event-based, because the sensor objects fire sensor change events that notify other software objects (e.g. in the semantic layer) about changes in the current sensor values. Figure 19 shows the sensor interface each sensor component needs to implement. The method `getQuality()` constitutes an important method of this interface because it returns an assessment about the quality of the sensor value, which considers and joins aspects such as precision or signal availability. This quality assessment enables components receiving sensor values to decide on the further treatment of this value. Any component that implements the `SensorChangeListener` interface can register to any sensor via the `addListener(listener: SensorChangeListener)` method in order to be informed about these changes. All sensors extend the `AbstractSensor` class, which provides a basic implementation of the sensor interface.

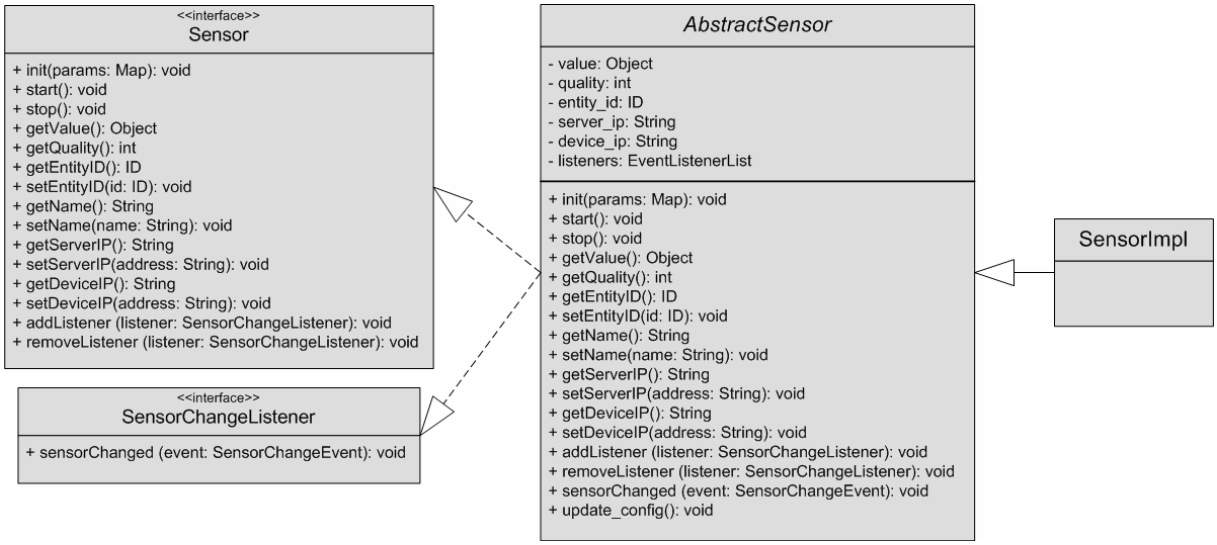


Figure 19 UML Diagram of the Sensor Package

The current implementation of the Context Toolkit already comprises abstract sensor realisations that allow access to information sources such as hardware sensors, web services, databases and explicit user input. Existing concrete sensor implementations include components for position tracking via GPS, WiFi and optical systems (Lorenz et al., 2005), noise and motion detection, light intensity, infrared access detection as well as temporal information such as time or date. In addition, some data retrieval sensors allow access to weather information (like temperature, wind speed, humidity, etc.), Really Simple Syndication (RSS) newsfeeds from the Internet and filtering of newsgroups. Each of these software objects can be extended through the sensor interface. Some sensors deliver more than one sensor value, like for example the weather sensor provides values for the

temperature, wind speed, humidity, etc. The event notification mechanism allows for a nesting of several sensors, and thus, enables a successive transformation of the complex sensor value into several single sensor values.

```
<SENSORS>
     <TrackingSensor CLASS="toolkit.sensor.TrackingSensor"
          SENSOR_SERVER_IP="129.26.166.220"
          DEVICE_IP="129.26.166.184"
          tracked_device_ip="129.26.167.219"
          ENTITY_ID="Carmen"
     />
     <NoiseSensor CLASS="toolkit.sensor.NoiseSensor"
          SENSOR_SERVER_IP="129.26.166.220"
          DEVICE_IP="129.26.167.219"
          ENTITY_ID="Mobile_Phone_7"
     />
     <TimeSensor CLASS="toolkit.sensor.TimeSensor"/>

     ...

</SENSORS>
```

Figure 20 XML Configuration of the Sensors (Excerpt)

The class `SensorFactory` generates instances of local sensors from the markup document "sensors.xml", which Figure 20 shows a section of. Each remote sensor requires a similar configuration document. All sensors obtain the IP address of the device or server that receives the sensor data (omitting the IP address leads to the assumption that "localhost" acts as the default recipient). Furthermore, each sensor gets assigned an entity identifier indicating the entity the sensor belongs to. This identifier enables the context-aware application to keep together all components and data associated with one entity. Moreover, the sensor configuration may contain additional information for the initialization of the sensor.

A light-weight sensor server handles the connection to sensors distributed over a network. The sensor server receives the sensor values and assigns them to correspondent local sensor objects that can be accessed via the sensor interface by other objects of the context-aware application. Besides the sensor value the sensor server extracts other additional information about the quality and status of the sensor. Since the Context Toolkit was designed as a modelling tool, its current implementation does not support the automatic detection or recognition of sensors.

5.2.2 Semantic Layer

The semantic layer of a context-aware application provides all components of the application with an accurate image of each entity's context. It defines the context model of the targeted context-aware application and semantically enriches sensor values. This layer holds all

possible context descriptions of every entity that is relevant and modelled for a particular domain. In addition, the components of the entity context layer allow for a further structuring and organising of the context information associated with one entity in order to package this context information regarding specific characteristics such as mentioned in Section 2.4.

The Context Toolkit supplies developers with a simple but flexible and powerful context representation based on attribute-value pairs. Several advantages turned the balance towards this approach: Efficient representation, transparency, extensibility, easy administration, and an easy integration with third party providers of context information. Additionally, several network transmission protocols exist for this type of representation such as the Attribute-Relation File Format (ARFF) (see ARFF (2007) for documentation and Weka (2007) for more general information about the project). Therefore, a context denotes a set of one or more typed attributes, which associate a name with a value that holds the current context information. Thus, the set

$$C := \{A_1 = a_1 : T_1, A_2 = a_2 : T_2, \ldots, A_n = a_n : T_n\}$$

represents a formalism for a context, with A_i being the names, a_i being the values, and T_i being the types of the context attributes. Currently implemented basic context attribute types include *number* (ideal for measures and other numeric values), *symbol* (ideal for associations and mappings), *Boolean* (ideal for modelling binary information), *string* (ideal for unstructured information), *time* (ideal for temporal information), *set* (ideal for a number of alternatives) and *composite* (ideal for encapsulated information such as coordinates).

Each context attribute is connected with zero or more sensors, and in turn sensors deliver information to one or more attributes, creating a cross-linked network between sensors as information sources and attributes as information interpreters. Context attributes receive sensor values and map them onto attribute-specific values, e.g. the context attribute “daytime” interprets the value of a time sensor. If one attribute listens to several sensors delivering the same type of information, the developer can activate the built-in technology handover functionality: The context attribute prioritises the value of the sensor indicating the highest quality (e.g. switching between GPS or WiFi for position tracking). Furthermore, context attributes can be connected to other context attributes in order to generate semantically more enriched information.

The current implementation provides model-based interpretation of attributes that map sensor data to different abstractions of time (such as time, date, timer, daytime, time schedule and time interval), to locations based on location models (cf. Section 5.3.5) and to certain degrees of noise and motion. Additionally, attributes may derive their values from algorithmic data fusion from more than one sensor (for example, speed is derived from position and time). In most cases the type, name and value of a context attribute are not enough information to build a working context-aware application.

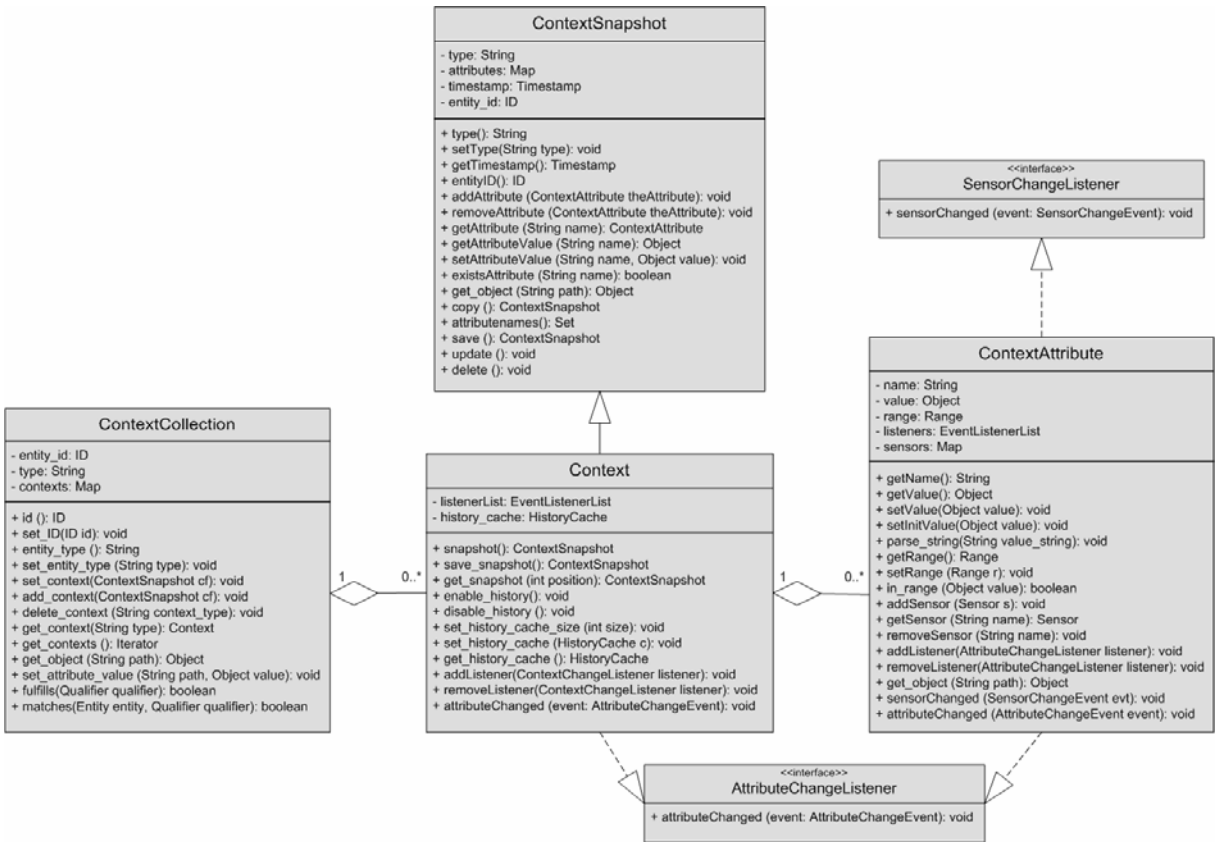


Figure 21 Context Collection, Context and Context Attribute in UML

The Context Toolkit provides a couple of additional properties that further specify a single context attribute and enable a proper management of its values:

Timestamp: This property contains a date-time value describing when the context was sensed. It is needed e.g. to create a context history and deal with sensing conflicts.

Description: A literal description containing details about the context attribute. This property is especially helpful to application developers when new sensors can be dynamically added to the system.

Confidence: The confidence property describes the uncertainty associated with this context attribute value because not every data source delivers accurate information, e.g. location data suffers from inaccuracy dependent on the used tracking tool. Per default, this property equals the quality indicator of each sensor connected to this context attribute.

Value Range: The value range property defines a set of possible or allowed values for a context attribute, e.g. the age of a human lies in between 0 and 150. This value range enables a consistency check of delivered sensors values.

User Preference: The user preference property enables the user of the context-aware application to define a weight (value between 0 and 1) for a specific attribute. This property can correlate with the confidence in the value and affect the decision on whether to rely on low-quality data or not.

```
<user>
     <individuality>
          <ATTRIBUTES>
               <noise/>
               <motion CONTEXT_CHANGE="true" INIT_VALUE="default">
                    <ATTRIBUTES>
                         <spatial.speed/>
                         <individuality.noise/>
                    </ATTRIBUTES>
               <motion/>
          </ATTRIBUTES>
     </individuality>
     <spatial>
          <ATTRIBUTES>
               <timestamp/>
               <position>
                    <SENSORS>
                         <TrackingSensor/>
                    </SENSORS>
               </position>
               <speed>
                    <SENSORS>
                         <TimeSensor/>
                    </SENSORS>
                    <ATTRIBUTES>
                         <position/>
                    </ATTRIBUTES>
               </speed>
               <location CONTEXT_CHANGE="true" INIT_VALUE="general">
                    <ATTRIBUTES>
                         <position/>
                    </ATTRIBUTES>
               </location>
          </ATTRIBUTES>
     </spatial>
</user>
```

Figure 22 XML Specification of a Context Collection

The Context Toolkit hides specific aspects of the context model's complexity from developers and offers functionality for a convenient usage of this model. It allows for the definition and construction of a context collection with reference to a given entity. A context collection comprises several subcontexts of the entire context, which obtain a unique name and summarise a set of context attributes. This subcontext concept enables further structuring of

context information, e.g. into the five categories of context information according to Section 2.4.2, and also a special treatment of these emerging attribute clusters, e.g. monitor the value evolution of the clustered context attribute using the automatic history functionality as described in Section 5.3.1. A context collection requires the specification of the identifier of the entity it belongs to. Either the developer manually assigns this entity identifier or the context collection queries any sensor connected to a subcontext, since all sensors are entity-related and need to possess the respective entity identifier. Figure 21 depicts the interrelations of the main software components of the semantic layer of the Context Toolkit covered by the package `toolkit.semantic.context`.

Figure 22 shows an example of the definition and configuration of a context collection for one type of entity. The structure of the markup document reflects the procedure of setting up a context collection: Generate a subcontext, add context attributes to that subcontext, and connect sensor to the respective context attribute. Typically, the developer defines the subcontexts of the context collection in separate markup documents and refers to the required ones through their names. As a constituent of the context attribute definition section, the expression

```
CONTEXT_CHANGE="true"
```

indicates a context attribute that triggers a context change event if its value changes. The factory class `ContextFactory` realises the instantiation of empty subcontexts from the markup description through a method invocation of the form

```
ContextFactory.new_instance ("spatial")
```

Section 5.3.1 provides an explicit explanation of the configuration of the history functionality. In the same manner as for the subcontexts, a specialised factory class handles the instantiation of context collections for a specific entity type, expecting the name of the entity type as input:

```
ContextCollectionFactory.new_instance ("user")
```

If a developer needs to implement a new context class, this new class needs to inherit from the class `toolkit.semantic.context.AbstractContext`.

5.2.3 Control Layer

The control layer of a context-aware application encodes and realises the behaviour of this application (cf. Section 4.4.3). The software components located in the control layer exploit the information stored in the attributes of the entity context collection provided by the semantic layer. The interplay between a triggering engine, a filtering engine and an execution engine enables and realises the adaptive behaviour or the context-aware application. The triggering engine activates adaptation activities as a response to significant changes in the context of an entity. The filtering engine selects a special subset of a set of entities based on specific shared features. The execution engine assembles a set of commands that realise the selected adaptive behaviour.

The package `toolkit.control` covers the implementation of the control layer in the Context Toolkit. This package offers general-purpose filtering functionality as a programming abstraction, which Section 5.3.3 describes in more detail. However, the core constituent and inherent part of this layer forms a configurable rule system that bases on and extends the general-purpose rule system illustrated in Section 5.3.4. Rule-based systems present a popular way to deterministically express adaptation knowledge, because of their simple implementation. The basis of such systems forms a set of rules of the form "if-then-else". The first part of such a rule is called precondition and the then-part is called *conclusion*. Typically, the preconditions consist of Boolean logical expressions and the conclusion comprises a set of actions. If the precondition of a rule evaluates to "true", the rule fires and triggers the conclusion of the respective rule. The triggering of the conclusion results in the execution of all associated actions.

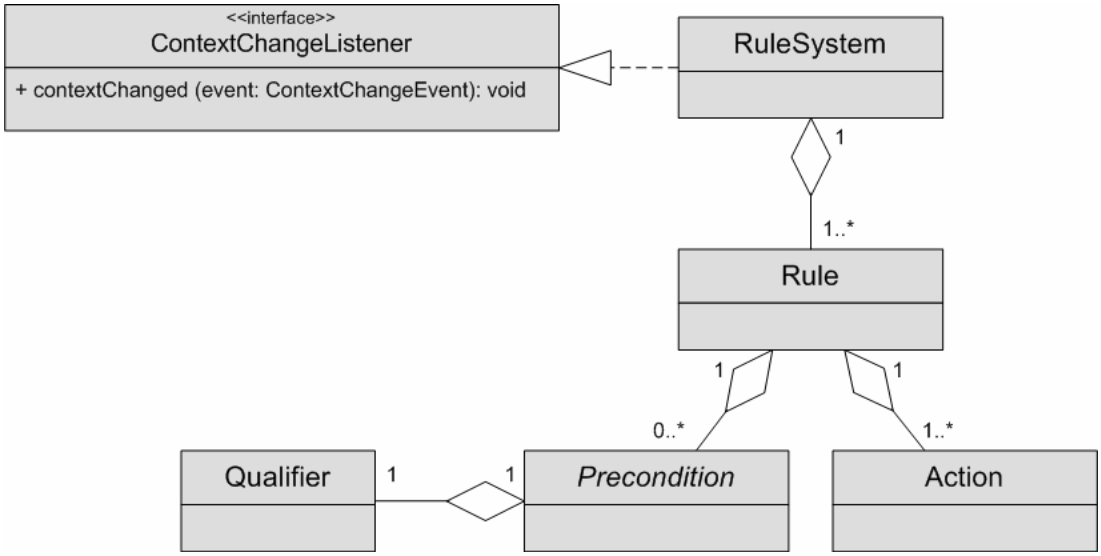


Figure 23 General-Purpose Rule System Displayed in UML

The rule system implemented for the control layer is firmly integrated with the programming framework and hard-wired with the preceding and succeeding layer. This rule system unites triggering and execution functionality and particularly aims at receiving context information from the semantic layer, taking decisions and executing actions realised by the actuator layer. Thus, the rule system determines the behaviour of the entire targeted context-aware application. For a better understanding of the interrelation of the software component located in the control layer, Figure 23 shows an UML diagram of the rule system. Section 5.3.3 provides an introduction to the qualifier programming abstraction used for the definition of the precondition of a rule.

The context model defined for an entity in the semantic layer demands the determination of such context attributes that trigger context change events (cf. Section 5.2.2) if they significantly change their values. Each context change event causes the interpreter of the rule system to sequentially evaluate all preconditions of all rules taking the current context

information provided by the semantic layer as the basis. If this context information fulfils the precondition of a rule, the rule interpreter invokes the set of actuators associated with this rule. The actuators need to be registered at the actuator layer because the rule system references them through their unique names. The sequential order, in which the rule interpreter propagates the context information through the entire rule network, can be interrupted by a special actuator named “break”. Furthermore, the implementation allows for the definition of trivial preconditions for rules that need to be executed per default every time the rule interpreter initiates its evaluation.

```
<CONTROL_RULES>
     <silence>
          <PRECONDITION>
               <AND>
                    <in_meeting/>
                    <EQUALS TYPE="symbol">
                         <REFERENCE ATTRIBUTE="spatial.location"/>
                         <REFERENCE VALUE="executive_office"/>
                    </EQUALS>
               </AND>
          </PRECONDITION>
          <ACTION>
               <set_volume>
                    <PARAMETERS>
                         <volume>
                              <REFERENCE VALUE="0"/>
                         </volume>
                    </PARAMETERS>
               </set_volume>
               <break/>
          </ACTION>
     </silence>
</CONTROL_RULES>
```

Figure 24 One Rule Segment Represented in XML

The rules of a context-aware application need to be constructed dynamically and openly for further changes and improvements in order to deal with restructuring and further differentiation because of new data. In addition, a management of these rules is essential for the verification of consistence. If there is time for permanent (manual) updating and sufficient knowledge available concerning relevant coherences, rule-based systems allow for a simple creation with fast effects. Typically, the behaviour is hard-coded as procedures using the chosen programming language. The non-declarative representation of the reasoning logic in object-oriented programming languages often makes code modification and human inspection difficult.

The configurability of the rule system implemented by the Context Toolkit separates the high-level reasoning logic from the low-level functional implementation. By separating the logic

from the functional implementation, developers or even end-users can modify or replace control layer components without requiring a significant amount of reprogramming efforts. Figure 24 shows a segment of a rule system represented in XML: The rule fires if the mobile phone detects a meeting situation and is located in the executive office (exploiting a relation with a user leads to the user currently taking part in a meeting with her boss). The action executed on this event resets the ringing volume of the mobile phone to zero.

5.2.4 Actuator Layer

In symmetry to the sensor layer as the information source, the "display" of a context-aware application can potentially be the entire environment or at least everything the application can have influence on. Actuators of the application realise changes to the environment and map the decisions taken by the control layer to real-world actions. Furthermore, actuators can be distributed over different devices or embedded in the user's environment. In turn, each device potentially hosts different types of actuators performing several types of changes to variable properties of this device or its environment.

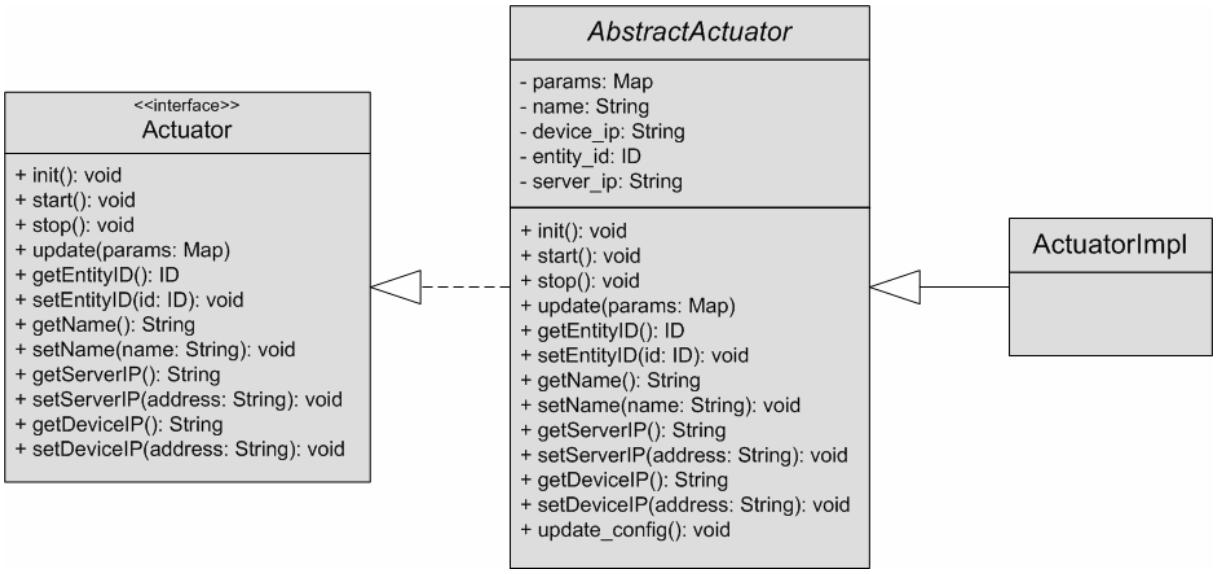


Figure 25 UML Description of the Actuator Package

Through the package `toolkit.actuator` the Context Toolkit offers various specialised software components located in the actuator layer, which process the adaptation method proposed by the control layer. The majority of physical actuators comprise functionality for the presentation of various types of visual content on different types of display panels and device screens. Available actuators include the display of text, images and webpages as well as for video streaming. In addition, the toolkit provides actuators handling the output of audio files and streams. Logical actuators are placed in the application itself and perform tasks such as changing values of context attributes or relations between entities, filtering of context

collections and administrating history entries (cf. Section 5.3) or manipulate entries in the database. These logical actuators enable a flexible management of context information. The available physical and logical actuators allow for the implementation of both active (automatic adaptation through changing behaviour) and passive (presenting updated context information) context-awareness. Figure 25 provides a description of the actuator package of the Context Toolkit, which serves as a basis of future extensions.

```
<ACTUATORS>
      <set_volume
            CLASS="toolkit.actuator.SetVolume"
            ACTUATOR_SERVER_IP="129.26.166.220"
            DEVICE_IP="129.26.167.219"
            ENTITY_ID="Mobile_Phone_7"
            PARAMETERS="volume"
      />
      <set_media_content
            CLASS="toolkit.actuator.SetMediaContent"
            ACTUATOR_SERVER_IP="129.26.166.220"
            DEVICE_IP="129.26.167.211"
            ENTITY_ID="Video_Wall_4"
            PARAMETERS="url, volume, quality"
      />
</ACTUATORS>
```

Figure 26 XML Configuration for the Actuators (Excerpt)

The markup document "actuators.xml" configures local actuators, which are instantiated by the `ActuatorFactory` class. Figure 26 depicts a part of the actuator configuration in XML. All actuators obtain the IP address of the device or computer that operates the actuator and an entity identifier indicating the association with an entity. Furthermore, the configuration contains initialisation information and a definition of invocation parameters. For maximum flexibility of the implementation of actuators, developers can extend the actuator interface and reference their actuator through the specification of the class name in the markup document. Freely programmable actuators facilitate a nesting of actuators and a wrapping of several actuators into a single one, which complies with actuator fission (cf. Section 4.4.4).

Just like for the sensors, a light-weight actuator server handles the connection to actuators distributed over a network. Each active remote actuator needs to register at this server first before the context-aware application gains access to it. The passing of parameters to actuators allows for domain-specific execution of such abstract actuators, and thus, context information can have influence on the entire adaptation process. As a feedback for the control layer, the actuators send back messages indicating the success or failure of their operation.

The stringently sequential execution of the set of actions assembled by the control layer avoids any conflicts among the actuators regarding the access to resources. The predetermined

functionality of the actuator layer requires waiting for a success or failure response of the currently executed actuator before the successive actuator is activated. The current implementation of the Context Toolkit does not support any synchronisation and prioritising of concurrent actuators.

5.3 Using and Configuring the Context Toolkit

The Context Toolkit offers a programming framework and provides libraries of software components that support developers in the implementation of adaptable context-aware applications. In a next step developers of such applications are provided with appropriate programming models and abstractions in order to ease the programming with adaptation. Furthermore, an array of automatisms for recurrent tasks further reduces the implementation effort. The programming techniques presented in this section bundle software constructs of the Context Toolkit in order to simplify the construction of context-aware applications that are flexible and easily customised.

The creation of *adaptable* context-aware applications starts with permitting users and experts control over the internals of the application. Adaptable context-aware applications need to provide mechanisms where application adaptation control can be reconfigured without the need for reimplementation. The authors of context-aware applications can make suggestions on how their work may be used, however the user can make use of the application in ways the author never dreamed of. All programming abstractions presented in this section offer configuration ability through markup documents and thus, the abstraction of the programming constructs allow for a domain-specific instantiation and initialization of software components of the entire application. The following subsections introduce and describe the programming abstractions contained in the Context Toolkit.

5.3.1 Discretisation Abstraction

The temporally diffuse stream of data delivered to the sensor layer may strain the operation of the entire context-aware application because some domains have to deal with highly dynamic alterations of sensor values. The Context Toolkit follows a publish/subscribe event model in order to optimise the event flow of the application, regulate the granularity of change of the value stream, and thus, mitigate the strain. Software components register to the event flow of other software components and are notified about changes of interesting values. Thus, the event model of the Context Toolkit introduces a discretisation of values.

In a first step, the event-based discretisation procedure prunes and cleans the sensor value stream. If the sensor value changes significantly, the sensor fires an event notifying all context attributes that have registered as listeners to this event. The context attribute receiving the new sensor value performs a transformation of this value according to several transformation and interpretation mechanisms (cf. Section 4.2.2). Again the context attribute sends an event

if its value changes, and thus, further reduces the sampling rate of the original value stream. Furthermore, only a dedicated set of selected context attributes notify registered components about changes in the entire context. The control layer receives this context change event and triggers the evaluation of the rule system. Only if the context information fulfils the precondition of a rule, the control layer executes all associated actions and adapts the behaviour of the context-aware application. This dovetail connection of events limits the overall computational complexity of the context-aware application. The following paragraphs introduce two programming abstractions that facilitate the exploitation of the mentioned discretisation procedure.

Context Snapshots

The concept of *context snapshots* continues and realises the approach introduced in Section 5.1.4, which distinguishes continuous and discrete context-aware applications. Since continuous context-aware applications continuously monitor the changing context of the user and create appropriate responses without much delay, some systems are unable to handle the flood of information, and thus, are barely usable. As an example, some tracking systems measure the position of an object in a ten-millisecond interval.

Hence, a technique performing a discretisation of a continuous domain constitutes a valuable programming abstraction. For this purposes the Context Toolkit facilitates discretisation through taking snapshots of a continuous context in discrete time intervals. Like a camera freezes the current environment and captures it on a picture, a context snapshot freezes the current values of all attributes of the context description. Obviously, the granularity of the captured information depends on the frame rate of the snapshots.

The implementation of the context snapshot approach provides the basis implementation of a context as Figure 21 in Section 5.2.2 shows. Each context as part of the entity's context collection implements a method called `snapshot()`, which freezes the values of the current context and returns a context snapshot containing only static values. Thus, the context snapshot approach allows the memorisation of context information in regular intervals as described in the following paragraph or the attachment of context information to content like shown in Section 5.4.2.

History Abstraction

The package `toolkit.semantic.process.history` of the Context Toolkit offers a range of methods of the configuration of the timing and persistence of context snapshots, which can be defined for each type of context separately. Each context of this type obtains a history cache of a variable size that receives and records any taken snapshot in the first place. In addition, a persistence mode specified for each history cache determines the treatment of context snapshots that need to be taken from the cache if the cache size is exceeded. Three persistence modes are available:

- FULL: write the context snapshot into the cache and make it persistent
- ON_OVERFLOW: make the oldest snapshot persistent if the cache sizes is exceeded
- NONE: store the context snapshots in the cache only and delete the oldest snapshot deleted if the cache size is exceeded.

The history functionality provided by the Context Toolkit allows for a determination of triggers that automatically invoke the two-step process of taking a context snapshot and putting it into the history cache. Three possible triggers enclose synchronous time intervals, events of changing context attributes and explicit requests. Figure 27 shows an example of the configuration of a history cache for a context type using XML. The example configuration causes the history cache for the context type "spatial" to automatically take a context snapshot in a two-second interval and store it in a five-element history cache for further processing. If the cache overflows, the oldest context snapshot is stored in a database.

```
<spatial>
     <PERSISTENCE
          ADAPTER="toolkit.persistence.adapter.database.DatabaseAdapter"
     />
     <CACHE
          SIZE="5"
     />
     <HISTORY
          PERSISTENCE_MODE="ON_OVERFLOW"
          EVENT_TYPE="SYNCHRONOUS"
          TICK="2000"
     />
     <ATTRIBUTES>
          ....
     </ATTRIBUTES>
</spatial>
```

Figure 27 History Cache Configuration in XML

The history abstraction forms a part of the process sublayer of the semantic layer. This programming abstraction supports the profiling task of personalization engines and enables an analysis of the user's behaviour and the derivation of preferences, interests, and so forth. Moreover, the history programming abstraction comprises a persistence adapter, which abstracts from the three storage media database, file system and active memory. This persistence adapter can be exchanged like a plug-in and allows for a management of context information in the three mentioned types of storage media.

5.3.2 Reference Abstraction

A reference is an identifier which refers to data stored somewhere else, as opposed to containing the data itself. References constitute a fundamental construct for many data structures and facilitate exchanging information between different parts of an application. The access to a reference implies the access to the data this reference points to. The data itself need not be moved, and thus, references increase the flexibility in terms of where objects or values can be stored and how they are allocated. They also make sharing of data between different parts of the application easier if each part keeps a reference to it.

The package `toolkit.semantic.context.reference` of the Context Toolkit manages references to various types of objects as part of a context-aware application: entities, context attributes, memory segments and static values. The references either return a handle of the respective object or the actual value. Besides the manual instantiation of ready-made reference classes in the application source code, the configurability of references and their textual representation play a major role within the markup documents of the design view. In the following the usage of the configuration for the four mentioned reference types is explained in more detail.

Entity references point to context collections associated with one specific entity. This reference type requires the declaration of the targeted entity type in combination with the unique identifier associated with the actual entity. For example:

```
<REFERENCE ENTITY="user.Carmen"/>
```

Context attribute references extend the entity references and refer to handles or values of context attributes of a specific entity context. Since each entity possesses a collection of context, the targeted context type needs to be specified first. After the context type the position in the history cache of the respective context can be specified (or omitted, default is the current context). This position points to the desired context snapshot. The context attribute reference returns a handle to the respective context attribute, and depending on the utilisation of this reference the current value of this context attribute needs to be accessed in a succeeding step. The following example specifies a reference to the context attribute "daytime" of the temporal context of the user with the identifier "37839" and returns the second position of the history cache:

```
<REFERENCE ATTRIBUTE="user.37839.temporal.2.daytime"/>
```

Memory references allow for the storing and loading of temporary values of variables. The scope of application of such a variable depends on the context of the evaluation and can be *local*, *global* or *system*. The variable itself is referenced by its name. In the same manner as for the context attribute reference, the memory reference returns a handle to the respective variable. For example:

```
<REFERENCE MEMORY="system.temp"/>
```

Value references make fixed values accessible. The Context Toolkit tries to convert the type of the value into the type required by the accessing object. This type conversion might be necessary for the comparison of two values of different types. The configuration value reference is very straightforward, for example

```
<REFERENCE VALUE="25"/>
```

The use of names as parameters for the reference requires the assignment of meaningful names for the referenced components. As the system grows, some names might not be unique anymore. Therefore, the reference abstraction relies on the creation of well-structured and cascaded names. The following paragraphs depict the application and the advantage of references. In particular, the programming abstraction for the facilitation of matching and filtering build upon the four described reference types.

5.3.3 Matching

The ability of matching context information constitutes an integral part of context-aware computing because matching procedures can be useful for many different purposes. The first matching procedure takes into account the contexts of two entities in order to determine the similarity between them and derive possible relations between the two entities, which allows for a further exploitation (e.g. users in the same room). Second, a matching procedure can check whether one entity's context fulfils certain criteria (e.g. for the determination of a specific situation such as "in meeting"). A third matching procedure enables the filtering of entities from a set based on certain similar characteristics their contexts share (e.g. filter five paintings from the exhibition that match the user's interests).

The Context Toolkit supports all three types of matching procedures and offers the package `toolkit.semantic.context.matching` for the implementation and configuration of matching functionality. This library facilitates the assembly of qualifiers (Boolean qualifiers and similarity measures) that make the matching of two context attributes, two subcontexts and two context collections possible. Furthermore, these qualifiers represent the basic construct for filtering context collections from a set and for triggering based on context information. In addition, the qualifiers continue functioning in circumstances, where the value of a context attribute is unknown (e.g. due to a sensor failure), and remain unaffected by appending the context model with additional context attributes. The following paragraphs explain the qualifier abstractions Boolean qualifier and similarity measure in more detail, and introduce the configurable filtering functionality of the Context Toolkit.

Boolean Qualifiers

Boolean qualifiers allow the comparison of entity contexts using Boolean expressions. The evaluation of a Boolean qualifier results in either the value "true" or "false". The Context Toolkit represents a Boolean qualifier in a tree, which is generated from the following constituents:

- *References* to static or context attribute values, or variables (leafs),
- *Predicates* as relational operators for the referenced values (inner nodes),
- *Conjunctions* or *disjunctions* as concatenations of these constituents and
- *Negations* for the inversion of the evaluation.

The references compose leafs of the Boolean qualifier tree. As inner nodes of this tree, predicates concatenate exactly two references. In turn, conjunctions and disjunctions are inner tree nodes, as well, and concatenate two or more predicates. Figure 28 illustrates the interrelations and the structure of the Boolean qualifier programming abstraction in more detail.

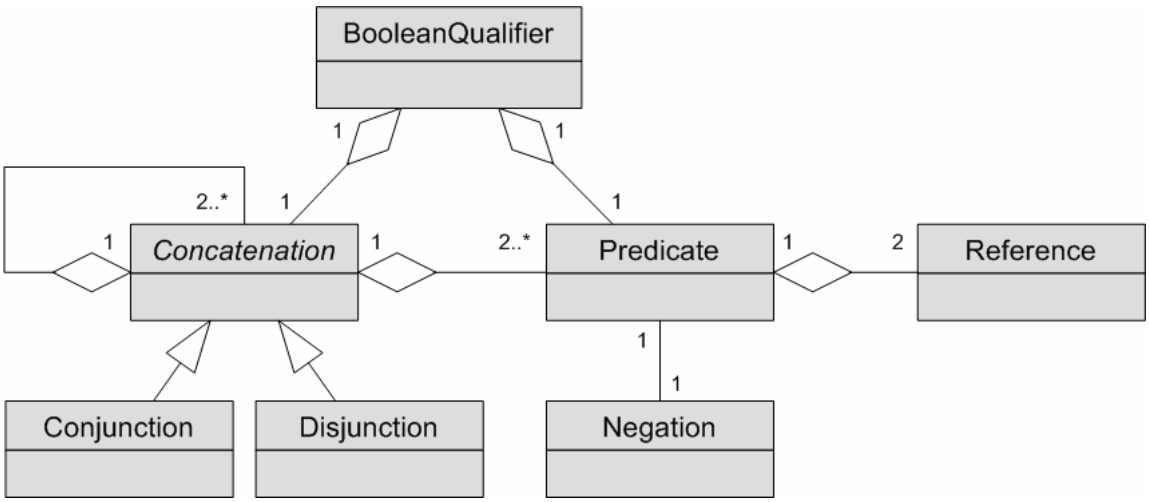


Figure 28 UML diagram of the Boolean Qualifier Programming Abstraction

The Context Toolkit already covers implementations of predicates for the following context attribute types: symbol, number, Boolean and time. Orthogonally to these types the implemented predicates cover a range of typical operators: `EQUALS`, `GREATER`, `LESS`, `GREATER_OR_EQUAL`, `LESS_OR_EQUAL`, `IS_LIKE`, `NOT_EQUAL`, `CONTAINS`, `CASE_SENSITIVE_LIKE` or `IS_NULL`. Since the predicates and concatenations are Boolean expressions, their evaluation results in “true” or “false”, whereas negations of these elements return the inverted result. Boolean qualifiers expect two context collections as an input.

The matching abstraction provided by the Context Toolkit resolves the values needed for the evaluation of the Boolean qualifiers through references as described above. In addition, the toolkit library enables a definition of reusable and universally applicable Boolean qualifiers for matching, which take any context collection of an entity as an input. Therefore, a special context attribute reference needs to be introduced, which exchanges the entity reference at its beginning with the keyword “query”.

```
<BOOLEAN_QUALIFIERS>
     <quiet>
          <LESS TYPE="NUMBER">
               <REFERENCE ATTRIBUTE="query.individuality.noise"/>
               <REFERENCE VALUE="24"/>
          </LESS>
     </quiet>
     <loud>
          <GREATER TYPE="NUMBER">
               <REFERENCE ATTRIBUTE="query.individuality.noise"/>
               <REFERENCE VALUE="73"/>
          </GREATER>
     </loud>
     <normal>
          <AND>
               <quiet INVERT="true"/>
               <loud INVERT="true"/>
          </AND>
     </normal>
</BOOLEAN_QUALIFIERS>
```

Figure 29 Qualifier Configuration in XML

Figure 29 exemplifies three Boolean qualifiers for the determination of noise levels ("quiet", "normal" and "loud") represented in XML, which shows the usage of the keyword "query". The markup document "boolean_qualifiers.xml" contains a list of configurable Boolean qualifiers needed by the targeted context-aware application. The static class `BooleanQualifierManager` manages all the Boolean qualifiers listed in this document and controls the access to them through the specification of names for each Boolean qualifier. In addition, this class allows for the registering of newly defined Boolean qualifiers.

Similarity Measures

For many context-aware applications the expression of value range by means of predicates suffices. In some cases, however, a more differentiated statement than simply "true" or "false" is desired for the comparison of two contexts. Similarity Measures return values in the interval [0, 1], which indicate the degree of semantic proximity of two objects. The value "0" expresses the minimal and accordingly the value "1" the maximal similarity. The notion of similarity rests either on exact or on approximate repetitions of patterns in the compared objects. The similarity or dissimilarity between two objects constitutes a valuable source of information about the two involved objects.

The package `toolkit.semantic.context.matching.similarity` of the Context Toolkit defines the framework of the composition of similarity measures and provides a set of default implementations of similarity measures between two context collections. A configuration of similarity measures is not possible, yet. Typically, the calculation of the

similarity between two context collections bases on the weighted accumulation of the similarity of their context attributes. The similarity measure between two context attributes is called “local similarity measure” and the one accumulating these local similarity values is called “global similarity measure”.

Local similarity measures might become very complex, and therefore, the toolkit library offers a set of ready-to-use comparison metrics for context attribute types such as symbols, numbers, and time. Newly added local similarity measures need to inherit from the class `AbstractLocalSimilarityMeasure`, which implements the interface `LocalSimilarityMeasure` and expects two context attributes as input. Extensional global similarity measures inherit from the class `AbstractSimilarityMeasure` implementing the interface `SimilarityMeasure`, which defines two context collections as input.

Filtering

A filter produces a subset of a given set on the basis of specific properties all elements of the subset have in common. In context-aware computing the filtering of contexts or context collections from a set constitutes an important task as the two examples show: For the notification of all people in room number “133”, the context-aware application needs to determine exactly this group of users, whose contexts exhibit certain similar properties, i.e. room number “133”. For the recommendation of exhibits of a museum that might be of interest to the user, the context-aware application needs to find a subset taken from all exhibits, which fit best to the user’s interests.

For such purposes the Context Toolkit offers a filtering engine that operates on an arbitrary list of context collections and returns elements that fulfil certain criteria. For the filtering process qualifiers and similarity measures deliver the required filer criteria as a basis. For more flexibility the filtering engine expects zero or one context collection as an input and as a source of values for the matching process. If a context collection is specified, its values can be referenced by the “query” keyword as described above.

The markup document “filters.xml” contains a set of filters that can be referenced by their names throughout the context-aware application. Figure 30 depicts an example configuration of a filter that retrieves all songs from a music repository that match the user’s mood and possess a length below three minutes. The static class `FilterManager` administrates the corresponding markup document and supplies instances of the configured filters. The filtering process results in a list of zero or more context collections that fulfil the filter criteria. If similarity measures serve as a basis of the filtering procedure, the result is an ordered list with the best matching context collection in the first position. However, the current implementation of the Context Toolkit only allows for an application of similarity measures in filters that operate on the active memory. Filtering context collections from the database requires the loading of all elements into the active memory first.

```
<FILTERS>
     <short_song_filter>
          <AND>
               <LESS TYPE="number">
                    <REFERENCE
                         ATTRIBUTE="song.individuality.length"
                    />
                    <REFERENCE VALUE="180"/>
               </LESS>
               <EQUALS TYPE="symbol">
                    <REFERENCE
                         ATTRIBUTE="song.individuality.character"
                    />
                    <REFERENCE
                         ATTRIBUTE="user.individuality.mood"
                    />
               </EQUALS>
          </AND>
     </short_song_filter>
</FILTERS>
```

Figure 30 Filter Configuration in XML

5.3.4 Triggering Abstraction

The triggering abstraction provided by the Context Toolkit supports an event-driven and asynchronous programming style because actions are invoked as a response to context changes. The implemented triggering mechanism follows the event-condition-action model (Dayal et al., 1988), in which each trigger includes a precondition on the invocation of the specified action that is evaluated upon detection of the event. Typically, developers of context-aware applications make extensive use of this model in order to realise functionality such as pro-activity.

The Context Toolkit package `toolkit.control.eventtrigger` forms the foundation of the implementation of triggering functionality with the class `AbstractTriggeringEngine`, which developers can extend for their own purposes. This abstract class implements the event-condition-action model and also allows for its configuration through a markup document. Developers create a set of rules, each consisting of a qualifier and a corresponding action, which are kept in a triggering repository. The triggering engine instance accounts for detecting the occurrence of significant context change events of one context collection and then invoking actions in accordance with the rules as an appropriate response to this event.

Besides manually implemented extensions of the abstract triggering engine, the Context Toolkit already offers two specialised extensions, which facilitate the programming and application of situations and rule systems. The general principle of the triggering functionality

remains the same, only the type of the invoked actions changes. The following paragraphs describe the two special-purpose triggering engines in more detail.

Situation Abstraction

The situation abstraction enables the specification of context classes by placing constraints on relevant context attributes (Section 2.3.3 provides a definition of the term *situation*). The situation abstraction resides above the components that represent context information in the context model of an entity and enables the definition of context classes through logic operations of certain context attribute values. A situation can be understood as a description of the states of context attributes, and thus, the situation abstraction allows developers of context-aware applications to interact with the context of an entity as a whole instead of querying and subscribing to several components individually. This mechanism allows for an automatic initiation of actions based on certain characteristics of the context parameters.

```
<SITUATIONS>
     <in_meeting
          TRIGGERS="user.spatial.location"
     />
          <AND>
               <LESS TYPE="number">
                    <REFERENCE ATTRIBUTE="user.spatial.speed"/>
                    <REFERENCE VALUE="0.03"/>
               </LESS>
               <EQUALS TYPE="symbol">
                    <REFERENCE ATTRIBUTE="user.spatial.location”/>
                    <REFERENCE VALUE="meeting_room"/>
               </EQUALS>
               <GREATER TYPE="number">
                    <REFERENCE ATTRIBUTE="user.individuality.noise”/>
                    <REFERENCE VALUE="42"/>
               </GREATER>
          </AND>
     </in_meeting>
</SITUATIONS>
```

Figure 31 Situation Configuration in XML

The Context Toolkit package `toolkit.semantic.context.situation` realises the situation programming abstraction. A situation is said to hold in a given context exactly when this context satisfies certain properties, and thus, a situation is associated with the context collection of an entity. The triplet expressing a situation consists of a list of context attribute names and a list of qualifier-symbol pairs. The list of context attribute names determines which change events the triggering engine of the situation programming abstraction needs to register for. The symbols simply mark the possible names or identifiers for the situation. The main part of the situation definition comprises the specification of the qualifiers and their

association with each of the symbols. The name-based referencing of qualifiers (see Section 5.3.3) allows for the reuse and combination of predefined qualifiers using concatenation operators in order to form richer situations.

Developers can either create instances of the class `Situation` manually or through the static `SituationFabric` class. This fabric operates on markup documents such as shown in Figure 31, which defines the example situation "in_meeting". Then, each situation instance needs to be associated with the context collection of an entity, which causes the situation trigger to register and listen to context attribute change events in accordance with the context attribute names specified in each situation description. Each time the triggering engine receives such a context change event, the situation qualifiers are sequentially evaluated on the basis of the currently available context information provided by the context collection. If the context information fulfils a situation qualifier, the situation instance adopts the name of the respective symbol and sends out a change event. The current implementation only supports one name per situation instance.

Rule System Abstraction

A context-aware application may contain several rule systems associated with and operating on the context collection of any entity. The Context Toolkit package `toolkit.control.rulesystem` includes a rule system abstraction that bases on the triggering functionality provided by the toolkit and establishes the basis of the definition of control rules in the control layer (cf. Section 5.2.3). Each rule system comprises a set of rules, which in turn consists of a list of context attribute names, a qualifier and a list of class names. Just as described for the situation abstraction, the list of context attribute names determines the change events, which the triggering engine of the rule system needs to register for. The qualifier forms the precondition of the rule, which serve as a trigger condition that needs to be fully satisfied by the current context. The class names specified in the conclusions part of the rule refer to programmed classes that implement the `Action` interface.

An instance of the class `RuleSystem` establishes a container for a set of rules, which can be assembled manually or automatically parsed from a corresponding markup document. The developer then needs to associate this rule system instance with one context collection. The instance registers to the change events of the specified context attributes and initiates an evaluation of the qualifiers of all rules if it receives such an event. For each rule that fires, the rule system instance instantiates all classes of the conclusion part and executes the associated source code. Leaving the precondition part of a rule empty (i.e. a trivial precondition), the conditions part of the rule is executed with every invocation of the rule system. The processing of all rules happens in a sequential order.

As shown in Figure 24, the parameters can be assigned to the classes described in the conclusion's part of a rule, which influence the initialisation of the instantiated class. Furthermore, the rule system abstraction makes use of the memory reference as described in Section 5.3.2. This reference enables the definition and assignment of variables that facilitate

the buffering of values emerging from the execution of actions. The scope of a variable can be *local* for one rule only, *global* for the entire rule system or *system*-wide. The rule system abstraction offers a comprehensive functional implementation, which allows easy modifications through its configurability.

5.3.5 Special Purpose Context Attributes

The programming abstractions presented so far form the structure and the basis of an adaptable context-aware application. In order to further ease the programming with adaptation, the components in the package `toolkit.semantic.context.attribute` of the Context Toolkit comprise basic as well as special purpose context attributes that can be utilised for the simplified generation of semantically enriched information.

Complementary to the history functionality of the Context Toolkit, the context attribute `InterestAttribute` enables the observation and examination of the user's interaction with other entities of the domain. The `StereotypeAttribute` context attribute enables the definition of user classes that allow for a fast classification of new users to the application according to their initial behaviour. The context attributes `ZoneAttribute` and `ScheduleAttribute` process overlay models for the context information space and time, and supports the interpretation of this context information through imposing a structure on the value range of the context attribute. The following paragraphs explain these special purpose context attributes in more detail, and provide examples for their configuration through markup documents.

Interests

The access to the interests of a user enables adaptive systems to better meet the user's information needs. Interests describe such information needs and allow for a filtering of information that is relevant to the user. The description of interests always relates to elements of the domain, like for example paintings or sports, and therefore, a domain-independent description of interests is hard to achieve.

The Context Toolkit offers an interest model associated with a special purpose context attribute, which works with an additive extension and assessments of interests in topics. The package `toolkit.semantic.process` contains the class `InterestModel`, which forms the basis of the context attribute `InterestAttribute`. The interest model represents interests according to the two-level tree structure as illustrated in Figure 32. Each node of this structure obtains an integer value v_{ii} indicating the frequency of occurrence of the topic. The occurrence frequency value of the inner nodes of the tree structure equals the accumulated occurrence frequency values of their children.

The interest model processes an input in the form of a topic-subtopic pair, which causes the model to refine its tree structure accordingly. If the topic-subtopic pair is missing in the structure, it is inserted with an initial occurrence frequency (i.e. the value "1") of the subtopic

leaf node. If the topic-subtopic pair is already available in the tree structure, the value of the occurrence frequency of the subtopic is incremented by one and the occurrence value of the associated topic is adapted accordingly. Thus, the interest can be modelled and displayed in a transparent manner in contrast to concepts such as neuronal networks as inaccessible black boxes.

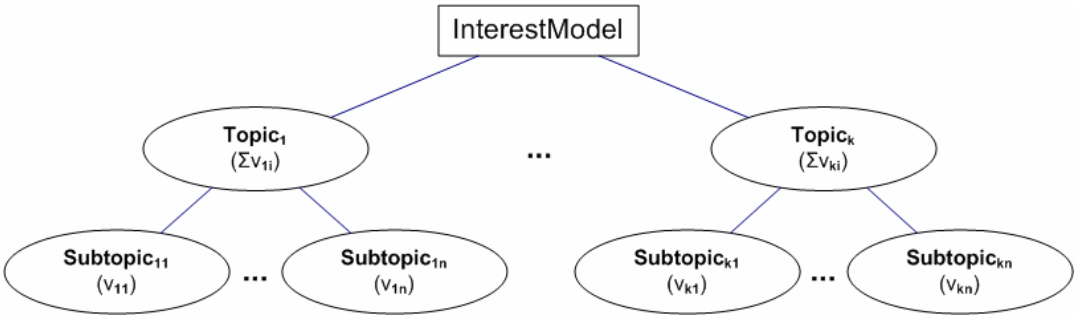


Figure 32 Two-Level Tree Structure of the Interest Model

Such a score-based interest model enables the derivation of the user's interests in a certain object from the properties this object exhibits and from the feedback this user implicitly or explicitly provides for this specific object. This procedure requires all objects to be described according to their properties in a set of topic-subtopic pairs, which involves considerable effort for the investigation of these properties. Every time the user provides positive feedback for a specific object, the topic-subtopic pairs associated with this object build up and further refine the interest model. Since this model reflects the user's demands for information, the interests can be exploited in the filtering process for information that is relevant for the user.

Stereotypes

The stereotype approach was introduced to the user modelling community by Rich (1989) and illustrates a popular technique of the fast acquisition of user models. A certain amount of basis information about the user is required when conclusions on information about the user are to be drawn. If users can be classified according to specific shared characteristics, the definition and application of stereotypes is reasonable. They allow for the initialisation of certain context attributes of new users to a context-aware application with values that are typical for the respective class of users. Thus, stereotypes enable adaptive applications to work with (preliminary) information and draw assumptions about the user right after the beginning of the interaction. The preparation and assignment of stereotypes splits up into three essential steps (Kobsa, 1993):

- *Identification of User Groups*: Homogeneous subsets of the totality of users need to be identified regarding application-relevant properties.
- *Identification of Key Characteristics*: Distinctive attributes of the users need to be identified, in order to enable the assignment of new users to the subsets.

- *Representation in Stereotypes*: The application-relevant attributes of the users need to be formally modelled in a convenient representation scheme.

The stereotype context attribute of Context Toolkit facilitates the representation of stereotypes through the specification of interrelations between context attributes with Boolean qualifiers. This special purpose context attribute continuously triggers the evaluation of the Boolean qualifiers and obtains the value of the stereotype, whose qualifier evaluates to "true". Figure 33 depicts definition of the stereotype "motion", which describes various classes of motion behaviour.

```
<motion>
     <NO_MOVEMENT>
          <LESS TYPE="number">
               <REFERENCE ATTRIBUTE="user.spatial.speed"/>
               <REFERENCE VALUE="0.03"/>
          </LESS>
     </NO_MOVEMENT>
     <SLEEPING EXTENDS="NO_MOVEMENT">
          <LESS TYPE="number">
               <REFERENCE ATTRIBUTE="user.individuality.noise"/>
               <REFERENCE VALUE="10"/>
          </LESS>
     </SLEEPING>
     <WALKING>
          <GREATER TYPE="number">
               <REFERENCE ATTRIBUTE="user.spatial.speed"/>
               <REFERENCE VALUE="0.1"/>
          </GREATER>
     </WALKING>
</motion>
```

Figure 33 Stereotype Definition in XML

A stereotype of one user subgroup stands for the totality of all represented characteristics of this user subgroup. If the qualifier of one stereotype forms a subset of the qualifier of another stereotype, hierarchies of stereotypes can be constructed. The root of such a hierarchy typically constitutes a default stereotype, which represents the user group for standard or new users, who just started the interaction. The further progress of the user behaviour determines a further specialising or generalising of the stereotype.

Overlay Models

An overlay model reproduces the structure of an underlying model like for example a domain model. The overlay model contains specific information for every element of the underlying model and a reference to this element. Through this referencing the navigation through the domain model is equivalent of the navigation through the overlay model. Therefore, the

exploitation of the information represented by an overlay model allows for a further specification of the user's context information.

```
<SEGMENTATION>
     <meeting_room TYPE="toolkit.semantic.model.spacemodel.SquareZone">
          <X1_COORDINATE VALUE="33.0"/>
          <Y1_COORDINATE VALUE="433.0"/>
          <X_EXPANSION VALUE="30.0"/>
          <Y_EXPANSION VALUE="57.0"/>
     </meeting_room>
     <executive_office TYPE="toolkit.semantic.model.spacemodel.SquareZone">
          <X1_COORDINATE VALUE="100.0"/>
          <Y1_COORDINATE VALUE="433.0"/>
          <X_EXPANSION VALUE="30.0"/>
          <Y_EXPANSION VALUE="57.0"/>
     </executive_office>
</SEGMENTATION>
```

Figure 34 Space Segmentation as an Overlay Model Specified in XML

```
<TIME_SEGMENTATION CLASS="toolkit.semantic.model.timemodel.Schedule">
      <morning HOUR="6" MINUTE="0"/>
      <noon HOUR="11" MINUTE="0"/>
      <afternoon HOUR="13" MINUTE="0"/>
      <evening HOUR="18" MINUTE="0"/>
      <night HOUR="22" MINUTE="0"/>
</TIME_SEGMENTATION>
```

Figure 35 Time Periods as an Overlay Model Defined in XML

The package `toolkit.semantic.context.model` of the Context Toolkit offers two freely configurable overlay models for space and time. The models `Segmentation` and `Schedule` allow for a structuring of space and time, and enable a mapping of position information onto disjunctive zones and a mapping of temporal information to time intervals. Figure 34 shows an excerpt for such a structuring for the realisation of a floor plan of an office building. The subsequent Figure 35 depicts a partition of the daytime into the five sections morning, noon, afternoon, evening and night.

5.4 The Tool Suite

The advantage of discrete context-aware applications is that they can all be encompassed by a single general mechanism: an author creates content in an appropriate format, attaches a specific key situation to this content, a triggering mechanism retrieves this content when its key situation matches the user's present situation, and a mobile device presents the retrieved

content to the user. This type of context-aware application rather demands creative than programming skills from an author or even an end-user. A tool suite provided by the Context Management System assists authors and end-users without much programming experience in the creation of such discrete context-aware applications.

The resulting facilitation of the development process accompanies a set of simplifications and assumptions that authors and end-users need to consider for this specific type of discrete context-aware application. The first simplification affects the equitation of the user's context and the context of the device the user carries. Since the user and her device share nearly the same spatial context, this type of discrete context-aware application only regards the context of the user's device. The basic task of the operational discrete context-aware application constitutes in matching the device's (i.e. the user's) present situation with the context snapshot, and thus, with the associated content supplied by the author of the application. The second simplification restricts the number of entities to the user's device and content entities. This restriction allows the application of a standard matching procedure, which filters contents from a set that suit the context of the user's device. The triggering engine continually initiates this filtering procedure in a three second interval, and triggers the user's device to display those contents that match the current situation of the device and the user.

The tool suite builds upon the libraries and the configuration abilities of the Context Toolkit, which developers and experts make use of, and supplies user interfaces, which assist authors and end-users in the generation of this specific type of context-aware application. The Design Tool exploits the configurable parts of the Context Toolkit and offers means of modifying the design view on the targeted context-aware applications (cf. Section 5.2 and 5.3). The Mobile Collector facilitates the connection of arbitrary content to a context snapshot in the actual context of use. After a discrete piece of information is selected from a content management system, the Mobile Collector can take a snapshot of the current context and attach this snapshot to the selected content. The Content Player retrieves content when the user enters the situation described by the context snapshot attached to a specific content.

Section 6.1.8 provides an illustration of the application of each of the tools in the realisation of a context-aware museum guide. The following subsections describe each tool in more detail.

5.4.1 The Design Tool

The mode of operation of the software components provided by the Context Toolkit and the entire design of the targeted context-aware application are to a certain extent configurable using the markup language XML. The Design Tool provides an appropriate design and authoring interface, which enables the modification of the design view and the saving of changes. The tool constitutes a regular editor for XML files augmented with some intelligence that is intended to prevent users from making mistakes. This interface puts together panels that facilitate the design of the application. Panels are available for the administration of

sensors, actuators, context attributes, context models, control rules, and qualifiers. In the following, the functionality of these panels is described in more detail.

The SENSORS and the ACTUATORS panel enable its user to control the sensing and actuation capabilities of the context-aware application. With these two panels the user can add new, update existing, and remove unused components. The system already offers a range of sensors to the user, which can be reused in different applications. Each sensor can be referenced by a unique name, which is used as an identifier throughout the application. Distributed sensors, to which the application has access only over a network, have to be configured with respect to their IP address, ports, and communication protocol (e.g. TCP/IP or HTTP). In the same manner, the user can select actuators that realise the behaviour of the context-aware application in a real-world environment. Again, each actuator needs to obtain a unique name as an application-wide reference. In addition, an IP address needs to be specific that determines the device to be addressed. The current implementation of the Design Tool only supports the displaying of content on various devices. However, the diversity of supported formats of the content, like images, sounds, text or Hypertext Markup Language (HTML), enlarges the range of potentially creatable context-aware applications with this tool. All sensors and actuators need to obtain a unique identifier of the entity they belong to.

After the user reconsidered possible effects of her context-aware application and which indicators might be relevant to trigger these effects, the composition of the targeted application proceeds with the creation of a context model of a user of the future operational application. First, the ATTRIBUTES panel allows for the addition and removal of context attributes, which are again identified by unique names. Context attributes can be assigned to the user's context model using their names as a reference in the next step and then referenced within the rule system in order to access current context attribute values.

At this stage, all basic components of the context-aware application are defined and ready to use for the modelling step. The MODELLING panel facilitates the design and administration of any entity's context model used throughout the application and allows for the definition of several types of context for each entity of the domain. Furthermore, the panel supports the allocation of sensors to context attributes. Figure 36 shows an example of the allocation of sensor named "TimeSensor" to the context attribute "daytime". In the same manner, a context attribute may be allocated to another context attribute in order to feature nesting. In addition, the MODELLING panel offers means of the configuration of history caches for each type of context.

The Design Tool automatically generates a database scheme according to the context model designed with the MODELLING panel. This functionality translates the markup contained by the XML configuration file into Structured Query Language (SQL) statements that create an accordant database scheme. This database scheme allows for the persistent storage of context information used for example by the history functionality or by the Mobile Collector for the creation of persistent links between a content identifier and a context snapshot.

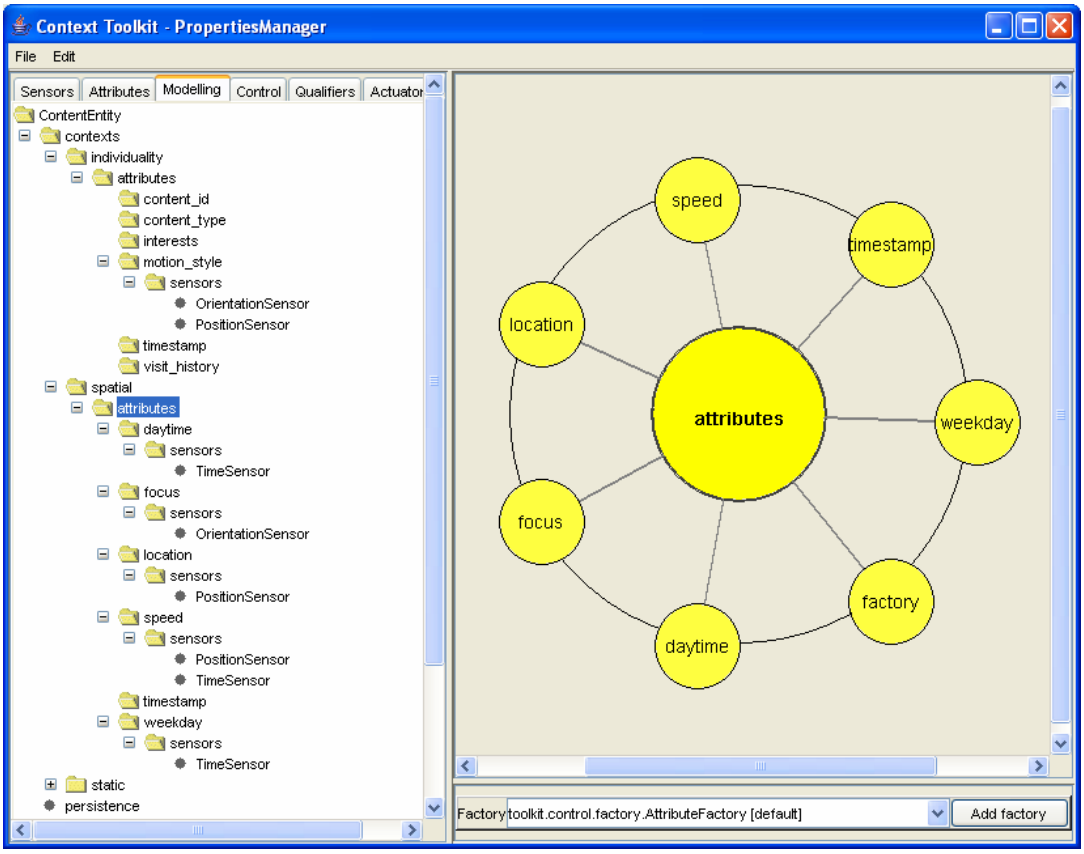


Figure 36 Screenshot of the Modelling Panel of the Design Tool

The QUALIFIERS panel allows for the preparation of qualifiers that serve as preconditions for the rules controlling the application behaviour, which need to be defined in the subsequent step. Therefore, this panel offers a tree-like representation for the creation of such conditions as described in Section 5.3.3, in which the inner node represents the operation and two child nodes the respective operands.

With the user interface provided by the CONTROL panel, the user specifies aspects of the application's behaviour. One or more qualifiers defined earlier are conjoint into a precondition for a rule and integrated with a list of actions, which are executed if the precondition is fulfilled. Both the names of queries and the names of actuators used for the definition of a rule, refer to the components defined in earlier steps. The functionality behind the CONTROL panel registers the generated rules at a general-purpose triggering engine.

During the operation of the system, this triggering engine continuously analyses the current situation of the user, evaluates the preconditions of the rules and triggers the execution of the actions if the preconditions evaluate to “true”.

5.4.2 The Mobile Collector

The annotation of content with context information for a subsequent filtering is a special task which can most effectively be done directly in the context of use. The Context Management System supports authors of context-aware applications with a tool for recording context information and linking it to arbitrary content from a content management system. The Mobile Collector represents an efficient tool that enables authors to produce contextualised content. Via wireless LAN, this tool accesses and displays real-time context information from the Context Toolkit and at the same time content from a repository. Since the Mobile Collector runs on a light-weight Tablet PC, the author can move it around, take a snapshot of the current context directly in the context of use and attach it to the selected content.

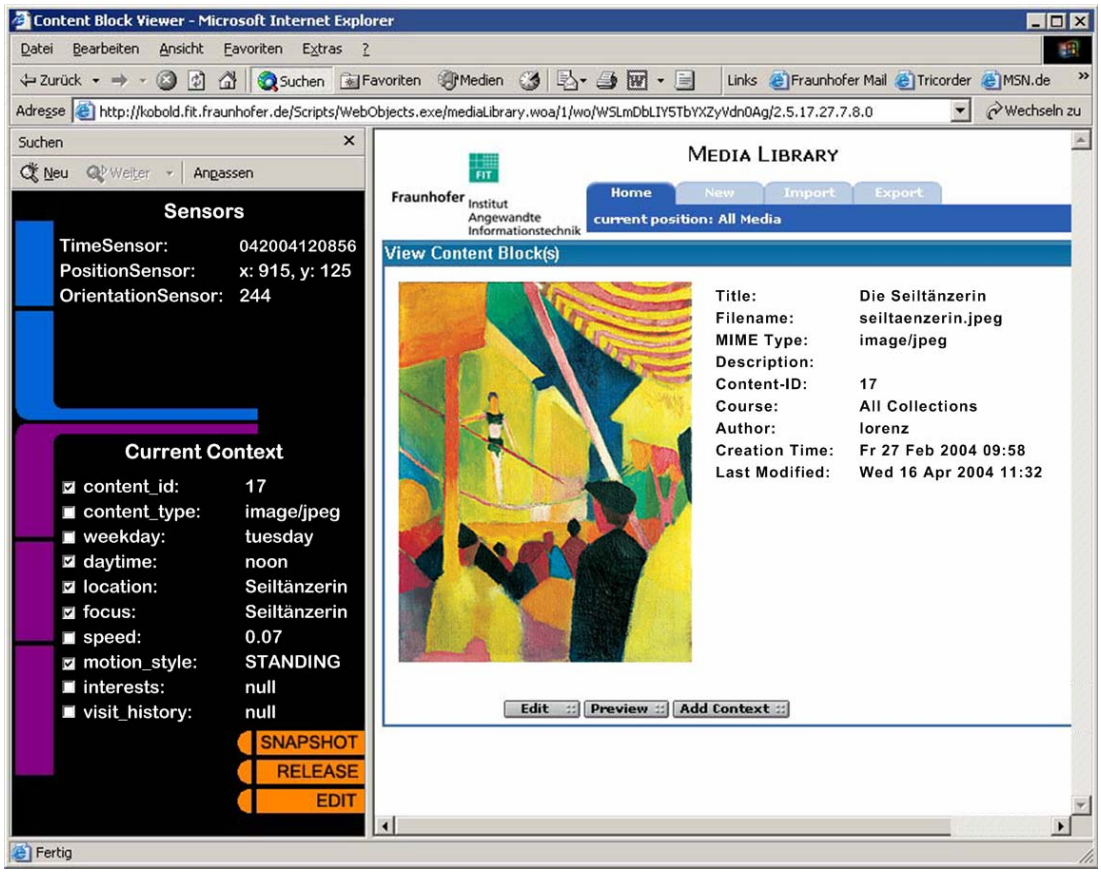


Figure 37 Screenshot of the Mobile Collector Running on a Tablet PC

Figure 37 illustrates the Mobile Collector during operation. The right-hand side shows the web front-end of the content management system integrated with the Context Management

System. This screen provides the user with functionality for adding, removing, searching, and browsing content such as images, sounds, videos, or even entire HTML pages. The left-hand side of the figure depicts current context information in the lower panel and the current sensor values in the upper panel. Sensors directly connected to the device send their values to the Context Toolkit sensor server first, before they are displayed in the Mobile Collector. If any changes occur to the values of the sensors or context attributes, both panels are immediately updated. Since the left-hand panel represents a specific plug-in to a standard browser, the displaying of context information does not affect the user while she is browsing the content or even the Internet.

The lower left panel of the Mobile Collector user interface reflects the context models implemented with the Context Toolkit or designed with the Design Tool for each domain entity. This user interface allows the switching between the entities and their respective context model. The Mobile Collector is able to display various data types of context information covering Integer, Real, String; Boolean, and Point. If an unknown data type occurs, the Mobile Collector falls back to the `toString()` method each context attribute has to implement for displaying the respective value.

Furthermore and most importantly, the Mobile Collector enables the user to capture the current context, i.e. create a context snapshot. The user can then edit and change the frozen context attribute values. In addition, the user can (un)select attributes that are considered (ir)relevant in this specific situation through (un)checking the check-box for that attribute. This process establishes a precise copy of the current situation that causes the subsequent filtering process to be less restrictive.

After the user has selected the correct content and adjusted the context appropriately, she can create the link between these two by clicking on the “snapshot” button. The context snapshot consists of current (potentially edited) values for each context attribute at the specific moment at which the button was pressed. The Mobile Collector then triggers the Context Toolkit to create a database entry consisting of the identifier or uniform resource locater of the selected content enriched with the value vector of the context snapshot. Since the Mobile Collector is a tool for collecting context snapshots and linking them to appropriate contents, it is not appropriate for administrative purposes. Breaking up these links between contexts and contents, i.e. removing the context annotation for a specific content, requires further treatment using an administrator tool on a desktop PC like for example the MySQL Control Center (MySQLCC, 2007).

5.4.3 The Content Player

The Content Player constitutes the operative part of the finalised context-aware application. End-users of this context-aware application carry around this device and automatically receive content suitable and adapted to the current context. The Content Player presents various types of content that the author of the application has been “depositing” in specific situations using the Mobile Collector in a preceding step. The usage of this tool provided by the Context

Management System does not necessarily and exclusively limited to the operational use by the end-user, but also represents a valuable debugging tool for authors and designers of the targeted context-aware application.

The Content Player comprises an adapted web browser running on a mobile device like a Personal Digital Assistant (PDA) as Figure 38 shows. Just like the Mobile Collector, the Content Player reads out the sensors connected to the mobile device and sends their values to the Context Toolkit sensor server. Based on these and other values, which other sensors connected to the application provide, the Context Toolkit derives the values for the context model that has been implemented or designed earlier.

Figure 38 Screenshot of the Content Player Running on a PDA

The Context Toolkit interprets the current context of the Content Player, and context change events trigger the determination of the behaviour of this device. In this case, the determination of the behaviour involves the displaying of a content selected from the content management system as a function of the current context attribute values. For the selection of the most suitable content a filtering procedure compares the value vectors of the context snapshots stored in the database with the current attribute values of the current device context and returns the best match. The content identifiers associated with the best matching context snapshots give information about which contents needs to be retrieved from the content management system.

As a response to the context change event the Context Toolkit sends one or more composed uniform resource locators back to the Content Player. Then, the Content Player exploits these locators and retrieves the contents from the content management system. After all contents have been downloaded, the Content Player refreshes the webpage displayed by the browser and fills it with the new contents. At the same time the browser treats content according to its type and displays images, plays back sounds or visualises videos.

5.5 Summary

This chapter introduced and described a Context Management System that aims at facilitating the development and maintenance of adaptable context-aware applications. This Context Management System addresses the demands for support of all actors identified in the preceding chapter and comprises functionality for the construction, integration, authoring, administration and tailoring of context-aware behaviour.

In the preface of the description of the constituents of the Context Management System, Section 5.1 discussed general design considerations that affected the character of this system. This section identified five design aspects and illustrated how these aspects are reflected by the Context Management System.

Section 5.2 described the core constituent of the Context Management System, the Context Toolkit that offers a software framework and a library of ready-to-use software components. The Context Toolkit implements the layered software architecture presented in Chapter 4 and guides developers during the software engineering process of context-aware applications.

For each layer, the context toolkit offers a separate package of Java software components that hide complex technical details from the developer. Section 5.2.1 and Section 5.2.4 illustrated the current implementation of the sensor layer, which comprises concrete sensor implementations accessing several information sources, and the actuator layer, which includes actuator implementations presenting various content types. Section 5.2.2 introduced the context representation realised by the semantic layer of the Context Toolkit, which bases on attribute-value pairs and supplies developers with a simple but flexible and powerful representation means. A set of context attributes are embraced by a subcontext, and in turn, a set of subcontexts is enclosed by a context collection, which is associated with exactly one domain entity. Section 5.2.3 explicated the rule system implemented for the control layer, which is firmly integrated with the preceding and succeeding layer and determines the behaviour of the entire targeted context-aware application.

In order to further ease the programming with adaptation for developers, Section 5.3 introduced appropriate programming models and abstractions. The programming techniques presented in this section bundle software constructs of the Context Toolkit in order to simplify the construction of context-aware applications.

Section 5.3.1 described the context snapshot abstraction as a means of discretisation of continuously changing context values and managing past and the history abstraction for creating and managing historical context information. The reference abstraction facilitates the information exchange between the software components and the architecture layers of a context-aware application.

Section 5.3.2 introduced reference abstractions for entities, context attributes, memory segments and static values. The matching abstraction addressed in Section 5.3.3 supports the assembly of Boolean qualifiers and similarity measures of the matching of two context attributes, two subcontexts and two context collections.

The programming abstractions mentioned so far form the basis of the more complex triggering programming abstraction that implements the event-condition-action model. Section 5.3.4 described this triggering programming abstraction as well as two specialised extensions, which facilitate the programming and application of situations and rule systems.

In order to further ease the programming with adaptation, the Context Toolkit comprises special purpose context attributes for the simplified generation of semantically enriched information. Section 5.3.5 characterised the three context attributes *interests*, *stereotype* and *overlay*, that support the interpretation of context information.

The Context Toolkit provides the basis of adaptable context-aware applications through an accordant design view, which permits end-users and experts the control over the internals of the application. Adaptable context-aware applications need to provide mechanisms where application adaptation control can be reconfigured without the need for reimplementation. In conjunction with the implementation details, Section 5.2 and Section 5.3 provided illustrations of how to use the configuration ability of the Context Toolkit. The described markup documents allow for a domain-specific instantiation of the abstract implementation of programming constructs and flexible and easy customisation of software components.

The last section of this chapter detailed the tool suite that encompasses the Context Toolkit and the content management system. This tool suite builds upon the libraries and the configuration abilities of the Context Toolkit and supplies user interfaces that stronger include authors and end-users in the generation process of discrete context-aware applications. Section 5.4.1 explicated the Design Tool that enables modifications to the design view of the targeted context-aware application, i.e. the configurability of the Context Toolkit. The Mobile Collector (cf. Section 5.4.2) facilitates the creation of contextualised content and associates arbitrary content to a snapshot taken in the actual context of use. Section 5.4.3 introduced the Content Player as a tool that serves as a display of the operational context-aware application and retrieves contextualised content from the applications repository that suits its current context.

An investigation of the concepts, the software libraries, the configuration ability and the tool suite needs to prove the general applicability of the presented Context Management System. The following chapter documents the application of the Context Management System for two case studies: a museum guide and an intelligent advertisement board.

Chapter 6

Case Studies

The Context Management System described by Chapter 5 aims at facilitating the realisation of adaptable context-aware applications and at addressing the demands of developers, domain experts, authors and end-users in their specific roles within the accordant software engineering process. This chapter presents two case studies that serve as a proof of the validity of the investigations and findings achieved by this thesis. The Context Management System has successfully provided the fundament for the design, realisation and deployment of several context-aware applications from various domains: a museum guide, an intelligent advertisement board, a pro-active assistance system for warehouse workers, and a treasure hunting game. The museum guide and an intelligent advertisement stand out from the set of example applications because they cover the most comprehensive application of the Context Management System and the concepts behind. The museum guide emerged from the combined efforts of partners from research and industry jointly working on a triennial research project that was funded by the European Commission. The intelligent advertisement board represents a project conducted by the Fraunhofer Institute for Applied Information Technology that presented this context-aware application at the trade fair CeBIT 2004.

These case studies represent operational and adaptable context-aware applications that sense their environments, construct a model of the user's context, and adapt their behaviour according to changes in the context. They generated high demands on the recurrent adaptations of their context-aware behaviour and allowed for an evaluation of the Context Management System in practical use. The implementations demonstrate the universal applicability and validation of the conceptual framework and the software architecture introduced in Chapter 4. Furthermore, the case studies show the application of the Context Toolkit and its programming abstractions during the implementation. In addition, the evaluation conducted for developers, domain experts and authors proves the utility of the tool suite for these actors. The museum guide additionally enabled the evaluation with end-users and therefore, encompasses the core part of this chapter. A concluding summary depicts the advantages and drawbacks of the approach proposed by this thesis.

6.1 Museum Guide

One core vision of ubiquitous computing (Weiser, 1994) is that items of our daily environments acquire computational capacities (e.g. sensing data in the context, processing it and adapting the system to the context of interaction), provide new functionalities and enable new activities. The physical reality will then be overlaid by an additional virtual layer (Mixed Reality). Naturally moving in space and/or manipulating physical objects in our surroundings will be mapped to information browsing, handling and manipulating activities in the virtual layer. This enables a new augmented user experience and forms the basis of novel applications in ubiquitous computing.

The LISTEN project (LISTEN, 2006) conducted by the Fraunhofer Institute for Media Communication is an attempt to make use of the inherent integration of aural and visual perception, developing a tailored, immersive audio-augmented environment (Eckel, 2001). The LISTEN system extends everyday environments with interactive soundscapes. The user of the system is provided with an interactive access to personalised and position-dependent acoustic information spaces while she naturally explores her everyday environments. While using the LISTEN system, the user automatically navigates a dynamic virtual auditory scene designed as a complement or extension of the real space she is exploring.

The user of the LISTEN system moves in physical space wearing motion-tracked wireless headphones that are able to render three-dimensional sound. This sophisticated auditory rendering process takes into account the user's current position and head orientation in order to seamlessly integrate the virtual scene with the real one. Speech, music and sound effects are dynamically arranged to form an individualised and situated soundscape offering information related to visual objects as well as creating context-specific atmospheres. The user listens to audio sequences emitted by sound sources virtually placed in the environment. These sound sources design soundscapes that create situations of perception similar to everyday hearing experiences.

The LISTEN project enables the composition of sound, body and space through a tight correlation of the three perception aspects: self-perception, visual and tactile space perception, and the acoustic space perception. On the one hand, virtual soundscapes are produced for the user by means of virtual sound sources that emerge at specific points in space with different acoustics that can be perambulated. On the other hand, the person's position within space related to other objects is known, which allows for a response to the listener's behaviour in space. The system knows where in space the user moves, where she has been, how fast she is, and what she has heard, and reacts to these parameters in the composition.

The LISTEN project has defined and investigated a new form of multi-sensory content in order to increase the perceptual, emotional and pedagogical effect of a variety of applications ranging from exhibitions to marketing and entertainment events. Everyday environments like exhibition halls, storage areas, public places or living rooms might become interfaces that enable intuitive and non-intrusive access to a three-dimensional auditory information

environment. In the course of the project several prototypes and a virtual-reality-based authoring tool have been developed. One prototype has been evaluated in a public exhibition called the Macke Laboratory, which is described in the next section.

6.1.1 The Macke Laboratory

In October 2003, a LISTEN environment was set up for a representative selection of paintings by the artist August Macke (1887-1914) at the Kunstmuseum Bonn (Unnützer, 2001). The *Macke Laboratory* integrates Macke's paintings into a differentiated acoustic space, in which spoken texts and sounds reflect and supplement the visual experience in an associative manner. The visitors to the museum experience personalised audio information about exhibits and in parallel navigate through a complex data structure.

Figure 39 A Visitor of the Macke Laboratory

The innovative auditory features of the LISTEN system make different perspectives on the artworks by August Macke accessible: different real and fictitious reviewers, art historians and restorers speak to and approach the visitor, and bring alive selected places and situations in Macke's creative activity. A variety of linguistic levels are marked by different room acoustics that accompany the quotations with sound collages. Thus, the acoustic and spatial productions merge: The combination of paintings, sound events, and the architecture of the exhibition hall spans a topography of the contents, which outlines locations and situations of Macke's life.

Challenges

The special requirements of a museum guide offer a valuable proving ground for the concepts of the LISTEN project. Museums and exhibitions have already been explored as domains in several research projects. Many of them focus on the goal to provide a museum guide including content concerning the artworks (Oppermann and Specht, 1999), to immerse the user in a virtual augmented environment built in virtual museums (Chitarro et al., 2003), or to provide orientation and routing functionalities (Ciavarella and Paterno, 2003). In addition, the interaction methodologies of museum visitors with the mobile guide vary significantly. Recent approaches such as the HyperAudio System (Petrelli and Not, 2006) rely on a pen and a palmtop computer and the ec(h)o system described by Wakkary and Hatala (2006) depends on a wooden cube for the selection of audio presentations. Regarding the paradigm shift in Ubiquitous Computing, where technology becomes invisible to the user, the visitors of the museum audio-augmented by the LISTEN system are neither forced to carry any devices in their hands nor interact with the system explicitly.

The challenge of the Macke Laboratory was to provide a personalised immersive augmented environment, which goes beyond the guiding purpose, and is based on the combination of aural and visual perception. The concept of the audiovisual environment picks up on the pedagogically proved experience with audio guides and expands this conventional form of informative guiding through a museum with elements of the radio feature and the surround sound installation. Existing problems of traditional audio guides like “querying” each exhibit, the inconvenience of carrying the device and the co-listening in crowds, can be solved by the installation of the LISTEN system into such an environment.

The visitor moves freely in this audio-augmented environment and without explicit interaction with the technology, which steps back for the benefit of the aesthetic experience. The visitor is not limited to a stream of presentations, but can move through different soundscapes and stay in one place as long as wanted. What the visitor hears depends on her movements through the space and the direction of her gaze at any given moment. If the visitor approaches an exhibit, the attached acoustic information and simulation of a moving sound source is activated: the visual experience is virtually and acoustically accompanied, and the visitor can control the intensity and extent of the acoustic information by head movements, approaching and turning away from the exhibit.

In a further stage of extension, the Macke Laboratory investigated an additional option: The integration of microphones into the exterior of the headphones enabled the reproduction of the real space, i.e. the talking of other visitors of the exhibition and surrounding noise. The microphone signals are mixed with the radio-transmitted signals within the headphones. This technology allows for the communication with other viewers and consequently a joint experience of the virtual soundscape. However, this special option did not come into operation at the Kunstmuseum Bonn.

For the museum installation the LISTEN system constructs multilayer presentations on an individual basis and offers the possibility to turn off disturbing repetitions, and to start or stop

audio elements at appropriate points, which results in a more detailed control of the presentation. Additionally, the system allows for an adaptation of the presentation to the user's interests expressed by her behaviour and enables the provision of recommendations. The curator of a museum obtains a high degree of freedom in the design of the interaction (as a response to the movements and the behaviour of the visitor), the selection and composition of the content of the sound, and the design of the presentation of the content within a three-dimensional soundscape. The LISTEN system offers the visitor of a museum an augmented and deepened perception of the exhibition hall and the artworks.

Requirements and Contextualisation

The combination of high-definition spatial audio rendering technology with advanced context modelling methods creates audio-augmented environments. Rather than a predetermined, pre-recorded audio programme, the Macke Laboratory needs to offer listeners a personalised audio environment, based on their interaction within the real space. The enhanced audio format has to provide layers of information with increasing levels of involvement. It will allow the visitor to find her own level of engagement with an exhibition. The varying depth of experience gives each visitor the chance to find her own level or area of comfort and interest.

The Macke Laboratory needs to exhibit an adaptive behaviour that selects, presents and adapts the information considering the visitor's goals, preferences, knowledge, and interests, which requires the system to take into account the visitor's current context. Traditionally, the acquisition of a user model is driven by monitoring the user's activities and explicit interactions with the user interface. However, one of the main objectives of the LISTEN project was to avoid any portable device or remote control for the visitor except for the headphones and the movement in physical space. For the contextualisation process, this means that only implicit feedback is available in any LISTEN application and the visitor's body is the only interface for interaction.

As a difference to traditional audio guides, the audio-augmented environment deployed for the Macke Laboratory needs to provide an intelligent memory in order to reduce repetitive loops in the audio presentation. The system needs to register the repetition of an action and react immediately with offering other sound entities and new audio sources. The contextualisation process needs to detect and carefully apply redundancies in the presentation of audio information. Therefore, the context model has to keep track of each visitor's visit history and the system then adapts the presented information with respect to what the visitor has already experienced. In addition, the analysis of the visitor's behaviour should allow for the provision of a personalised tour through the exhibition.

The adaptive behaviour of the LISTEN installation for the Macke Laboratory has to create enhanced, interactive and intelligent soundscapes tailored to the context of the individual visitor. The following section illustrates the instantiation of the general framework of context-aware applications for this specific installation of the LISTEN system.

6.1.2 The Software Architecture of the Macke Laboratory

The creation of an immersive audio-augmented environment requires the combination of context-aware behaviour with a virtual-reality-based world modelling as well as authoring and simulation techniques. For a museum visitor, the LISTEN system basically consists of the wireless headphone with two attached antennas, which enable the localisation of the person wearing these headphones. In addition, the visitor recognises a laptop placed in the entrance area, with which the museum's staff is able to reset the environment for each visitor. However, several computer systems required for the operation of the LISTEN system are hidden from the visitor. Figure 40 illustrates the hardware architecture installed for the Macke Laboratory.

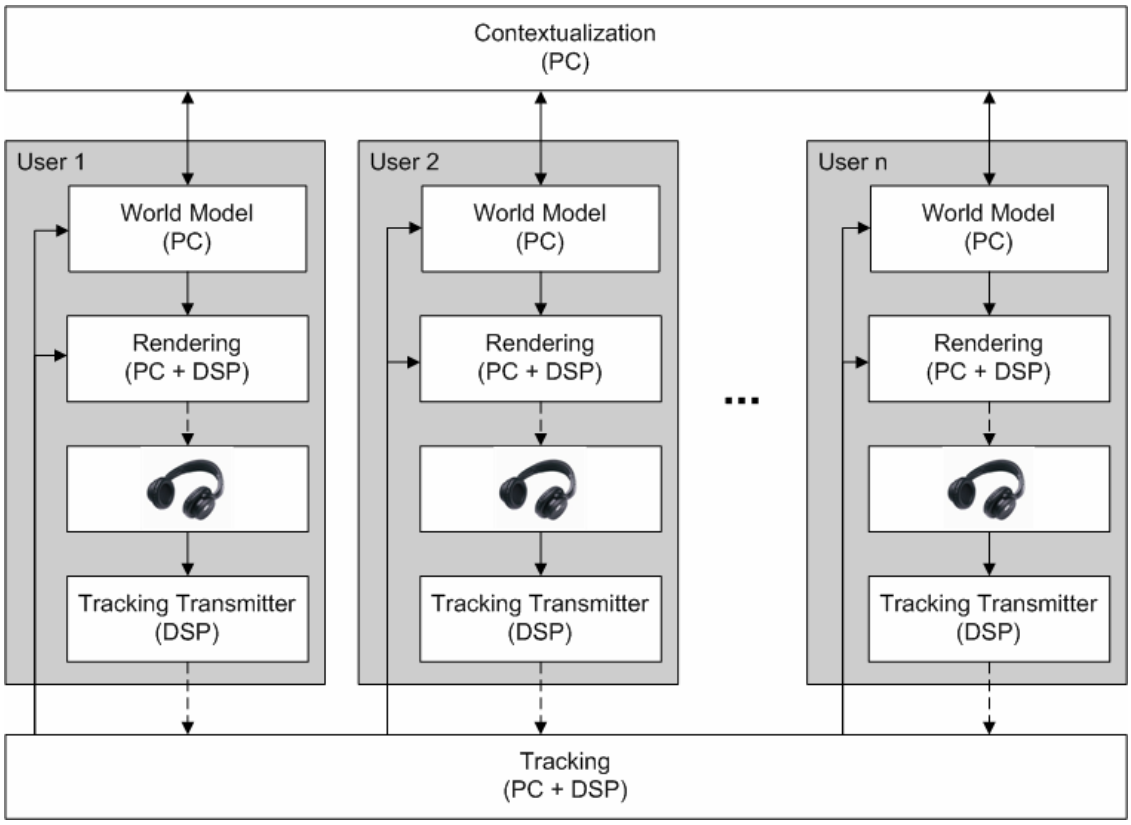


Figure 40 Hardware Architecture of the LISTEN Installation for the Macke Laboratory

One localisation computer determines the current positions and the head orientation of all visitors within the exhibition hall. The localisation computer sends this information to the two computers that attend to one single person: a Linux computer, which interprets the localisation data based on the world model, and a Macintosh computer, which generates the three-dimensional sounds for the visitor's headphones. A virtual reality environment running on the Linux machine contains a model of the real exhibition hall, which enables the identification of significant behaviour of the visitor in space like looking at an exhibit or staying in a certain area.

The Macintosh computer controls the combination of single sound elements and serves the virtual sound sources surrounding the visitor with audio data. This computer transmits the audio data to the headphones of the visitor it is associated with. A digital signal processor (DSP) receives the transmitted audio signal and renders it into three-dimensional surround sound that the visitor is able to hear. In addition, each headphone is equipped with a tracking DSP that transmits the tracking signal to the localisation computer via the two antennas attached to the headphones.

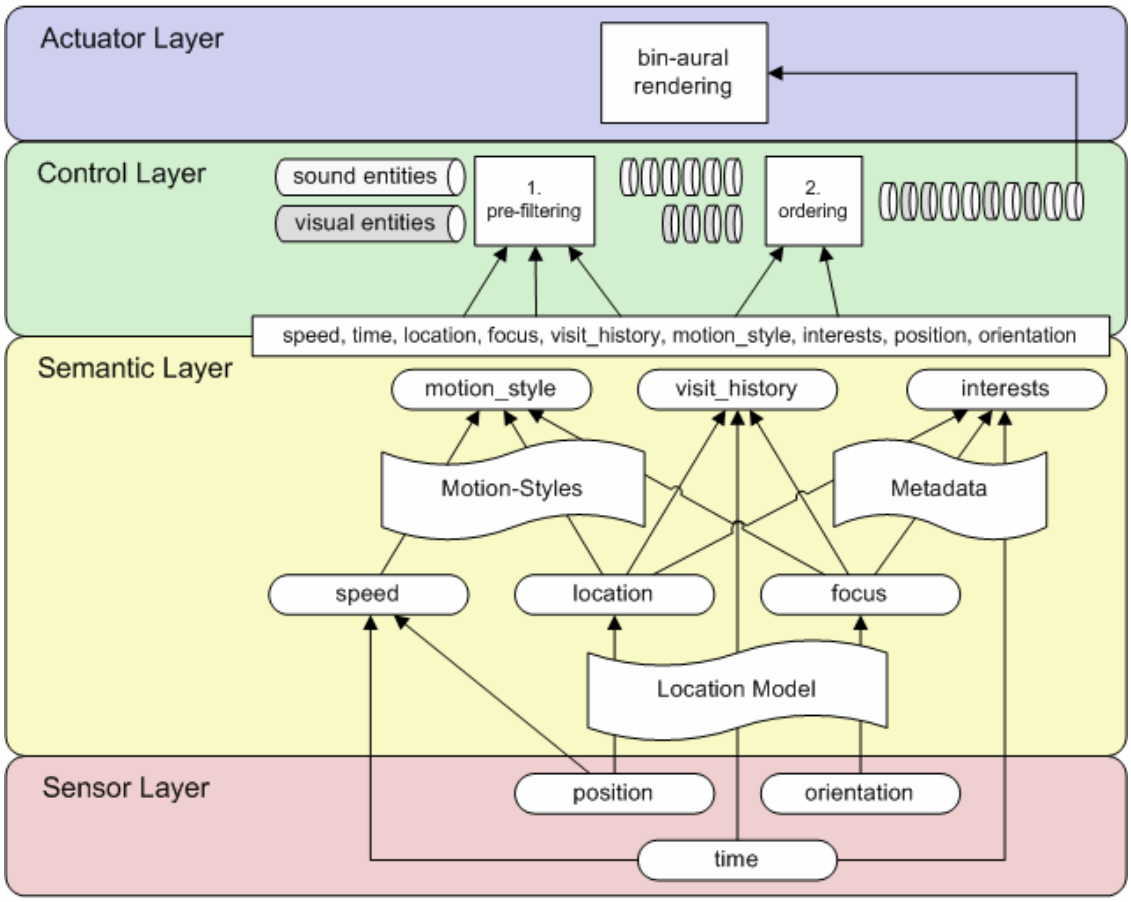


Figure 41 Instantiation of the Software Architecture for the Macke Laboratory

The information provided by the virtual reality environment serves as a basis of the contextualisation of the audio presentation to the visitor's situation at a given moment. A centralised contextualisation server for all visitors assures the adaptive combination of the appropriate content and the current context of the visitor, which makes the LISTEN installation a context-aware application. While the visitor moves in physical space, events are sent to the contextualisation server, which refines the context model of each visitor. The context model contains knowledge about the visitor and influences the adaptation of the audio-augmented environment to the visitor's individual behaviour and interaction. The

contextualisation server provides the virtual reality environment of each visitor with recommendations of an individual adaptation of the audio scene.

Figure 41 provides an overview of the instantiation of the abstract architecture for context-aware applications proposed in Chapter 4 for this specific installation of the LISTEN system at the Kunstmuseum Bonn. This figure illustrates the distribution of the basis component of the application on the four layers of the architecture. The following subsections describe the instantiation of the architecture in more detail.

6.1.3 The Tracking System

The sensor layer of the architecture depicted in Figure 41 comprises the sensor components "time", "position" and "orientation". The "time" sensor acquires the current time from the system clock. The sensors "position" and "orientation" correspond to the visitor's position in the museum and the orientation of her head. In order to acquire these sensor values, a high-quality tracking system is necessary to detect the spatial structures of the environment. The requirements on the tracking technology arise from the goal to completely immerse the user into a convincing virtual acoustic scene (Bregman, 1994): Continuous low-latency tracking of the position of the user's head and its orientation is necessary covering the entire area to be augmented. A tracking system for people in virtual reality situations requires high accuracy in the spatial resolution down to the ranges of centimetres as well as high access rates up to 30 Hz in order to create an immersive environment. The total system latency, i.e. time interval between the head motion and the adjustment of the sound presentation, must be below 59 ms (Wenzel, 1998). Additionally, the tracking system needs to be installed in an indoor environment and shall be robust.

In contrast to concepts where the object to be tracked is receiving information, like for example the Global Position System (GPS), the user of the LISTEN system carries the wireless navigation transmitter. The navigation transmitter, mounted upon the user's headphones, enables the system to determine the user's position and head orientation. Since the transmit antenna is mounted to the user it needs to be as small sized, low weighted and concerning the power consumption as efficient as possible. As a LISTEN project partner, the Institute of Industrial Electronics and Material Science (IEMW), now Institute of Sensor and Actuator Systems (ISAS, 2007), accounts for the development of a tracking system that complies to this requirements.

For the LISTEN system a time-of-arrival tracking concept using short radio frequency burst signals was selected (Goiser, 1998). The navigation receivers receive the navigation signals transmitted from the device attached to the user's headphones. The navigation receivers need to be arranged and set up in a favourable geometry and each receiver obtains a known fixed location. A central signal-processing unit accumulates the data collected by the network of receivers and calculates the absolute position and the orientation of each navigation transmitter. The position is determined by geometric triangulation based on the relative time-of-arrival differences between the receivers. For the measurement of the orientation each

transmitter is equipped with two antennas arranged in a defined distance to each other. This assembly generates a small delay that enables the calculation of the angle the user's head displays compared to a fixed landmark.

The determination of the position in (x, y)-coordinates and the orientation angle requires at least four receivers with direct line-of-sight contact to the transmitters. In order to gain appropriate accuracy and reliability and to cover the whole space of a large room like in the Kunstmuseum Bonn (15m x 15m x 5m) eight receivers need to be deployed[1]. The tracking system automatically selects the most favourable device from the array of receivers for the calculation of the position and orientation. Non-line-of-sight links as well as interfering multi-path signals are discarded by plausibility checks concerning the strength and quality of the received signal and the calculated position. For a proper system operation the assumption must be satisfied that in any situation at least four line-of-sight links are available. With this setup the tracking system is able to measure the positions of eight users simultaneously with an absolute accuracy of the head position of about 10 cm. The minimum granularity of the orientation angle comes to 5 degrees. The battery live time of the navigation transmitters lasts at least one hour.

6.1.4 Modelling the Domain

A central issue in context modelling concerns the structuring of the information domain because a system that presents individualised media and creates augmented environments relies on a detailed description of the domain. An information engineer, who designs such information structure, needs to understand the domain from the perspective of the visitor. In particular, the design of personalised information services demands the intuitiveness of the information structure for the successful application of contextualisation and personalisation methods.

For the Macke Laboratory the LISTEN system dynamically arranges speech, music and sound effects to form an individualised and situated soundscape offering exhibit-related information as well as creating context-specific atmospheres. The adaptation process exploits an extensive amount of annotations for the artworks respectively for the different sound elements associated with each exhibit. Therefore, the domain model of the system holds information and metadata about all presented sound and visual entities. In addition, the domain model builds up a virtual acoustic space, which attaches the sound and visual entities to the physical space. The following two paragraphs describe the location model and the content model of the LISTEN installation in more detail.

---

[1] Due to the long period of supply for the planned tracking system, the Macke Laboratory was equipped with a different tracking system by Advanced Realtime Tracking GmbH (see ARTracking (2007) for more details). This system uses infrared cameras that recognise the target mounted to the headphones and exhibits similar accuracy compared to the planned tracking system.

Location Model

The LISTEN application observes the visitor's spatial position and head orientation through a tracking system (cf. Section 6.1.3) and interprets the acquired information based on a virtual environment connecting the real-world objects with virtual objects. Since the museum's visitor is mobile in physical space, her spatial positions is translated into virtual positions in the electronic space relative to virtual objects. The virtual environment enables the definition of virtual sound sources and the segmentation of the physical space into virtual zones.

The *world model* describes the physical environment the visitor moves through while interacting with the system. In the LISTEN environment, this model contains the detailed geometric information of the exhibition space and its objects. The LISTEN world model is an extensive virtual-reality-based geometric model created for the AVANGO software platform (Tramberend, 1999). This extensible virtual reality framework supports programmers in the rapid prototyping and developing of virtual reality applications and environments. AVANGO allows for the generation of a geometric scene graph of the real environment. This geometric model enables authors and developers to conduct tests and simulations using the display setups like the Cave Automatic Virtual Environment (CAVE) system (Cruz-Neira et al., 1993). This virtual reality system projects virtual environments on a curved, 240 degree wide-angle screen and creates an almost realistic, three-dimensional reproduction of the real environment. The combination of the AVANGO platform and the CAVE projection capabilities enables the exploration and interacted with the virtual LISTEN prototype and the fast deployment of the system into real environments.

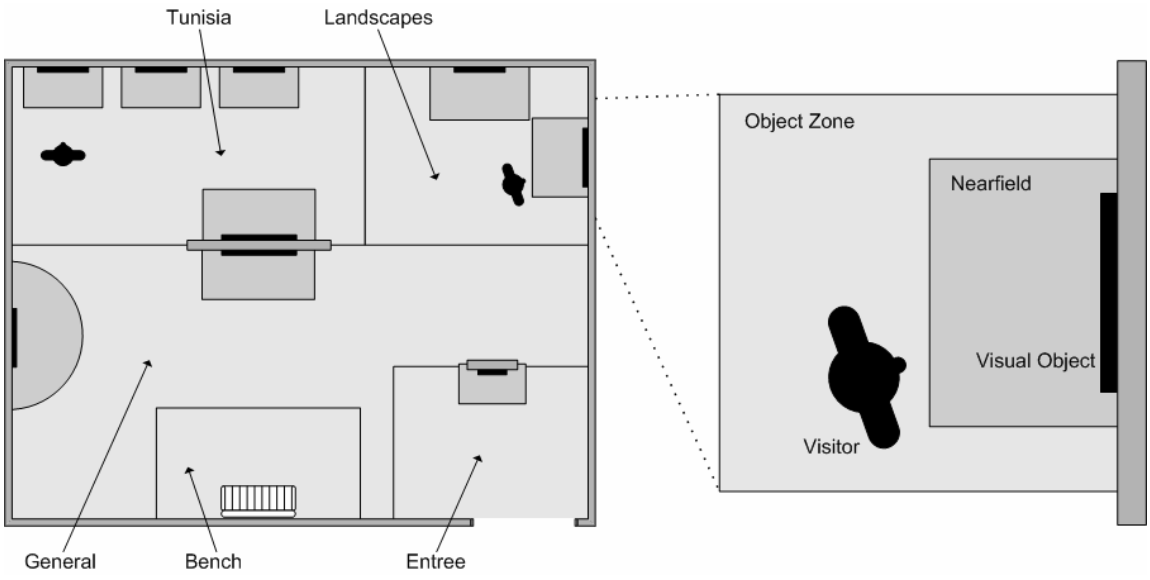


Figure 42 Location Model of the Macke Laboratory

On top of the world model the *location model* defines areas within the world model that the visitor of the system interacts with. This model allows the LISTEN system to gain the location and the focus of the visitor through the interpretation of the visitor's position and

head orientation. It obtains the visitor's relation to the physical space through mapping the position to virtual zones and the head orientation to object identifiers. The virtual reproduction of the real environment enables the positioning of virtual sound sources. The right-hand side of Figure 42 shows a sound emerging directly from the painting.

Figure 42 illustrates the LISTEN location model of the Macke Laboratory on the left-hand side and a magnification of a part of this model on the right-hand side. As the figure shows, a location model segments the exhibit hall into *object zones* and *near fields* (Gossmann and Specht, 2002), which are connected to the respective visual objects placed within the environment. The object zones establish an association with their accordant visual object and take the user's position into account. The near fields refer to smaller, more detailed parts of this visual object and additionally regard the visitor's focus because the spot the visitor is looking at plays a more important role. In general, the object zones as well as the near fields could assume any imaginable shape; however, squares and circles are easier to model.

Modelling the Content

Besides the detailed modelling of the physical environment, for which the LISTEN system has to be deployed, the structured representation of the audio content and the content represented by the visual objects constitute a central issue of the work. The LISTEN system makes use of a large amount of audio information in order to completely immerse the visitor in a realistic soundscape. The available audio content consists of sound entities that form the main entities for generating immersive audio presentations. The single sound entities as such are independent episodes or chunks of an audio presentation that can be combined flexibly.

In order to guarantee a flexible combination of the sound entities, the facts that they can possibly contribute to a variety of visual objects and that they may emerge from a diversity of sources need to be considered. Additionally, in order to make the usage of the LISTEN system an interactive experience, the combination of sound entities needs to enable the accomplishment of a variety of stylistic means. The following paragraphs present a structure of the information domain that addresses these demands and describe the use of an information brokering service for an effective depositing and mediation of the content.

Structuring the Information Domain

The outcomes of the LISTEN workshops (see Section 6.1.9) comprise the election of different strategies and methodologies to structure sound entities and answer the question, how to combine these entities in a highly flexible way for several visitors? In addition, the analysis of the needs of museum visitors and the observation of human museum guides revealed that the event character of museum visits in most cases is much more important than plain provision of information about the art object as such. The presentation of information about artworks is just one style of presentation that can be appropriate and sufficient but which is rarely used in the everyday work of human museum guides.

In most cases human museum guides extend the visual perception of such artworks with "stories or impressions" taken from the respective time period of the artist to immerse the visitor into an authentic experience of the works and the life of the artist. Concerning the introduction of several artworks by August Macke, his works are interconnected by personal episodes, events and experiences of his life. Furthermore, August Macke's interaction with his social environment, such as the reactions and responses to personal letters, media or press articles, are relevant as well as the zeitgeist and other factors regarding the world at his time. Thus, the structured description (meta-tagging) of a possibly large collection of such "stories or impressions" enables the LISTEN system for the immersion of the user into an audio-augmented experience in a flexible way.

The LISTEN system composes and describes the information space of the intended audio-augmented environment through sound entities as the smallest information units. The effective contextualised delivery of sound entities presupposes the intelligent structuring and internal representation of the sound entities. The sound entities can be described along a variety of dimensions, and thus, these entities can be classified into a category system spanning the respective dimensions. The LISTEN system applies the ontology methodology offering *concepts* and *categories* for the generation of a meaningful model of the audio content. Besides the simple classification of the sound entities into a tree structure, the ontology concept provides means of classification of sound entities into *multiple* categories, which allows for the catenation of sound entities and enables individualised sequencing and presentation of these entities. Furthermore, this classification facilitates the association of sound entities to the corresponding visual objects.

Regarding the August Macke exhibition, the category system reflects the need to capture "stories or impressions" from his time and offers dimensions for describing the sound entities technically and from a stylistic point of view. The domain-specific ontology of the Macke exhibition provides classification means based on:

- *technical descriptions* of the sound entities, e.g. the length of the item or type such as music, speech, or sound effects,
- *relations* of the sound entity to physical or visual objects of the environment, e.g. other exhibits, object zones or foci,
- *content* of the described visual object, e.g. phases of work, image genre, or technical aspects regarding the artwork,
- *intended target group of visitors* for the sound entity, i.e. stereotypical listeners, emotional impacts or dramaturgy.

In particular, speech sound entities are further distinguished into subcategories like Citation, Collage, Diary, Letter, Newspaper and others to describe their style of presentation. In an analogous manner, the category system captures the visual objects of the August Macke exhibition, i.e. his artworks. The annotations for the exhibits allow for the derivation of the visitor's interest in August Macke's art and cover matters like:

- *description* of the visual object, e.g. the name of the artwork, its creation date, the type of artistic media,
- *relations* to physical space, e.g. the relation to other exhibits in the environment, anchorage in object zones or focus,
- *content* of the visual object, e.g. phases of work, image genre, or technical aspects regarding the production of the artwork.

The following paragraph illustrates how an information brokering tool enables the creation of this domain-specific ontology for the Macke exhibition and facilitates the authoring of the content of the domain.

Information Brokering for Content Categorisation

The LISTEN system relies on a large amount of information and the structured internal representation of the metadata associated with each sound entity and each visual object enables the effective authoring and distribution of the audio content. Nevertheless, there exists a trade off between the efforts of authoring the sound entities in a highly enriched information representation and the daily work of curators of a museum. The right balance offers an information brokering service as a central management component for all sound entities (Zimmermann et al., 2003a).

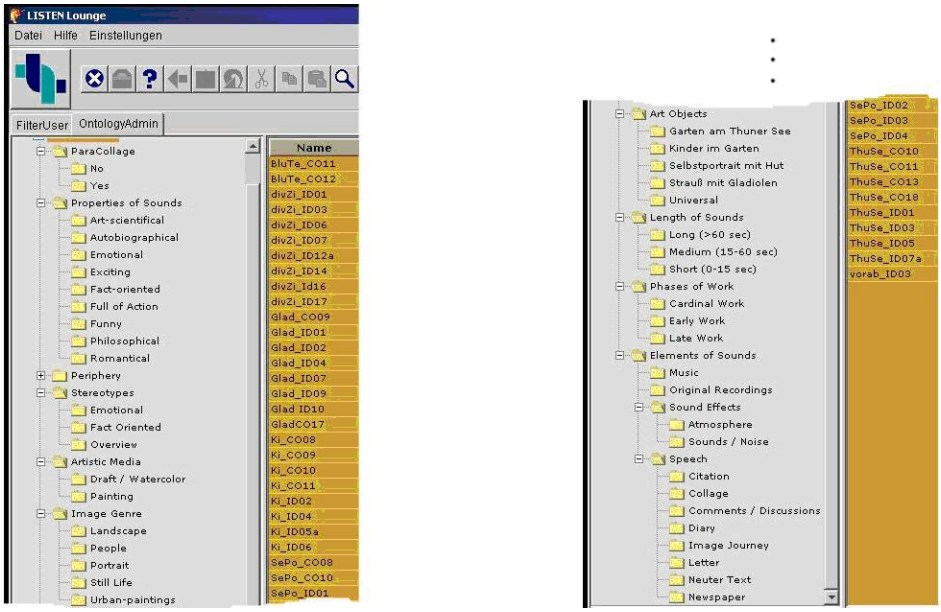


Figure 43 Ontology Used for the August Macke Exhibition

Klemke (2002) defines *information brokering* as the value-adding process of mediation between information demands and information offers. Information brokering processes create and make use of a number of information items that describe single units of information (Klemke and Koenemann, 1999). Each information item is an instantiation of a concept,

which describes the structure of the brokered items (Klemke, 2002). In order to organise information items, *categories* describe fundamental principles or ideas. In case of a virtual exhibition, the use of a central management component for all information items facilitates the management of items composed of texts, sounds, graphics, movies and other hypermedia items like interactive flash components. The Broker's Lounge (Jarke et al., 2001) facilitates the development and management of information brokering services that deliver filtered information on demand. The tools provided by this environment enabled the efficient creation of an information brokering service for the Macke Laboratory and allows for fast changes of the structure. Figure 43 shows the graphical user interface of the Broker's Lounge. It shows the model of the content organised in a tree structure and a list of sound entities required for the Macke exhibition. The multidimensional classification and the concatenation of sound entities on several channels, i.e. typically music, effects, and speech, allows for a combination of entities in a collage style. In association with the Mobile Collector (cf. Section 5.4.2) this information brokering service becomes a valuable instrument that eases the authoring task of the museum curator.

6.1.5 Derivation

Based on the available information provided by the tracking system and encoded in the domain model, the semantic layer of the LISTEN system architecture derives the complete model of the user's context represented by a set of attribute-value pairs. The context model comprises the context attributes "time", "position" and "orientation", which directly correspond to the accordant sensor components described in Section 6.1.3, and the context attributes "location", "focus", "speed", "visit_history", "motion_style" and "interests" (cf. Figure 41, semantic layer).

Name	Type	Description
time	Time	Contains the current time to the second.
position	Position	Contains two float values that indicate the x,y-position of the visitor using the world model as the basis.
orientation	Integer	Contains the value of the visitor's head orientation in angular degree.
location	String	Indicates the identifier of the exhibit entity associated with the zone the visitor is currently standing in.
focus	String	Indicates the identifier of the exhibit entity that the visitor is looking at.
speed	Float	Indicates the speed that the visitor is moving with.

visit_history	History	Holds a list of identifiers of exhibit entities the visitor has seen and identifiers of sound entities the visitor has heard.
motion_style	Stereotype	Indicates the motion style of the visitor through the exhibition.
interests	Interests	Indicates the visitor's interests in August Macke's artworks.

Table 5 Description of the Context Attributes Used for the Macke Laboratory

Table 5 provides an overview of the context attributes instantiated from the Context Toolkit and used to model the visitor's context for the Macke Laboratory. As Figure 41 shows, all context attributes are listeners to change events provided by the tracking sensor. Whenever an event is received, these attributes start the derivation of semantically enriched information. On the basis of the interaction history, the motion styles, and the interest model, the LISTEN system is able to adapt the auditory scenery and to generate different sound presentations for the Macke Laboratory.

```
<listen_location_model>
     <Tunisia TYPE="toolkit.semantic.model.spacemodel.SquareZone">
          <X_COORDINATE VALUE="146.0"/>
          <Y_COORDINATE VALUE="77.0"/>
          <X_EXPANSION VALUE="139.0"/>
          <Y_EXPANSION VALUE="70.0"/>
     </Tunisia>
     <KinderImGarten TYPE="toolkit.semantic.model.spacemodel.CycleZone">
          <X_COORDINATE VALUE="7.0"/>
          <Y_COORDINATE VALUE="220.0"/>
          <RADIUS VALUE="55.0"/>
     </KinderImGarten>
     <Bench TYPE="toolkit.semantic.model.spacemodel.SquareZone">
          <X_COORDINATE VALUE="200.0"/>
          <Y_COORDINATE VALUE="320.0"/>
          <X_EXPANSION VALUE="80.0"/>
          <Y_EXPANSION VALUE="43.0"/>
     </Bench>
</listen_location_model>
```

Figure 44 Translation of the Location Model into a XML Representation (Excerpt)

The location model described in Section 6.1.4 allows the LISTEN system to interpret the visitor's position and head orientation and maps the position to virtual zones (location) and the orientation to the identifiers of visual objects (focus). The context attributes representing the visitor's visit history, motion style and interests require a more complex derivation process, which is illustrated by the following paragraphs.

Visit History

The derivation of the context attributes "location" and "focus" bases on the location model. Figure 44 shows the XML representation of the location model, which is used for the translation of the visitor's position to virtual zone identifiers and the head orientation to object identifiers. Therefore, the location model constitutes the main source of establishing relations between a visitor, a visual entity and a sound entity. The location and the focus of a visitor lead to relations with visual entities. In turn, this relation causes the establishing of a connection between the visitor and a sound entity. Both types of entity relation allow for a refinement of the visitor's interests and visit history.

The current time constitutes an important variable for the derivation process. The Context Toolkit provides this sensor and reads the required information from the system clock. The combination of the visitor's spatial position and the time information allows for the derivation of the speed the visitor moves with. The current time contributes to building up a visit history database. This history database stores the currently viewed visual object and the current location of the visitor attached with a timestamp.

Motion Styles

People walking through an environment often show different kinds of common or stereotypical behaviour (e.g. clockwise in museums (Oppermann and Specht, 2000)). In a museum environment, the definition of such meaningful stereotypes is a non-trivial task (Rich, 1989). Some easy to identify stereotypes are for example adults and children. However, the course of the Macke Laboratory has shown that such simple stereotypes are not expressive enough because of the lack of distinguishing features within each category. A more visitor-oriented approach requires a better understanding of what the visitor's motivations for the visit are. These motivations can be looked at from different perspectives like for example the visitor's learning or visiting style. McCarthy and McCarthy (2005) distinguish four different types of learners (imaginative, analytical, common sense and experimental) and Gardner (1993) orthogonally identifies seven different types of learning approaches (linguistic, logical-mathematical, musical, bodily-kinaesthetic, spatial, interpersonal and intrapersonal). Each visitor of a museum belongs to one or more of these categories. However, a fast classification of a visitor into these categories without any pedagogical tests prior to the actual visit of the exhibition will be inaccurate.

Visitors exhibit different approaches to experience the exhibition. Véron and Levasseur (1983) determined visiting styles as an approach to classify the way how different visitors explore an exhibition following observations of animals: ant (following the curator's path), fish (holistic view), butterfly (interest in all exhibits without following the curator's path), and grasshopper (interest only in specific exhibits). According to these visiting styles, the Macke Laboratory introduces motion styles as the stereotypical behaviour of a museum visitor. Motion styles can be seen as representing possible ways of looking at exhibits:

- *Sauntering*: the visitor is slowly walking around with an excursive gaze
- *Goal-Driven*: the visitor displays a directed movement with the gaze directed towards a specific artwork
- *Standing, Focused*: the visitor is standing with the gaze directed towards a specific artwork
- *Standing, Unfocused*: the visitor is standing or sitting with an excursive gaze

The three context attributes "location", "focus" and "speed" determine the motion style as a stereotypical behaviour of the visitor. The stereotype context attribute provided by the Context Toolkit (cf. Section 5.2) is reused for the implementation of these motion styles in a museum environment.

```
<motion_style>
     <standing>
          <LESS TYPE="NUMBER">
               <REFERENCE ATTRIBUTE="visitor.spatial.speed"/>
               <REFERENCE VALUE="0.03"/>
          </LESS>
     </standing>
     <standing_focused EXTENDS="STANDING">
          <EQUALS INVERT="true" TYPE="SYMBOL">
               <REFERENCE ATTRIBUTE="visitor.spatial.focus"/>
               <REFERENCE VALUE="None"/>
          </EQUALS>
     </standing_focused>
     <sauntering>
          <AND>
               <GREATER TYPE="NUMBER">
                    <REFERENCE attribute="visitor.spatial.speed"/>
                    <REFERENCE value="0.03"/>
               </GREATER>
               <EQUALS INVERT="true" TYPE="SYMBOL">
                    <REFERENCE ATTRIBUTE="visitor.spatial.focus"/>
                    <REFERENCE VALUE="None"/>
               </EQUALS>
          </AND>
     </sauntering>
</motion_style>
```

Figure 45 Three Selected Motion Styles Represented in XML

Figure 45 illustrates the accordant definition of motion styles in XML. This figure shows the extension of a basic "standing" stereotype through a more specific "standing_focused" stereotype. Each change in the visitor's context triggers the next determination of the best-suited motion style. The introduction of motion styles enables the system to accordingly adapt the scenery and cause a different sound presentation based on the visitor's observation type.

Interests

In addition to the motion styles of the museum visitor, the visitor's interests need to be inferred from her interaction with the environment in order to provide the visitor with a personalised selection of appropriate contents. Because the LISTEN installation for the Macke Laboratory disclaims explicit visitor feedback, the approach of Pazzani and Billsus (1997) has been adopted for the derivation of interests: It is assumed that the more time the visitor spends with a specific exhibit the more she likes it. Statistical models calculate the average time visitors spend in front of a painting, which enables the determination of points of interest in the exhibition. The combination of this average time and the meta-information concerning the paintings and sound entities allows for an assumption about the visitor's interests. What the visitor apparently likes is expressed in terms of this meta-information, which is assessed regarding the time that the visitor spends with a specific visual or acoustic object, and transferred into the interest model. The interest model equals a name-value list, in which each name corresponds to an interest area (e.g. landscape, portrait, etc.) with an associated value indicating the level of interest. Such a score-based interest model induces a procedural aspect concerning time, space and meta-information.

6.1.6 Implementation of the Context-Aware Behaviour

The control layer of the LISTEN system takes decisions between several high-level adaptation methods and different strategies for tailoring the soundscape presentation within the visitor's environment. The selection and presentation of sound entities base on semantically enriched information about the visitor's context as described in the previous section. The audio-augmentation of a museum additionally requires authors and designers of the LISTEN environment to regard domain-specific constraints. This section illustrates the steps that lead to the realisation of the context-aware behaviour of the LISTEN system installed for the Macke Laboratory.

Adaptation Goals for Audio-Augmented Museums

A LISTEN environment can increase the perceptual, emotional and pedagogical effect of an exhibition. The installation for the Macke Laboratory concentrates on the pedagogical issues in the procurement of arts and disregards the artistic use of the system. The three major goals of the Macke Laboratory are the provision of information regarding the particular artworks, the assistance in the perception of the artwork compilation and the support of visitors walking through the exhibition in groups. Accordingly, three adaptation goals have been defined for this LISTEN installation, which are described by the following paragraphs.

Increasing Knowledge

The consideration of the level of detail makes it possible for visitors to access information depending on their interests and favour. The information contained by the sound entities should comply with the visitor's knowledge and interests. The LISTEN system assumes that

the longer the visitor's focus lingers on some visual object the more interest in this object is expressed. The level of interest corresponds to the complexity, the amount, and the style of already received information about a single object and is transferred to succeeding objects. If one of these succeeding objects complies with the visitor's interests, the sound presentation directly steps into the right level of detail, and the system plays sound entities that are classified on the adequate information depth and style.

Increasing the Comprehension of Semantic Relations

In comparison to the recognition of the exhibits and their associated information, the underlying information structure is not that easily recognisable by the visitors. A structure among the exhibits or among the associated information items can emerge a semantic, hierarchical or chronological link between these objects. The adaptation of the information structure takes into consideration the semantic relations between objects. Traditionally, this information structure corresponds to a tour arranged by museum curators. Such tours exploit the knowledge of the experts about the overall collection of artworks and form a systematic representation of this knowledge.

Considering the Social Context

If visitors depict a spatially and temporally similar behaviour, they might want to receive similar audio information, e.g. a family walking through an audio-augmented museum. In contrast to each single person having heard an entirely different presentation, a discussion after the visit about the seen objects would become possible through the clustering of similar people. Vice versa, the adaptation to the social context includes breaking up clusters of people. This would lead to a better distribution of people among several visual objects.

Adaptation Targets and Means

Adaptation does not always occur at the same level and different parameter combinations accomplish a wide range of adaptation methods and means. The three major targets for adaptations are the information structure, the sound presentation and the location model. The following paragraphs describe these three adaptation targets and provide means of their modification according to a changing context.

Adaptation of the Information Structure

The basis of every kind of adaptation of sound presentation forms the selection of the appropriate sound entity. The sequence of the selected sound entities qualifies the information structure. One major aspect that has influence on this selection process comprises the content the selected sound entity reflects. For an audio-augmented museum the content represented by the sound entity needs to comply with the content of the exhibits. Another important facet that affects the selection of a suitable sound entity denotes the entity type regarding aspects such as speech, music or effects.

Adaptation of the Sound Presentation

Besides the selection of the next sound entity to be played, a variety of parameters affects and influences the character of the sound presentation. The character of the sound presentation predominantly depends on the sound source, which play-back the sound entity, and the alternations or mutations of this sound source over time. The selection of the sound source determines the direction, from which the sound emerges, and the motion, with which the sound moves. In addition, the sound presentation can be modified taking into account when, with which volume, and how long a sound is played.

Adaptation of the Location Model

The location model of the LISTEN system virtually divides the physical space into zones that are associated with specific visual objects. In the LISTEN system visitors enter and leave zones in virtual space. Some visitors want to step back and look at the object from a different viewpoint. Because the visitor still shows interest in this specific object, the associated zone should adapt to the visitor's particular behaviour and expand up to a predefined point ("zone breathing") in order to further provide the visitor with sound entities. The adaptation of the space model allows the transportation of a sound entity, which has not completed playing, from one zone to the other as well.

The Role of Context

The context of a museum visitor plays two important roles (cf. Section 4.1.2) in the adaptation process: it is a trigger and a filter. The system observes the evolution of the visitor's context, and changes in certain attributes of this context information trigger different activities of the adaptation process. The context attributes used as triggers for such an activity are "location", "focus" and "visit_history". Changes in the visitor's location and focus indicate that she moved into different zone or examines a different exhibit, and changes in the visit history indicate that a sound entity has been successfully played back. Both types of changes require the selection of a new sound entity in order to continue the presentation.

Triggered by changes in the context, the control layer uses the three context attributes interest, motion style and visit history as a filter in order to generate three lists: a list of sound entities, a list of visual entities and a list of visitors. The list of sound entities and the list of visual entities represent the set of LISTEN domain objects that can be recommended to the visitor because these objects are ranked with regard to the visitor's interest, motion style and visit history. The list of visitors is used to identify clusters of visitors that have certain characteristics in common. The LISTEN system is able to detect clusters of similar visitors and to react appropriately.

Two-Phase Adaptation Process

For the realisation of a specific and domain-dependent tailoring of the soundscape to the individual visitor and for the achievement of the adaptation goals, the adaptation process built into the LISTEN system assembles the abovementioned adaptation targets and means as well as context roles to achieve more complex adaptation strategies. Therefore, adaptation strategies from the field of adaptive hypermedia (Brusilovsky, 1996) such as *adaptive guiding*, *adaptive annotation* and *adaptive hiding* have been analysed and assessed regarding their suitability and applicability for the adaptation of audio-augmented environments. *Adaptive prompting* denotes a basic behaviour of the system, which either follows the visitor in a more passive mode and paces her exploration of the space, or actively gives auditory cues to guide her on a predefined adapted tour.

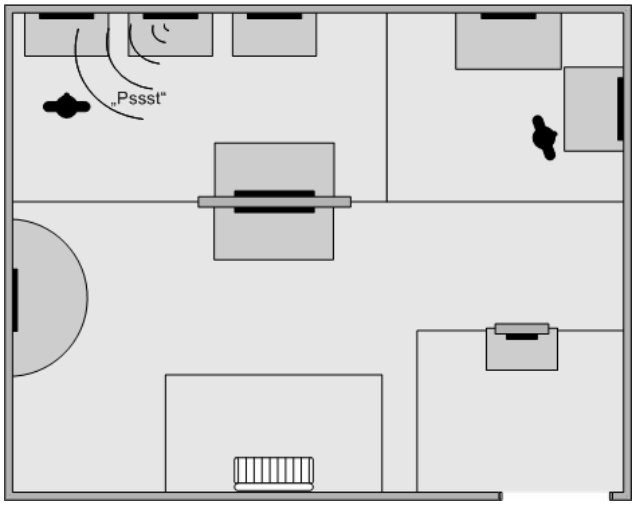


Figure 46 Attracting the Visitor's Attention

During the exploration of the space the system tries to make the interaction with the system transparent for the visitor. Therefore, the system applies two auditory layers, the *sonification* and the *dialogue layer*. The sonification layer immediately responds to the position of the visitor and acoustically makes the invisible interface elements (zones, segments) comprehensible through significant atmospheres. These atmospheres align with the content accessible in the zone or segment and are adjusted to the present exhibit. The characteristic of the sonification adapts to the analysis of the motion style. The content of the dialogue layer depends on the analysis of the motion and interests of the visitor as well as her position. The visitor recognises this layer independently from the sonification layer. The emerging temporal structures of these two layers can overlap, but are independent in general.

The orthogonally aligned pro-active behaviour of the system consists in the recommendation of an exhibit. The LISTEN system emits special attractor sounds from specific sound sources in order to draw the visitor's attention to a certain exhibit (see Figure 46). This adaptive method allows the recommendation of entire predefined tours through the audio-augmented

environment, e.g. arranged by artists or curators of a museum, chronologically ordered, or determined by the visitor's personal interest or motion style.

The adaptation process follows adaptive prompting as the overall strategy, which passively follows or pro-actively leads the visitor through the exhibition hall. Basically, the adaptation process decomposes two phases: a *contextualisation phase*, which considers the three context attributes "location", "focus" and "visit_history" as both a trigger and a filter, and a *personalisation phase*, which considers the remaining context attributes as a filter.

Changes in the context attributes "location" and "focus" indicate a visitor, who moved into different zone or views a different exhibit. Furthermore, the visit history indicates the completed playback of a sound entity. An alternation in the values of any of these three context attributes triggers a pre-filtering process, which retrieves a preliminary set of sound entities from the repository, a set of exhibits and a set of visitor identifiers. The properties of the retrieved entities correspond to the location and focus of the visitor. In addition, the pre-filtered sound entities and exhibits have not been heard or viewed so far.

Sauntering	⟶	Curator speech, comparison of landscapes depending on the visitor's interests, sonification
Goal-Driven	⟶	Only sonification, no dialogue
Standing, Focused	⟶	Single dialogue (speech) to the artwork, sonification steps to the background.
Standing, Unfocused	⟶	No sonification and dialogue (music)

Table 6 Motion Styles and their Influence on the Selection of Sound Entities

The completion of the retrieval activates the subsequent personalisation phase, which regards the visitor's motion style and interests. The first step of this phase sorts and combines the pre-filtered set of sound entities and the set of unvisited exhibits, which results in an ordered list of items that fit best the visitor's motion style and interests. This step utilises a similarity measure (see Section 5.3.3) that matches the two attributes "motion_style" and "interests" of the visitor's context with the descriptions of the two entity types provided by the ontology of the information brokering service (cf. Section 6.1.4). Furthermore, the similarity assessment procedure considers existing connections among the entities preset by the authors of the audio-augmented museum environment. Per default, the sound entities are connected through the information depth they represent and the visual entities are connected through chronological constraints. For the Macke Laboratory, the authors created some additional constraints based on the visitor's motion style as shown in Table 6. In addition, some combinations of motion styles with the location prioritise a pro-active strategy like for instance sitting on the bench (see Figure 47).

The second step of the personalisation phase selects the best-suited entity from the ordered list of results. Depending on what type of entity is ranked higher in the sorted list, the most appropriated one can be a sound entity or a visual entity. In case of a sound entity, the system plays this sound from the sound source connected to the location and focus of the visitor. In case of a visual entity, the system emits an attractor sound from the sound source connected to the respective visual entity.

```
<play_bench_sound>
       <PRECONDITION>
           <AND>
                <EQUALS TYPE="SYMBOL">
                     <REFERENCE ATTRIBUTE="visitor.spatial.location"/>
                     <REFERENCE VALUE="bench"/>
                </EQUALS>
                <EQUALS TYPE="SYMBOL">
                     <REFERENCE ATTRIBUTE="visitor.spatial.motion_style"/>
                     <REFERENCE VALUE="standing_focused"/>
                </EQUALS>
           </AND>
       </PRECONDITION>
       <ACTIONS>
              <play_sound>
                     <PARAMETERS>
                           <sound_name VALUE="bench_03.aiff">
                     </PARAMETERS>
              </play_sound>
       </ACTIONS>
</play_bench_sound>
```

Figure 47 The Segment “play_bench_sound” of the Rule Set of the Museum Guide

The authors of the audio-augmented museum environment defined the behaviour of the application through a rule system. With the aid of this rule system, the system controls and accordingly adapts the scenery in order to cause different sound presentations. Figure 47 depicts an example of the rule “play_bench_sound” as one part of the rule system. This figure illustrates, that the rule is divided into a *precondition* and an *actions* part. The system executes all actions step by step if the preconditions are fulfilled. The rule causes the system to play the sound file named “bench_03.aiff” if the visitor’s location equals “bench” and the motion style equals “standing_focused”.

6.1.7 Actuation

The visitor experiences the information regarding August Macke’s paintings through motion-controlled wireless headphones and what the visitor hears depends on her position in space, head movements and the moving direction. The environment adapts to the individual behaviour of each visitor within a well-defined space. The resulting auditory presentation

exceeds the conventional stereophonic headphones by an extended binaural rendering technology that immediately adapts to the visitor's head movements. The sounds seem to emerge from everywhere around, and thus, the sound presentation extends the physical space and immerses the visitor in an extended three-dimensional environment that can be intuitively explored.

Spatial sound rendering and the segregation of sounds emanating from different directions are key factors for improving the naturalness of the LISTEN environment. The Institut de Recherche et Coordination Acoustique/Musique (IRCAM, 2007) has developed a real-time modular software system processing spatial sound, the *Spatialisateur*, which allows the reproduction and control of the localisation of sound sources in three dimensions (auditory localisation) and the reverberation of sounds (room effect) in an existing or virtual space (Jot, 1999). The provided library of signal processing comprises elementary objects and operations for reconstructing localisation and room effect cues associated to one source signal. Several operations can be integrated in a single compact processor, which in turn can be associated in parallel in order to process several source signals simultaneously. A higher level user interface controls the different signal-processing submodules of a processor simultaneously, and provides direct access to perceptually relevant parameters for specifying distance and reverberation effects. The LISTEN system exploits these methods for synthesising complex three-dimensional sound environments for describing the interaction of each sound source with the virtual space and for the preparation of the rendering activities.

Besides the technical rendering of the sounds, the natural hearing impression requires headphones with digital wireless technology that do not show a typical headphone sound with an associated, disturbing in-head-localisation. The Individual-Virtual-Acoustics-Technology (IVA-Technology) developed by AKG Acoustics GmbH (AKG, 2007) makes a simulation of the natural spatial hearing via headphones possible. This technology completely eliminates the headphone-inherent phenomenon of the in-head-localisation. The coordination of the IVA-Technology with the head tracking system enables a noiseless and uninterrupted transmission of the audio signals via digital radio communication. The freely configurable radio transmitter is capable of processing any algorithm for positioning sound waves in space.

The actuator layer of this instantiation of the LISTEN system changes the audio augmentation of the environment according to the input from the control layer. The commands received from the control layer basically denote a certain sound entity and a specific sound source, which describe the next activity. The specification of the sound sources as well as the definition of their behaviour is prescribed prior to the system operation using the software modules and user interface provided by the Spatialisateur. Additionally, the control layer provides the actuation layer with parameters that dynamically adjust the rendering of the audio information through changing characteristics of specific sound sources.

6.1.8 Application of the Tool Suite

This section illustrates the role of each of the tools provided by the Context Management System in the design process of a pre-version of the museum guide. This pre-version represents a location-based museum guide that enables visitors to a museum to obtain personalised information about exhibits displayed on a Personal Digital Assistant (PDA) that they carry around. The information about the exhibits comprises the title of the painting, auditory content, which provides curator explanations and narrations, and visual content, which offers visual interpretations of the actual painting.

During the design of this specific context-aware application, the tools provided by the Context Management System supported the software development and content authoring, and enabled a fast testing and debugging of the operational system. In the final version of the LISTEN system, the wireless headphones replaced the Content Player because the objectives of the LISTEN project claim the exclusive use of auditory content and the omission of any additional technology.

In the first step, the Design Tool supported the developer in the specification of the sensor for the position and orientation because it allowed for the predefinition of entity identifiers for the users of the system. In addition, the ACTUATORS panel enabled the developer to change the IP address of the Content Player. Due to the continuous operation of the Content Player during the testing and debugging phases and the shortened battery lifetime, the devices needed to be exchanged and the new device IP address needed to be communicated to the system. Furthermore, the developer applied the Design Tool for the association of the sensors with the pre-implemented context attributes (cf. Figure 36 on page 160). In a next step, the developer constructed the rule system and appropriate configuration of the filter for the retrieval of the content from the content management system. However, the ordering of the retrieved contents according to the motion styles and interests required implementation.

During the preparation of the exhibition hall, the curator of the museum in their role as domain experts and authors used the Mobile Collector to annotate the exhibits with suitable content from the content management system. In front of each painting, the curator selected exhibit-specific and reasonable contents from the content repository and associated their identifiers with the current context snapshot (cf. Figure 37 on page 161). In many cases the values of this context snapshot required some adjustments because the context attribute "motion_style" was difficult to capture. The museum curator was mostly standing in front of the artworks in order to author the content of the targeted context-aware application, and thus, the two motion styles SAUNTERING and GOAL-DRIVEN required explicit input.

During the operation of the system, the Content Player running on a Personal Digital Assistant filtered the content from the content repository according to its (and thus, the visitor's) current context. The Content Player displayed the title of the artwork and the accordant images. The playback of the associated auditory content needed to be triggered by double-clicking on the icon presented on the screen (cf. Figure 38 on page 163).

6.1.9 Evaluation

At the Kunstmuseum Bonn an art exhibition with the deployed LISTEN application opened for the public in the form of the Macke Laboratory in July 2003. Prior to and during this exhibition an evaluation of the contextualisation component of this LISTEN installation has been performed targeting the employment of the tool suite, the adaptation of the design view and the operation of the deployed application. From the perspective of the realisation of the adaptive behaviour of this LISTEN installation, the participating people represented the roles of three actors (cf. Section 4.5): domain expert, author and user.

Prior to the opening of the LISTEN art exhibition, the prototype has been reviewed within the scope of two LISTEN expert workshops. For the conduction of these expert workshops, a part of the real museum has been reproduced in a laboratory setting. The test exhibition comprised four paintings arranged in the corner of a room. The group of participants chosen for the two workshops comprised artists, lyricists, sound designers, and museum curators. The *first* expert workshop led to a large list of requests for changes of the system, in particular, changes regarding the contextualisation component of the system. These requests were addressed and realised for a *second* expert workshop, which constitutes the final review prior to the opening for public visitors.

The user evaluation comprised a group of 699 visitors to the Macke Laboratory at the Kunstmuseum Bonn who were interviewed subsequent to their tour through the exhibition. The visitors were asked to provide a personal statement regarding the application of the LISTEN system in the museum environment. The remainder of section describes the results of these three evaluation processes.

The First Expert Workshop

The LISTEN installation composed for the first expert workshop differed significantly from the final installation described by the preceding sections. Particularly, the personalisation aspect of the adaptation process created for this first expert workshop based on a set of selectable stereotypes. Three expressive stereotypes have been defined in previously conducted project meetings:

- *Fact-oriented* putting a high weight on spoken text,
- *Emotional* prioritising music pieces and sound effects, and
- *Overview* focussing mainly on short sound entities.

The classification of a visitor to one of these stereotypes took place prior to the start of the visit. Potential visitors were requested to manually choose their specific stereotype using a touch screen at the entrance of the museum. During the operation of the system, the pre-selected stereotype influenced the character of the sound presentation and the deepness of information the visitor would hear. The leading thought of this approach was to receive an explicit statement from each visitor without bothering her during the presentation.

Assuming the role of users of the LISTEN installation, and thus, of visitors of the reproduced museum's section, the domain experts expressed the utilisation of these stereotypes as their main point of criticism. They neither liked to be clustered or classified nor to publicly state to which class they belong: the request to think about what type of museum visitors they are generated irritation. A minority group did not even experience the personalised character of the audio presentation as a response of their stereotype selection. The elementary revision of this stereotype approach was planned for the next evolution step of the system presented and reviewed within the scope of the second expert workshop

In their role as authors who determine the appearance of the museum guide application the museum curators and sound designers found the combination of the Mobile Collector and the information brokering service useful for the authoring and designing of the audio-augmented environment. In particular, the museum curators acknowledged the possibility to supply content concerning the artworks in an innovative, enriched and less descriptive way.

The domain experts issued a lack of flexibility of the location model as the critical point: The zones surrounding each artwork were recognised as occasionally too small, thus, forcing a visitor to approach the artwork very closely. Furthermore, the domain experts could hardly localise the boundaries of overlapping zones. In this regard, the design view of the contextualisation part of the LISTEN installation allowed for fast adaptations of the location model: the domain experts were able to manually adjust the boundaries of the respective zone and immediately feel the effect, which proves the validity of the concept of adaptable context-aware applications.

In a subsequent discussion the domain experts aspired to a more dynamic concept of the zones. Moreover, the domain experts observed that the recognition of the real focus caused irritations: the tracking system determines the spatial position of a visitor, which enables the derivation of the zone where the person is located. However, the focus can potentially be on a visual object belonging to another zone. Furthermore, in particular the sound designers could not realise whether the changes in the audio-augmented environment were due to their movements in the space or were part of the audio sequence. An adaptation of the rule system that addressed a higher prioritisation of the focus did not achieve the desired effect and resulted in an additional development step for the second expert workshop.

Overall, after the presentation of the implemented system the domain experts jointly agreed on the success of the synesthetic experience: they enjoyed the combination of audio-visual perception and felt an augmentation of the interaction with the real visual objects. A few domain experts also claimed the provision of a future visitor with some mechanism of controlling the audio presentation more actively: The LISTEN approach inhibits the visitor in either switching to a different type of sound presentation while experiencing the environment, or starting and stopping the presentation. In a final discussion, the domain experts agreed on the consensus that the level of immergence is clearly increased without any additional technical equipment.

The Second Expert Workshop

With regard to the contextualisation component of the LISTEN installation, several improvements have been accomplished subsequent to the first expert workshop. This improved contextualisation component largely matches the final version deployed in the real museum, and therefore, the preceding sections can be referred for a detailed description.

The first improvement aimed at more flexible zones in the location model in order to overcome the problems of the small and static zones. Therefore, the concept of adaptive "breathing zones" was investigated, which links the sounds tight to a visitor's behaviour in observing the artworks. In addition, auditory icons, which provide landmarks in the virtual environment navigation, were inserted in the audio presentation in order to make a visitor aware of the interaction with the environment.

In a second step, the utilisation of stereotypes needed a revision. The manual selection of stereotypes prior to the visit of the museum contradicted the main objective of the LISTEN project to not provide the visitor with any input devices, neither desktops nor handhelds. The revised contextualisation component automatically categorises a visitor into stereotypes at runtime. These improved internal stereotypes consisted in the four *motion styles* as described in Section 6.1.5 and illustrated in Table 6 and represent the visitor's style of moving through the exhibition hall and formed the basis of the adaptive behaviour leading to different audio presentations.

As a third improvement of the first prototype, the analysis of the interaction history gained increased weight in the second expert workshop and played a more important role in the adaptation process of the system. Based on the calculation of the average time, which visitors spend in front of a painting and inside a certain zone, the application inferred an interest model of each visitor during the course of her visit (see Section 6.1.5). This interest model integrated and covered certain characteristics of exhibits and sounds.

Prior to the opening of the Macke Laboratory to the public, the second expert workshop targeted the concluding evaluation of the system with a focus on these three improvements. The same group of domain experts participating in the first expert workshop took part in this second expert workshop. The following paragraphs summarise the final results of the evaluation.

Due to the motion styles and the dynamic location model, the reviewing domain experts now positively noticed a clear interaction with the system only through their movements. By using their bodies as interfaces, the domain experts recognised the interaction means very fast and were able to intuitively use these mechanisms. The consideration of their acceleration and activity in the selection of suitable sound entities gained acceptance among the domain experts and was assessed as a valuable extension of the system.

The museum guide automatically determines the motion style through an analysis of the visitor's speed and focus. The possibility of adapting the design view of the museum guide enabled the domain experts to adjust the determination of the motion styles on-site at the

museum. Because the real-world movements of a visitor continuously refine her motion style the domain experts positively assessed this means of trial and error performed during the operation of the system. These statements again constitute a positive indicator and a clear validation of the concept of adaptable context-aware applications.

Due to the analysis of the interaction history, the domain experts positively recognised the reduction of repetitions in the sound presentation and a continuous story. In the combination of the interaction history and the interest model, which both have influence on the selection of the sounds for presentation, they realised a new procedural aspect concerning time and space integrated into the system. Furthermore, the analysis of the interaction histories of several visitors allowed the domain experts to reveal *points of interest* within the exhibition. For the museum curators and sound designers as authors of the museum guide, these points of interest made a corresponding adaptation of the design of the information space possible using the Mobile Collector.

A critical point in the centre of discussion became the general use of contextualisation and personalisation methods in the LISTEN system. The domain experts agreed on the opinion that an adaptation process makes sense in application domains like demonstrations, shows and events with a presentational character. However, domains such as museums and exhibitions are to be approached carefully because of the sensitive and old-fashioned view of long-established visitors. Overall, the reviewing domain experts came to the conclusion that the personalised selection of appropriate sounds from a large amount of sound entities denotes an important feature of the LISTEN system.

User Evaluation of the Macke Laboratory

In order to find out more about the general "acceptance" of the innovative LISTEN installation, the evaluation involved the total amount of 699 real museum visitors, who were asked to provide a personal statement regarding the application of the LISTEN system in a museum environment. Subsequent to their tour through the Macke Laboratory exhibition, these 699 visitors were asked to fill in prepared questionnaires. These mainly based on the selection among several predetermined statements and ratings. In addition to the marking of the statement that fitted best the visitor's opinion, the visitor's could provide some handwritten statements.

Figure 48 summarises the general attractiveness of the LISTEN system installed in a museum. In total 91% of the interviewed visitors responded positive to their audio-visual museum experience and only 2% considered this service as not suitable. 2% of the visitors had no opinion concerning this LISTEN installation. In order to consider the group of visitors that gave no statement, a worst-case or a best-case scenario could be pursued in the attempt of their classification. Depending on the potential allocation of these visitors to the negative or positive party, the ratio positive to negative assessments would decrease (91%/7%) or increase (96%/2%). However, the positive overall impression of the interviewed visitors still remains.

Furthermore, a special interest of the LISTEN consortium aimed at the visitors' personal reports about the combination of visual and auditory elements realised in the Macke Laboratory. For this part of the evaluation, the evaluation forms suggested specific terms regarding the description of certain characteristics of this combination. The visitors assessed these terms through choosing between three different nuances of fulfilment: yes, no or partly.

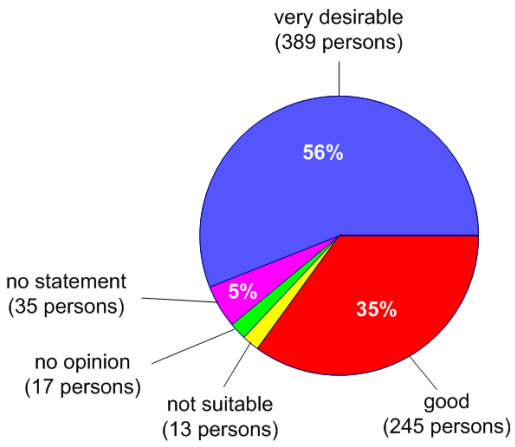


Figure 48 General Attractiveness of the LISTEN Installation in a Museum

Figure 49 shows the overall result of this evaluation. This figure displays the assessments of the visitors regarding the following characteristics of the LISTEN installation: "coherent", "irritating", "boring", "enriching" and "succeeded overall impression". The columns indicate the statements of the interviewed visitors: The blue columns correspond to the answer "yes", the red columns are associated with "partly", green columns refer to "no" and the yellow columns represent "no statement".

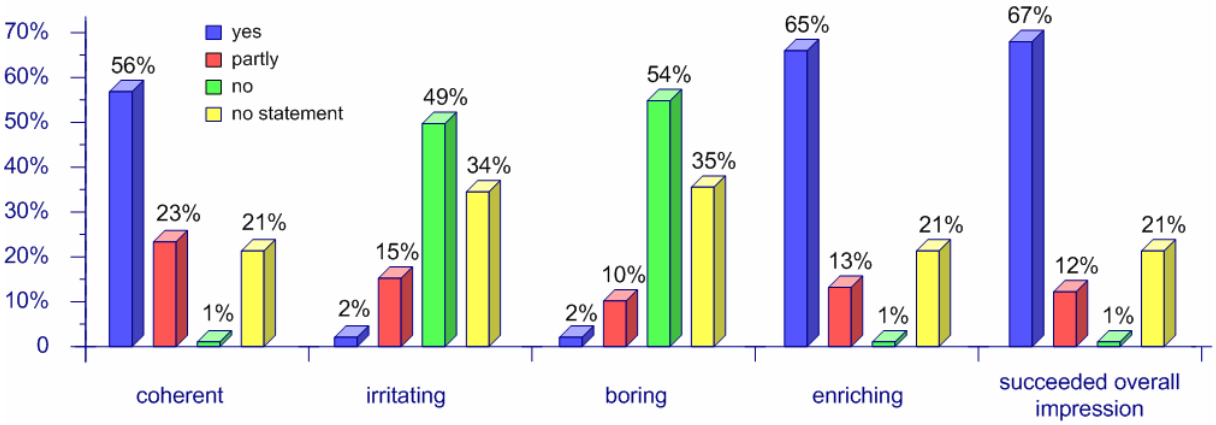


Figure 49 Evaluation of the Combination of Artwork and Auditory Information

In summary, a clear positive feedback has been provided for all the characterisations of the LISTEN installation in a museum. Two-thirds of all interviewees even rated the installation as enriching and succeeded in the overall impression, which can be an indicator for the visitors'

perception of the LISTEN installation as an add-on compared to ordinary museums. In addition, the percentage of negative feedback never exceeded 2% for all five characterisations of the LISTEN installation. This means that even in the worst case scenario, in which the visitors who gave no statement are allocated to the negative party, the positive impression still persists for all characterisations. Furthermore, a clear rise of the number of visitors who gave no statement to the negatively verbalised characterisations "irritating" and "boring" is noticeable. In general, the high percentage of visitors who provided no statement could be due to a wrong choice of terms for the characterisation.

The technological functionality of the LISTEN system was evaluated as well. The compositional structure of the Macke Laboratory allowed a more distinctive analysis of visitors' acceptance of the introduced innovative means of interaction because their effectiveness and functionality was interpreted more directly. The evaluated visitors were asked how they personally experienced the "activation" of auditory information within the Macke Laboratory, emphasising that they could give several answers: 68% of the 699 evaluated people acknowledged that the "auditory information seemed to have been activated depending on one's physical movements and spatial position". Only 4% of the visitors marked that "there was no comprehensible association between one's movements and position and the activation of auditory information". Some the remaining remarks regarding this question outline that the visitors "had to get used to the system first".

Most of the evaluations performed in some of the projects mentioned in Section 6.1.1 have been conducted on an academic level in order to test new prototypes, and therefore, lack a more substantial long-term perspective. Moreover, most of these evaluations were only meant to assess the usability of the prototype on a more general level. Therefore, their focus on personalisation is only marginal. Even if the two expert workshops conducted prior to the opening of the Macke Laboratory did not provide detailed and precise conclusions, these assessments, however, give an idea of what the benefit has been brought along by the contextualisation might or might not be, both for the museums and their visitors.

6.2 Intelligent Advertisement Board

At the CeBIT 2004, the Fraunhofer Institute for Applied Information Technology presented a context-aware advertisement board as an application scenario for the context management system described in Chapter 5. The intelligent advertisement board comprises a large plasma display showing different types of announcements controlled by a computer. Possible application domains of this technology are locations such as train stations, airports, or shop windows. The board responds to its surrounding environment and to changing conditions like noise, trains arriving and departing, the time of day, or people passing the board or showing interest in the announcements. The content of the board's advertisements comprise images from various thematic fields. The advertisement board outreaches traditional approaches because it takes its context into account in order to present content in an intelligent manner.

Without the consideration of the current context, the board would simply present the advertisements in a random order.

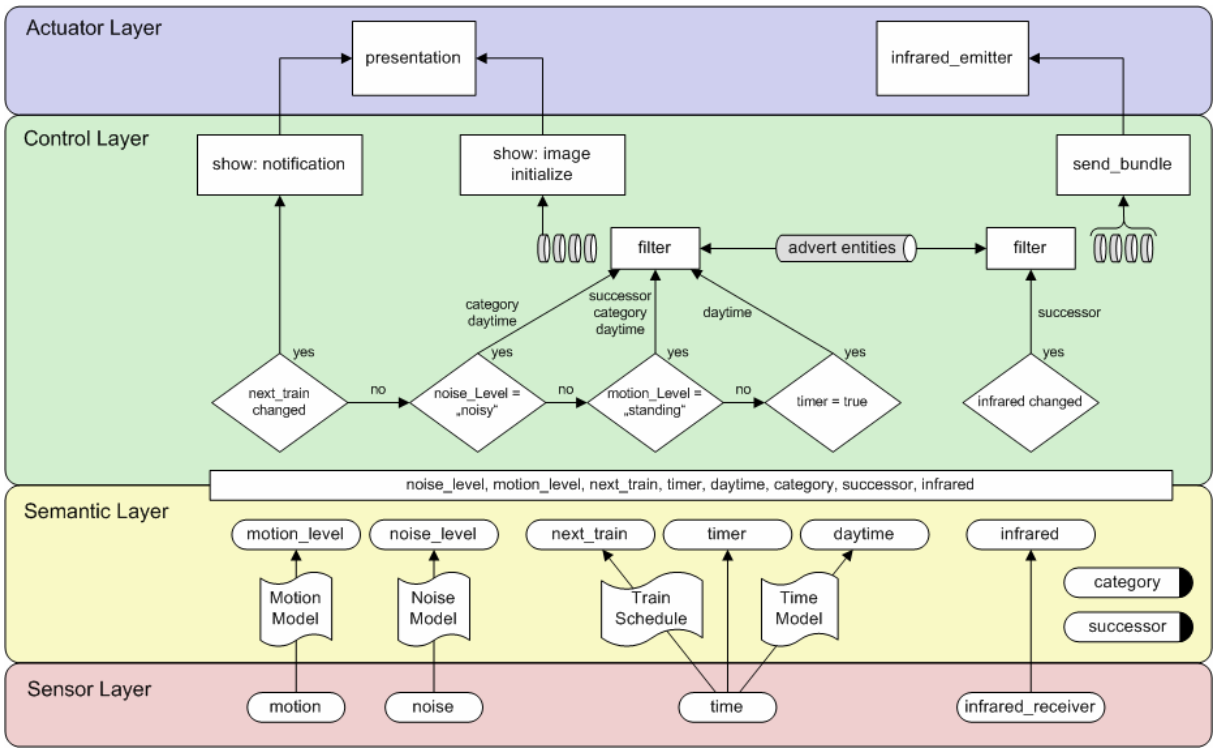


Figure 50 Software Architecture Instantiation for the Intelligent Advertisement Board

6.2.1 Software Architecture

The realisation of the intelligent advertisement board needs to meet several requirements. Its main task consists in the attraction of the public audience. If the attention of a single person or a group of people is attracted, the board should provide continuative information regarding the topic that caught the attention. However, while trying to attract people, the advertisement board needs to avoid dangerous situations for the addressee of the adverts. At train stations this means, that the board must not distract a person's attention from arriving or departing trains. Another requirement the intelligent advertisement board needs to fulfil consists in the downloading of adverts to a Personal Digital Assistant (PDA).

Figure 50 depicts the instantiation of the layered architecture presented in Section 4.4 for the realisation of this context-aware advertisement board. The image shows the distribution of the main application components on these layers. The following subsections describe the instantiation of the architecture in more detail.

6.2.2 Context Acquisition

The challenge in the realisation of the intelligent advertisement board lies in the detection of people showing interest in the currently displayed advert. The adaptation process of the intelligent advertisement board application takes into account the motion in front of the board, the noise in the surrounding of the board, the time and incoming infrared connections. Therefore, this specific context-aware application relies on four sensors: a simple and robust webcam, a standard microphone, an infrared receiver and the time.

The reception component implemented for the webcam connects to the data stream of the small camera. This component performs a pixel analysis based on the comparison of consecutive pictures extracted from the data stream. The reception component returns an integer value within 0 and 100 indicating less or much difference between two consecutive images. In the same manner, the reception component employed for the microphone processes the audio stream and measures the sound intensity. An integer value between 0 and 100 denotes the current volume of noise absorbed by the microphone.

Furthermore, the context-aware application processes data delivered by the two sensors for the time and infrared signals. The "time" sensor accesses the time information provided by the system clock of the advertisement board and delivers information regarding the seconds, minutes and hour of the day. The reception component implemented for the infrared receiver identifies signals from corresponding senders and extracts the identifier of the sending device.

6.2.3 Domain Model and Derivation

The intelligent advertisement board exploits a large repository of content during the adaptation process. The content comprises images, which are designed for the large screen resolution of the plasma display panel and are referred to as advert entities each with its own static context. The context of these advert entities remains static over time and consists of four context attributes:

- their identifier,
- the category they belong to,
- the time of the day to which they are relevant, and
- an identifier for their more detailed successor.

Each image the advertisement board can present falls into one of the categories *sports*, *food*, *shopping* and *news*, and is suitable for the times of day *morning*, *noon* and *evening*. Furthermore, the presentation of the advertisement board bases on two types of images, a large eye catcher and an associated, more detailed successor (see Figure 51). Each eye catcher possesses at least one further level of detail, which shows continuative information related to the main message of the eye catcher.

Figure 51 An Eye Catcher and its more Detailed Successor

The context-aware application has always established a relation to the advert entity that is currently displayed by the plasma panel. This relation allows the application to exploit relevant context information such as the current *category* and a potential *successor* available for the current advertisement. The semantic layer as displayed in Figure 50 on page 198 indicates this specific type of exploited context information through a black marking on the right-hand side of the name for the context attribute. The context model of the intelligent advertisement board comprises the context attributes “noise_level”, “motion_level”, “next_train”, “timer”, “daytime”, and “infrared”.

Table 7 provides an overview of the context attributes instantiated from the Context Toolkit and utilised to model the context of the intelligent advertisement board.

Name	Type	Description
noise_level	String	Indicates the noise level surrounding the board in three grades.
motion_level	String	Indicates the motion level in front of the board in three grades.
next_train	String	Contains the name of the name of the train arriving next.
timer	Integer	Holds the current value of the timer .

daytime	String	Indicates the daytime in three grades.
infrared	String	Contains the identifier of the sending device.

Table 7 Description of the Context Attributes of the Intelligent Advertisement Board

The context attributes “next_train”, “timer” and “daytime” of the context model register as listeners to the “time” sensor. The two context attributes “daytime” and “next_train” determine their values on the basis of specific models that abstract from the time information delivered by the “time” sensor. The context attribute “daytime” counts the hours elapsed during the course of the day and clusters it into three different periods: morning (8-11 hours), noon (11-14 hours) and evening (15-18 hours). The context attribute “next_train” maps the data provided by the “time” sensor to the train schedule of the train station “Messe/Laatzen” near the CeBIT fair ground. The value of this context attribute holds the name of the approaching train. Figure 52 shows an excerpt of the XML representation of the train schedule.

```
<schedule event_class="cebit2004.semantic.model.schedule.TrainArrivalEvent">
       <Event2
              name="S6"
              from="Hannover Airport"
              to="Hannover Trade Fair"
              hour="9"
              minute="15"
              duration="2"
       />
       <Event5
              name="ICE 91"
              from="Hamburg-Altona"
              to="Wien West Station"
              hour="9"
              minute="32"
              duration="2"
       />
</schedule>
```

Figure 52 Segment from the Train Schedule Modelled in XML

The context attribute “timer” receives the values from the time sensor and changes its value in a ten second interval, i.e. every ten seconds all listening components receive a change event sent by the context attribute “timer”. Such an attribute enables the context-aware application to start or abort certain actions based on a fixed time interval.

The context attributes “motion_level” and “noise_level” listen to the corresponding reception components “motion” and “noise” and form the basis of the determination of the activity level in front of the advertisement board (cf. Section 6.2.3). The values provided by these sensors need to be further categorised because they change in too tight time intervals. Table 8

illustrates the mapping of the sensor values to values for the context attributes “noise_level” and “motion_level”.

noise_level			**motion_level**		
value < 30	⟶	quiet	value < 21	⟶	no_motion
30 ≥ value > 70	⟶	normal	21 ≥ value > 38	⟶	standing
value ≥ 70	⟶	noisy	value ≥ 38	⟶	passing

Table 8 Semantic Models of the Context Attributes “noise_level” and “motion_level”

Each context attribute implements the respective comparisons required for this categorisation. These two context attributes indicate an emerging relation between the advertisement board and people in the vicinity of the board. If the noise level measured by the microphone is normal or quiet, and the motion level detected by the camera is standing, it is likely that a group of people stays in front of the advertisement board. The following section describes how the board exploits this relation in order to provide the people with intelligently filtered information.

6.2.4 Adaptation Process

The advertisement board is equipped with simple and reliable sensors, and a large plasma display used as an actuator showing announcements in order to generate an intelligent response to changes of the surrounding environment. The major objective of this installation consists in the addressing of as many people as possible with the presented advertisements. The following two paragraphs firstly specify the adaptation process and secondly describe the implementation of the adaptive behaviour of the context-aware application.

Specification of the Adaptation Process

The corresponding adaptation goals of this context-aware application comprise raising the number of people staying in front of the plasma display and increasing the time these people remain with the presentation. Basically, the intelligent advertisement board places emphasis on the appropriate type and the adequate level of detail of the presented announcements. In order to provide people with information that complies with their interests, the level of detail represented by the presented information needs to increase. The intelligent advertisement board assumes that people staying in front of the plasma display show particular interest in the presented information.

The realisation of this adaptation goal occurs through modifications of the information structure of the presentation, which constitutes the adaptation target of this application. As mentioned earlier, without any contextualisation the structure of the presentation would emerge randomly without any comprehensible connection between two presented

advertisements. The modification of the information structure of the presentation bases on two strategies: the attraction of people (while not bringing them in dangerous situations at the same time) and the provision of comprehensible semantic relations of the presented advertisements.

In order to accomplish the modification of the adaptation target, i.e. the adaptation of the information structure, the adaptation process needs to employ the context of the advertisement board as a trigger and a filter. If changes occur in the values of the context attributes "timer", "next_train", "motion_level", "noise_level", or "infrared", the intelligent advertisement board changes its presentation through the selection and displaying of a different advert entity. The context attribute timer indicates that a time period of ten seconds with no change in the presentation has elapsed. Alternations regarding the context attribute "next_train" report an arriving train. Changes in the context attribute "noise_level" evince people standing near the advertisement board having a chat. Deviations in the context attribute "motion_level" reveal people approaching, leaving or staying in the vicinity of the board. Furthermore, the context attribute "infrared" indicates a reception request for content from an infrared-enabled device.

Any type of change in the values of the abovementioned context attributes requires the selection of a new advertisement entity in order to continue the presentation. The selection of a new advert entity always occurs with regard to the currently displayed advert entity, which the context-aware application retains through a relation established to this advert entity. During the use of context information as a filter, the application exploits this relation because it feeds the selection process with information about a potential successor and the current category. Furthermore, the context attribute "daytime" serves as a filter in the selection process.

Implementation of the Adaptive Behaviour

The intelligent advertisement board observes the evolution of its context, and changes in certain attributes of this context trigger the selection of different announcements. This selection affects the information structure in a way that the presentation complies with the surrounding context. The board shows advertisements for nearby restaurants in the evening rather than at breakfast time because the time of the day is taken into account. Furthermore, the board presents eye-catching and attracting pictures if people in the further proximity do not show any interest. If someone's attention is attracted, the advertisement board displays additional and continuative information that is relevant to its eye-catching parent. The train schedule superposes all other information filtered by context. In case of an arriving or departing train, the advertising board presents with an appropriate announcement and warning and shifts its presentation style to less eye-catching, more informative notifications.

The rules specifying the behaviour of the intelligent advertisement board are informally illustrated in Table 9. If the filtering process activated by these rules delivers an empty set of advert entities, the adaptation process expands the search space through an adjustment of the filter characteristics and a fall back to less restrictive selection criteria. Based on the current advert entity, the filter algorithm gradually broadens the search space: in a first broadening

step, the algorithm filters an eye catcher from the repository, which belongs to the same category and daytime as the current advert entity. If the first step still delivers no results, a second broadening step eases the restrictions of the filter algorithm and causes it to randomly select an eye catcher of a different category. The only constraint of this filter consists in the daytime, which needs to correspond to the current advert entity. The final alternative denotes the random selection of any eye catcher disregarding the category and the daytime.

So far, the adaptation process described above adapts the behaviour of the advertisement board to an entire group and does not distinguish between the individuals involved. This type of contextualisation to user groups introduces a limitation because the adaptation in general does not match exactly individual user's needs. However, a large public display denotes an inappropriate output medium for personal or personalised information. The intelligent advertising board enables a person to show specific interest in one particular advertisement. An individual user can request a personalised information package by pointing with the infrared transceiver of her device to the infrared transceiver of the advertisement board. The board identifies a transmission request, bundles the current advertisement together with all of its successors, and transmits this bundle to the user's personal device.

next_train	$\longrightarrow$	Display a warning notification Block presentation until train departs
noise_level	$\longrightarrow$	Change category Select next advert randomly considering daytime Show next advert
motion_level	$\longrightarrow$	Find successor of the current eye catcher Show successor
timer	$\longrightarrow$	Select next advert considering daytime Show next advert
infrared	$\longrightarrow$	Send eye catcher and its successors to requesting device

Table 9 Rules of the Intelligent Advertisement Board Specifying its Behaviour

Four commands affect the presentation of the advertisement board: "filter", "initialise", "show" and "send_bundle". The "filter" command retrieves advert entities from the repository based on different combinations of the context attributes "daytime", "category" and "successor". The command initialise establishes a relation to the current advert entity displayed by the panel and initiates the context attribute "timer" to count seconds starting from zero. The show command causes the presentation actuator (cf. Section 6.2.5) to display the image represented by the selected advert entity or to show a notification of an arriving train. The "send_bundle" command addresses the "infrared_emitter" actuator (cf. Section 6.2.5) and initiates the transmission of a set of filtered advert entities.

6.2.5 Actuation

Two actuators comprise the actuator layer of the context-aware application: the "presentation" actuator, which is responsible for displaying images to a group of people, and the "infrared_emitter" actuator, which is accountable for transmitting a set of images to the PDA of an individual person. The presentation actuator is a light-weight Java programme, which either displays an image or a specific textual message in full-screen mode. The presentation of a textual message bases on a predefined screen layout, which places each text fragment to its respective position on the display (see Figure 53).

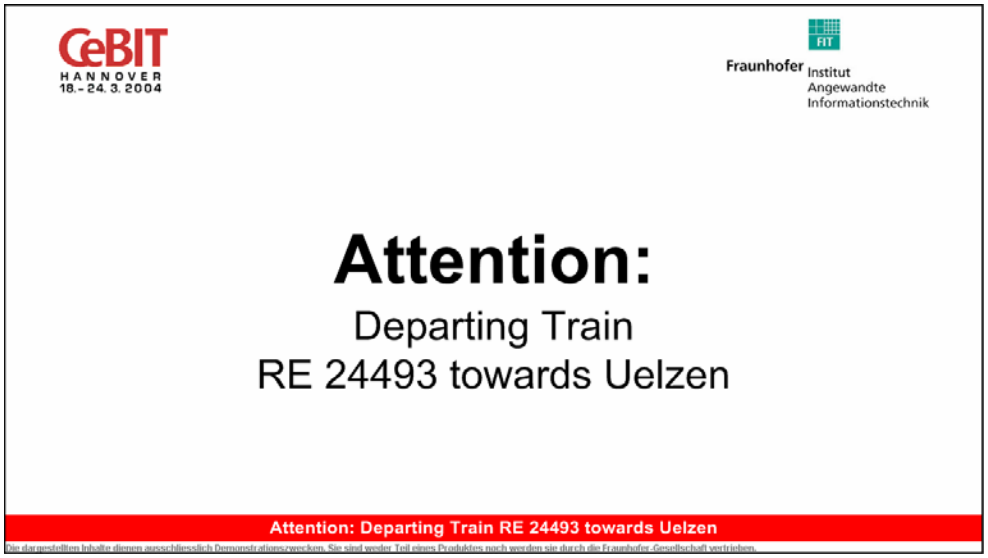


Figure 53 Layout of the Notification Message

The "infrared_emitter" actuator is also implemented in Java and processes the set of images sent by the "send_bundle" command. In a first step this actuator compresses the image set using the compression algorithms provided by the utilities package coming with the Java programming language (java.util.zip, see Java (2007)). The compressed file is then transferred to a PDA via infrared in a second step. The infrared connectivity is available through the third-party driver IrCOMM2k (Kiszka, 2004) for the infrared port in order to provide a COMM connection from the advertisement board side. Furthermore, the object exchange (OBEX) protocol, which is part of the Java application programming interface for Bluetooth Wireless Technology (JSR-82), enables the file exchange between two devices. Details can be found at IrDa (2007).

6.2.6 Application of the Tool Suite

This section describes the application of the tool suite offered by the Context Management System in the development and operation phase of the intelligent advertisement board. During the realization of this context-aware application, the tools supported the software development, design, and content authoring. During the operation of the application the tools

enabled a fast manual adaptation of specific application aspects to the changing conditions and requirements occurred on-site at the trade fair CeBit 2004.

During the development phase the Design Tool eased the definition of the overlay models utilised for the interpretation and abstraction of the sensors for motion, noise and time (see Figure 52 and Table 8). Furthermore, the Design Tool relieved the developers of an implementation of a complex rule system as depicted by Table 9 and facilitated the assembly of the rules required for the determination of the application's behaviour. In addition, the developers applied the Design Tool for the configuration of appropriate filters for the retrieval of advertisements from the content management system.

The preparation phase of the appearance of the intelligent advertisement board was characterised by the utilization of the Mobile Collector. The authors manually annotated all advertisements stored in the content management system with suitable context information. In contrast to the preparation of the exhibition hall of the museum guide, this design step did not take place on the move but stationary at the authors' desks. The context information associated with an advert entity remained largely static and needed to be specified once. Thus, the authors were required to edit the attribute values of the current context snapshot and change these values accordingly.

The Content Player did not play a significant role in the testing phase of the system. However, the already existing implementation of the Content Player could easily be extended by the transmission and presentation of the advertisement bundle selected by the user (cf. Section 6.2.5).

6.2.7 Evaluation

In contrast to the performed evaluation of the application of the Context Management System for the development, operation and configuration of the museum guide, a similar type of evaluation has not been conducted for the intelligent advertisement board due to the lack of time. However, conclusions about the employment and utility of the Context Management System could be drawn through observation and spontaneous interviews with the participating developers, experts, authors and users.

The Context Management System allowed for a harmonized cooperation of people with different education and expertise. Designers of the advertisements were able to easily add their new creations to the content management system. Authors and domain experts annotated these newly added contents with accordant context information using the "edit" mode of the Mobile Collector (cf. Section 5.4.2) that facilitated the content authoring. The work of the developers started at the same time as the work of the designers and authors. However, they finished earlier because they only needed to implement software for three parts of the application: the infrared transmission, the extension of the Content Player and the presentation module for the large-screen plasma display. In this regard a spreading of expertise became possible because each member of the development team could elaborate on a part of the application that demanded her correspondent expertise. Any other required

software was already covered by the Context Toolkit. In collaboration with the domain experts the developers realized the behaviour of the application determined by the rule hierarchy, constructed the context model and associated the sensors with accordant context attributes.

During the operation of the intelligent advertisement board at the CeBit 2004, the Design Tool and the Mobile Collector enabled a shortened reaction to the changing conditions accompanying this trade fair. The setup of the context-aware application prepared in the morning prior to the opening of the trade fair for the public required substantial reconfiguration to cope with the high throughput of people visiting the site. The stand personnel needed to adjust the responsiveness and sensibility of the context attributes "noise_level" and "motion_level" in order to respond to the increased number of people of the audience. Furthermore, the rule hierarchy has been subject to adaptations in order to higher prioritise the motion in front of the intelligent advertisement board compared to the noise. Minor on-site adaptations of the application affected the context attribute "timer" and the train schedule and aimed at an increased diversification of the context-aware behaviour.

The time required for the design and generation of appropriate advertisements exceeded the time necessary for the development of the actual context-aware application. Therefore, the designers still sent contents to the trade fair stand personnel via email that needed to be inserted in the repository and linked with the structure of the other advertisements at runtime. The Mobile Collector enabled a fast addition of new content on-site and linking with possible parents or successors in the content structure during the operation of the application. Furthermore, the Mobile Collector allowed for inevitable changes of the context snapshot associated with the content due to a wrong classification of this content regarding the daytime.

The diversity of conversations conducted at the CeBit 2004 provided a valuable source of feedback concerning the intelligent advertisement board. The board enabled a switching on and off of the application's context-awareness and thus, the audience could experience the random playback of advertisements contrasted by the presentation of intelligently filtered advertisements. The audience clearly perceived the difference between context-aware and non-context-aware behaviour, and a high percentage of this audience identified the benefit of this approach. Sporadically, people even used the context-awareness of the intelligent advertisement board as a means for an explicit interaction with the board. Furthermore, the personalisation of the downloaded advertisement bundle on a private PDA gained acceptance. Some interviewees appreciated this activity as a means to obtain privacy in the proximity of the public advertisement board.

6.3 Summary

This chapter presented practical results and experiences gained through the application of the Context Management System and the exploitation of the prevalent concepts in the

implementation of a context-aware museum guide and an intelligent advertisement board. Both case studies successfully designed, realised and deployed with the tool suite demonstrated the universal applicability of the conceptual framework and the software architecture. Furthermore, the case studies showed the level of support of the design, implementation, authoring and configuration of these context-aware applications provided by the Context Management System. In addition, the evaluations of the case studies reflected the experiences of developers, domain experts, authors and end-users with the application of the Context Management System and with the operational applications.

This section summarizes the issues involved in the development process of the case studies and the combined utility of the concepts, programming techniques and the tool suite developed in the preceding chapters. The first part of this section applies the key requirements elaborated in Chapter 3 for an assessment of the achieved results. The second part lists an array of lessons learned about the application of the Context Management System for the creation of adaptable context-aware applications. In connection with this summary of the evaluation of the case studies, Section 7.2 of the following chapter combines the results of this chapter into major aspects of future work.

6.3.1 Satisfaction of Requirements

This section takes the key requirements elaborated and assembled in Section 3.3.1 as an assessment framework for the Context Management System. The system aimed at a satisfaction of these requirements in order to achieve the goals of this thesis.

Holistic Context-Aware Application Architecture

The elaboration of the software architecture of context-aware applications presented in Chapter 4 already addressed the satisfaction of this requirement at an early conceptual stage. The architecture covers the end-to-end process chain and comprises a layered organization of knowledge processors for the acquisition and derivation of context information, for adaptation methods exploiting this context information, and for actuation means rendering the adaptation. As the realization of this conceptual framework the Context Toolkit implements corresponding software components distributed on a sensor, semantic, control and actuator layer. Each case study presented in this chapter represents one instantiation of this software architecture, and hence, proves the general applicability of this architecture for context-aware applications. Furthermore, they depict the progress of the described process chain: sensors acquire context information, semantically more enriched context information is derived, a rule system reacts to changes in the context information and specifies the adaptive behaviour of the context-aware application, and actuators accomplish the adaptations of the application.

Context- and User-Modelling Integration

On a conceptual level the integration of contextualisation and personalisation is addressed by the formal and operational definition of context provided in Chapter 2 because the described categorization of context information explicitly covers user characteristics. On the level of the software realization, the Context Toolkit offers a set of programming abstractions and model enabling an integration of context and user modelling. Furthermore, the case studies illustrate the realization of this integration in operational context-aware applications. In particular the museum guide takes advantage of the software components of the Context Toolkit and utilises the efficient multilevel filtering of context information in order to combine contextualization and personalization. This context-aware application extensively exploits context information in order to derive a visitor's interests in August Macke's artworks and her motion style representing a stereotypical behaviour of looking at exhibits. In addition, the separation of the adaptation process into a contextualisation phase and a personalization phase reflects the satisfaction of this requirement.

End-User Involvement

The requirement of involving the end-user in the development and operation of context-aware applications drove the conceptualization of the design view on this type of application. This design view enables the configuration of the context-aware application during the development phase and the manual adaptation of the application during its operation. The implementation of the Context Toolkit addresses the realization of the design view: the majority of the provided software components allow for a reconfiguration using XML configuration documents. This design view can be visualised by several different types of user interfaces. The Context Management System offers the Design Tool as an editor of the design view, which allows the involvement of end-users and other actors in the creation process of context-aware applications. Both case studies illustrated the role of the design view particularly in the development and operation phase because they generated high demands on the recurrent adaptations of their context-aware behaviour. The case studies prove the utility of the design view as a means to react fast to changing conditions at the deployment location of the context-aware applications.

Software Design Support

The conceptual basis of the support of context-aware application design is established by the description of the knowledge contained in context-aware applications. The knowledge containers demand on the developers of context-aware applications for the consideration of how their application will reflect the acquisition, derivation, adaptation, and actuation knowledge required for its operation. The Context Management System approaches this requirement by offering tools that satisfy the demands of all actors involved in the design, implementation, authoring and configuration of context-aware applications. The case studies demonstrate, in a concrete manner, the variety of software engineering issues involved in the

development of context-aware applications. In particular, the realisation of the context-aware behaviour can be formalized into three major steps: the identification of adaptation goals, the identification of adaptation targets and means of modifying them, and the identification of roles the context plays in the adaptation process.

Programming Support

The Context Toolkit represents the software realization of the conceptual framework and covers a broad range of programming abstractions and models (cf. Section 5.3) facilitating the development process of context-aware applications. The case studies illustrated the application of the provided programming abstractions during the realization of two operational context-aware applications. Furthermore, both case studies demonstrate the accessibility of these abstractions and models using the configurability offered by the design view on context-aware applications. The description of the case studies did not comprise an explicit reference to the discretization abstraction. However, all software components were integrated into the publish/subscribe event model of the Context Toolkit. Hence, a regulation of the granularity of change of the value stream delivered by the sensors, and thus, an optimisation of the event flow became possible.

Metadata Processing

The Context Toolkit provides a set of properties associated with a single context attribute: timestamp, description, confidence, value range and user preference (cf. Section 5.2.2). The confidence property typically is a function of the quality assessment of a sensor value associated with this context attribute. This metadata enables a further specification of this context attribute and a proper management of its values. The Context Toolkit provides this context attribute properties to any requesting component of the architecture, which satisfies this requirement. The current implementation of the Context Toolkit allows for a determination of the user preference using the design view of the semantic layer but desists a further processing of this value. Both case studies utilised the context attribute properties. The museum guide exploited the timestamp property to construct the visit history of a visitor and checked the values delivered by the tracking system for validity using the value range property. If the position of a user exceeded a specific threshold, the system fell back to a default value for this position. The intelligent advertisement board used the value range property in a similar manner and checked whether or not the values of the sensors “motion” and “noise” fall into the interval [0, 100].

6.3.2 Lessons Learnt

The case studies presented in this chapter allowed for gaining of experiences regarding the design, implementation and operation of context-aware applications and the application of the Context Management System. Furthermore, throughout the course of these case studies new questions and ideas arose, which might trigger a continuation of this work with specialized

foci on these new issues. This section enumerates and describes an array of lessons learned about the application of the Context Management System for the creation of adaptable context-aware applications.

Expressive Design View

The experiences gained in the evaluation of the case studies allow for the conclusion that the expressiveness of the elaborated design view on context-aware applications is sufficiently powerful to form operational adaptable applications. This includes the precise and clean representation of the composition of a context-aware application in all its relevant aspects. In addition, the design view enabled a testing and evaluation of context-aware applications in realistic settings and supersedes large simulation environments provided by laboratories.

However, depending on the complexity of the application, the associated design view depicts a strong variety in complexity. The complexity of the design view increases with the number of components involved and with the level of abstraction required for the representation of context information. Therefore, the configuration language represented by the design view nearly becomes a programming language for context-aware applications. This blurry and smooth transition arises from three aspects that markup and programming languages share: a syntax, a grammar and a semantic. Because the Context Management System aims at a reduction of the complexity of creating context-aware applications for domain experts, authors and end-users, appropriate visualisations of the design view are required in order to preserve the gentle slope of complexity (cf. Section 5.1.1).

The design of end-user tools for building context-aware applications must follow very simple approaches, with visual and intuitive interfaces probably being most suitable in general. The first prototype of the Design Tool seems promising because it already subdivides different aspects of a context-aware application into different panels of the tool. However, the preservation of the gentle slope of complexity requires the reduction of complexity the user is confronted with conforming to the different stages of expertise and development skills of a variety of users. Future interfaces of the Design Tool should offer more variable ways of visualising the design view for a broader range of actors and provide focussed views on more encapsulated units of the context-aware application.

Domain Modelling

The major goal of the evaluation of the case studies consists in the verification of the universal applicability of the models and frameworks presented in this thesis. Therefore, the case studies cover different application domains of context-aware applications. The design and development of the case studies revealed the necessity of performing a strong investigation and understanding of the domain, and uncovered several crucial challenges of modelling this domain. In general, the creation of the case studies gave evidence that a complete domain-independency of a development support of context-aware applications is impossible. In addition, this domain-independency affects all four layers of the application's

software architecture, primarily the semantic layer containing the domain-dependent context model of the application.

However, the case studies also revealed that a core, underlying and configurable programming framework generates a certain degree of domain-independency. Many software components can be implemented in an abstract manner and subsequently instantiated and initialised depending on the requirements of the domain and based on a markup language written to configuration documents. Both case studies prove the validity of this statement though their intensive use of instantiations of the overlay model context attributes and the corresponding design view implemented in the Context Toolkit. These abstraction models allow for a clustering of the sensor values into several categories and enable a modelling and fast adjustment of the boundaries of these clusters.

Both case studies created high demands on the supply of the end-user with contextualized and domain-dependent content as well: the museum guide dynamically arranged speech, music and sound effects and the intelligent advertisement board assembles advert images in order to form a personalised and context-aware application. Traditionally, the contextualised content delivery requires high creation and administration effort (Brown, 1998). The investigation and modelling of characteristics of sounds, paintings and images turned out to be a time-consuming task. The instant combination of content and context snapshots as realized in the Mobile Collector seems to be a promising approach towards a reduction of this effort. In addition, the modelling of any content along a variety of dimensions allows a filtering of this content and a derivation of the user's interest during the adaptation process of the context-aware application. However, the context snapshot approach depicts some limitations regarding the definition of actions for the realization of more complex system behaviour.

Design of Context-Aware Applications

The case studies demonstrate steps involved in the design of context-aware applications. The design of the application contradicts the information flow within the software architecture presented in Chapter 4, which starts in the sensor layer and leads to the actuation layer. The context-aware application performs adaptations that need to be in line with the user's needs in the current context of use and the goal or purpose of the application (cf. Section 4.2.3). Therefore, the application designer needs to think about potentially executable adaptations. Based on the goal of the adaptations, the designer needs to identify potential actuators performing these adaptations and relevant indicators referring to the need for an adaptation. These relevant indicators determine the role context plays in the adaptation process. Indicators are provided by sensors or result from inference algorithms that draw conclusions about the user's context based on these sensor values. In summary, the seven tasks arising during the design of a context-aware application comprise:

1. Identify the goals of the adaptation
2. Identify possible targets of the adaptation

3. Identify methods of adapting these targets
4. Identify context information affecting these adaptation methods
5. Identify indicators triggering an adaptation process
6. Identify acquisition methods for the indicators
7. Identify derivation methods for indicators that cannot be acquired directly

Furthermore, the spatio-temporal relationship of input and output need to be considered carefully in the design of a context-aware application. Each implementation of a layer or sublayer of the (potentially distributed) software architecture described in Chapter 4 introduces certain latency that adds up to a critical total latency of the application. In mobile settings context information changes rapidly, and a potential reaction to a changed context attribute may be delayed regarding time and space. Therefore, a context-aware application needs to be designed in a way that the spatio-temporal coordinates of the results of an adaptation process mesh the spatio-temporal coordinates of the initiating change in the context information. As implementations of the software architecture of context-aware applications presented by this thesis, the case studies did not show any crucial latency, although in particular the museum guide required a total system latency below 59 milliseconds (cf. Section 6.1.3) for a natural representation of three-dimensional sounds.

Motivation for Manual Adaptation

The knowledge about the adaptation goal of the context-aware application prescribes who is in control of the application in each phase of the adaptation process (cf. Section 4.2.3). Many end-users have problems applying adaptable properties of the application or do not use them at all (Karger and Oppermann, 1991). The costs that a user needs to spend on the adaptation of the application have to be rated against the costs that emerge from the further usage of the non-adapted application. The challenge is to provide enough motivation for users to utilise the adaptability of a context-aware application. The evaluation conducted for the two case studies revealed six main drivers increasing the user's motivation for manual adaptation:

Common Sense: The adaptation of the context-aware application becomes an everyday practise through making such applications more operational. An example could be the introduction of a specific context-aware service into a company and every employee *is forced to* adapt this service to her needs.

Completeness of Task: The development of courses of action and the selection of appropriate alternatives of actions are part of the process of completing a task with a context-aware application. Once none of these alternatives is suitable for achieving the task, the user should be able to adapt new work processes.

Individualization: Individualization aims at the desire of humans for being distinguishable from other humans. The exchanging and adaptation of

ring-tones for mobile phones serve as a good example for enabling users for individualization.

Design for Adaptation: If users flinch from taking the initiative of adapting certain properties of the context-aware application, it might be because of the complexity of these properties. The concept of direct activation (Wulf and Golombek, 2001) taken from the field of end-user development aims at lowering the inhibition threshold for users.

Controllability: Components of a context-aware application that are inappropriate for achieving a specific task and can be adapted by the user, allow the user to stay in control over the execution of her task. Controllability depicts an important dynamic aspect: The more a user knows about the power of the system, the more important is it for the user to control this power.

Qualification Advancement: Adaptable context-aware applications allow users for independently opening up new ways of operation. Adaptability enables scopes for design and thereby, animates the user's creativity and advances explorative means of working. Therefore, the adaptability of context-aware application advances learning and qualification.

However, the evaluation of the case studies also showed that the most effective and pleasurable manual adaptations are ones that do not have to happen. The effective and prudent use of defaults can result in no additional correction or adaptation required by the user. If the context-aware application maintains information about the user like the interaction history, previous settings or learned characteristics, this information could serve as a source of defaults.

Chapter 7

Conclusion and Future Work

This chapter concludes the thesis with a summary of results achieved by this thesis and outlines some foci on future research work that builds upon the major achievements presented here.

7.1 Summary of Contributions

The aim of this thesis was to extend the spectrum of actors involved in design, implementation, authoring and configuration of context-aware applications beyond developers in order to reduce the usability problems introduced by context-aware computing. The extension of this spectrum of actors bases on a common and comprehensive understanding of the field context-aware computing and a shared perception of the adherent terms. Previous definitions of prevalent terms in context-aware computing lack such a perception and demand a considerable amount of expert knowledge for their understanding. The understanding and definition of context introduced by this thesis (cf. Section 2.3.2) reaches a broad spectrum of actors in the creation and operation of context-aware applications. This context definition comprises three canonical parts: a definition per se in general terms, a formal definition describing the appearance of context and an operational definition characterising the use of context and its dynamic behaviour. The resulting perception of context puts each entity in the centre of a surrounding individual context. In order to determine the design space of context, the formal part of the context definition structures context information into five fundamental categories. Furthermore, the operational part of the context definition fosters a systematic foundation of the use of context in context-aware applications and emphasises the dynamic properties of context emerging from the transitions between contexts and the sharing of contexts among entities (cf. Section 2.5.2 and 2.5.3).

The technical implementation of context in computer applications results in automatic adaptations of the application behaviour based on an exploitation of context information.

Such adaptations automatically performed by a context-aware application traditionally represent the perspective of the application developer. Because developers cannot anticipate all potential situations and all possible ways of application behaviour during the development phase, users of context-aware applications potentially experience usability problems that in some cases outweigh the benefits of adaptation (cf. Section 2.6). Main causes of these usability problems comprise the lacking transparency of context-aware applications and unavailable means of control and customization of the application behaviour. The discussion on critical usability factors imposed by context-aware applications led to the introduction and definition of *adaptable context-aware applications* (cf. Section 2.7.3). This type of context-aware application involves the end-user as one "extreme" of the spectrum of main actors in the creation and operation of such applications and ensures an adequate fulfilment of the usability goals without eliminating the benefits of the adaptation processes.

An investigation of the state of the art regarding software infrastructures and programming toolkits aiming at a facilitation of the software engineering process of context-aware applications revealed that current approaches fail at providing support for the construction of *adaptable* context-aware applications. The survey conducted by Chapter 3 examined selected approaches of particular relevancy for this thesis and has been guided by the recent research directions focussing on the involvement of a broad range of actors in this development process. An identification of key requirements addressing the extension of the spectrum of actors allowed for a more detailed assessment and a summarisation of shortcomings of existing approaches (cf. Section 3.3). These shortcomings motivated the research pursued in the further course of this thesis.

The provision of programming and architectural-level support for adaptable context-aware applications requires a conceptual foundation that follows a holistic approach covering all relevant aspects of the application. This conceptual framework needs to comprise a software architecture of context-aware applications that is universally applicable in several application domains. The application-oriented decomposition of context-aware applications into major functional constituents resulted in such an architecture. Therefore, a systematic organisation and structuring of the knowledge required for the construction and operation of a context-aware application has been conducted. Independently from any architecture, each context-aware application maintains five different knowledge containers, in which it can store and access the knowledge required for its operation. Dependencies among these knowledge containers allow for a reorganisation, maintenance and improvement of the overall application knowledge over time. This knowledge container view inspired the design of the four-layer software architecture of context-aware applications (cf. Section 4.4).

The conceptual framework of context-aware computing established by Chapter 4 formed the basis of the practical realization of the Context Management System that aims at facilitating the creation of adaptable context-aware applications. This Context Management System addresses the demands of all actors identified in Section 4.5 and comprises functionality and tools for the construction, authoring, maintenance and tailoring of context-aware behaviour. The system provides developers, who need to encode acquisition, derivation, adaptation, and

actuation knowledge in their specific implementation of a context-aware application, with a Context Toolkit that offers a software framework that comprises ready-to-use software components and guides developers through the software engineering process of context-aware applications (cf. Section 5.2). In addition, the Context Toolkit introduces a design view of context-aware applications that enable domain experts, who know about the adaptation capabilities of the implemented context-aware application, to gain control over the internals of the application. The design view allows for a tailoring and reconfiguration of the operational context-aware application without the need for reimplementation. The Context Management System further comprises a tool suite that stronger includes authors and end-users in the design, maintenance and tailoring of the implemented context-aware application (cf. Section 5.4).

Two case studies successfully designed, realised and deployed with the Context Management System revealed practical results and reflected experiences of developers, domain experts, authors and end-users with the application of the system and with operational applications. The implementation of a context-aware museum guide and an intelligent advertisement board proved the validity of the conceptual framework and demonstrated the combined utility of the programming techniques and the tool suite developed by this thesis. In addition, these two case studies illustrated issues involved in the development process of adaptable context-aware applications, highlighted the success of the Context Management System in involving a broad range of actors in this process, and validated the achieved degree of flexibility introduced by the design view regarding the modification of the context-aware behaviour at run-time

The lessons learnt during the application of the Context Management System uncovered new questions and issues that were summarized in Section 6.3.2. Some major aspects show potential for future work that is described in the subsequent section.

7.2 Future Work

A variety of potential extensions to the presented Context Management System and the corresponding research can be identified. For future work the variety of different context-aware applications based on the described tool suite and layer architecture is planned to be expanded in order to be able to evaluate the approach more thoroughly and to refine it. Furthermore, an investigation can be conducted on a possible extension of the design view on context-aware applications towards a complete description language for such applications. Aside from that, researchers such as Andreas Lorenz concentrate on a complete distribution of the architecture layers and the software components therein over a network (Lorenz, 2003; Lorenz, 2005).

The following paragraphs emphasise three research directions for context-aware computing originating from adaptable context-aware applications and the Context Management System. They describe steps towards the shared initiative between automatic and manual adaptation

activities with the aim of completely involve the end-user in the adaptation process performed by context-aware applications.

7.2.1 Reflection and Transparency Components

The users need to understand and inspect the mode of operation of the context-aware application. However, inspection requires the application to externalise its current state as well as its composition. Externalisation or reflection means the provision of a human-readable description of components (e.g. sensors) plugged into the system, information about the current state of these components (e.g. activated or deactivated), and configuration settings needed for launching and operating the system. Since acquisition and interpretation processes may result in imprecise or ambiguous information, the application needs to be able to provide feedback about potential error-prone processes, particularly when the consequences are important to the user.

The Context Toolkit offers partial transparency through the design view on context-aware applications. Furthermore, appropriate user interfaces can present this design view in a human-readable form. In addition, the Mobile Collector offers a runtime view on the context-aware application during its operation. The left panel of this tool displays current values of sensor components and context attributes (cf. Section 5.4.2).

The ability of the Context Management System to reflect the current state of the operational application should not be restricted to the use of the Mobile Collector, but extended to multiple displays and encompass various components of the application. Reflection and transparency components should cater for architectural support for the ability to provide feedback when something the user would want to know about is occurring as determined by the application developer. Based on these components, separate user interfaces can be built that allow for monitoring and control as well as a fine-tuning of the behaviour of the context-aware application. Thus, these extensions to the Context Management System improve the user experience in context-aware computing.

7.2.2 Retaining Adaptation

Humans interpret the same situation differently, and thus, several views on the same situation exist. The translation of this understanding to the field of context-aware computing leads to the conclusion that individual situations are of different relevance to the users. Therefore, a context-aware application should account for such inter-individual differences of the users and enable the users to give feedback to the entire adaptation process or parts of it in order to retain the process for traceability. Furthermore, user feedback constitutes an important link between contextualisation and personalisation.

User feedback can be discriminated in implicit and explicit feedback. Implicit feedback arises from an analysis of the user's behaviour and an implicit extraction or mining of information about the user. However, implicit feedback is always afflicted with a certain probability of

misjudging the user. Explicit feedback results from the user's need of explicitly communicating something to the application in any form. A context-aware application should consider both positive and negative feedback of the user in the entire control flow of the adaptation process because this feedback is directly associated with the performance of the system.

Some context-aware applications such as the museum guide described in Chapter 6 already process user-related information and base on implicit feedback because they already analyse the user's behaviour. However, an adaptive system should enable the user to express her need of communicating feedback to the context-aware application in any (explicit) form. This feedback can be provided as a numerical value and express the weight a user puts on a certain aspect of the adaptation process. The scope of this weight can potentially be any component of the context-aware application or any subprocess of the adaptation process.

Such a weighing expression produces a rating that indicates the suitability of the component functionality and allows for an inference about the importance of an adaptation for the user. Given a numerical weight in the range [0, 1], where an increasing value represents an increasing desirability, a context-aware application can embark on several strategies for changing its predetermined adaptive behaviour. An example illustrates a possible influence of a weighting for a context attribute: A context-aware application determines the user's location on the floor of an office building based on a WIFI tracking sensor. Low-quality values of this tracking sensor for the user's position complicate the determination of the office the user is located. Depending on the weight, the user assigned for the location context attribute, the context-aware application can proceed as follows: a low weight causes the application to cope with the determination of the location based on this low-quality position; a high weight triggers a mediation dialogue offering the user a selection list with best-guess choices for the location.

The Context Management System can easily accommodate this approach through an extension of the design view and an accordant adjustment of the software components behind. By means of this design view extension the user can specify her weight for every software component or programming abstraction that exhibits an accordant design view. The explicit specification of weights contributes to the user acceptance because the user can actively influence the decisions of the context-aware application. In turn, the developer can exploit these weights in the implementation of the context-aware application and obtains a means of adjusting the balance between application autonomy and user control at runtime.

7.2.3 Shared Initiative

The third aspect of the future work addresses the trade-off between automatic application adaptivity and user controlled adaptability (cf. Section 2.1.2). Context-aware applications already adapt automatically to the user's context, and adaptable context-aware applications provide a design view that users can customise according to their needs. However, context-aware computing hardly tackles substantial customisations of applications by end-users

through tailoring activities during usage. Developers of context-aware applications need to implement the shared initiative between adaptivity and adaptability in order to empower end-users without or with limited programming skills to customize or tailor the application according to their individual or context-specific requirements.

Following the concept of shared initiative, the completion of an adaptation process may comprise several alternations between instances of automatic and instances of manual adaptation methods. The adaptation methods described in Section 4.2.4 abstract from manual and automatic adaptation, and need to be instantiated separately for each, i.e. visually for manual and algorithmically for automatic adaptation. A finite set of applicable dialogue principles accomplish the manual realisation of an adaptation method. Such visual instantiations of adaptation methods allow a context-aware application (respectively its developer) to provide a certain degree of end-user involvement in the adaptation process.

An enhancement of the Context Toolkit consists in the separation of the adaptation process from the operating application. Following and extending the mediator approach presented by Dey and Mankoff (2005), the actual adaptation process can be passed to a mediator, once the context-aware application identifies the need for adaptation. This mediator either executes an adaptation automatically (automatic mediators) or allows the user and computer to communicate about this adaptation (manual mediator). After such a mediated adaptation process is completed, the result is returned to the initiator, the adaptation is retained and the execution process is handled back to the context-aware application again. This mediation model encapsulates information about how to achieve a certain adaptation of the application.

The developer of a context-aware application should indicate all potentials for adaptation in the code base, whether a need for initiating an adaptation process exists or not. If such an indicator for adaptation is reached during the operation of the context-aware application, the user's display can reflect this state and call the user's attention to an upcoming adaptation process. Thus, the end-user is always able to intervene in an automatically executed adaptation process or take the initiative and launch a manual adaptation process.

Furthermore, the selection of the appropriate mediator depends on the current context, the specification of the developer and the weighting of the user (see previous section). Thus, dynamically changing conditions lead to the selection of different types of mediators for the same adaptation process and determine the level of required or permitted user involvement (e.g. launching automatic or manual adaptations in cognitively more or less demanding situations).

The programming framework of the Context Management System needs to provide a basic set of rudimentary mediators. Thus, a basis mediator may simply be instantiated from this repository or a special mediator may explicitly be created through the extension of an existing mediator. Mediators always address a specific adaptation target as enumerated in Section 4.2.4 and perform a specific adaptation of the types described in the same section to this adaptation target. Along these two dimensions a mediator component might cover very simple adaptation functionality without any user interaction or highly complex adaptation tasks with

multiple user interactions. In addition, the operational context-aware application needs to provide a runtime environment for instantiated mediators.

The abstraction from the adaptation target and adaptation method allows for the development of mediators that are completely independent from the domain. This separation of the mediation process from the identification of an adaptation need and from the application operation leaves the basic structure of an application and its interface unmodified. The power of such mediation processes directly affects the content of the accomplishable adaptations.

References

Aamodt, A. and Plaza, E., 1994. *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches*. Artificial Intelligence Communications, 7(1): 39-52.

Addlesee, M. et al., 2001. *Implementing a Sentient Computing System*. IEEE Computer, 34(8): 50-56.

AKG, 2007. *AKG Acoustics GmbH - Homepage*. http://www.akg.com (access date: March 26th, 2007). AKG Acoustics GmbH, Vienna, Austria.

Amazon, 2007. *Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more*. http://www.amazon.com/ (access date: March 26th, 2007). Amazon.com, Inc., Seattle, Washington, US.

Anderson, S. et al., 2003. *Making Autonomic Computing Systems Accountable: The Problem of Human-Computer Interaction*, 14th International Workshop on Database and Expert Systems Applications. IEEE Computer Society, Prague, Czech Republic, pp. 718-724.

ARFF, 2007. *Attribute-Relation File Format (ARFF)*. http://www.cs.waikato.ac.nz/~ml/weka/arff.html (access date: March 26th, 2007). The University of Waikato, Department of Computer Science, Hamilton, New Zealand.

Asthana, A., Cravatts, M. and Krzyzanowski, P., 1994. *An Indoor Wireless System for Personalized Shopping Assistance*, IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, US, pp. 69-74.

AT&T, 2001. *Sentient Computing Project Home Page*. http://www.cl.cam.ac.uk/research/dtg/attarchive/spirit/ (access date: June 11th, 2007). AT&T Laboratories Cambridge.

Bäck, T., 1996. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, USA, 328 pp.

Bäck, T., Fogel, D. and Michalewicz, Z., 1997. *Handbook of Evolutionary Computation*. Oxford University Press, 988 pp.

Bardram, J.E., 2005a. *Design, Implementation, and Evaluation of the Java Context Awareness Framework (JCAF)*, Centre for Pervasive Healthcare, IT University of Copenhagen, Denmark.

Bardram, J.E., 2005b. *The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications*. In: H.-W. Gellersen, R. Want and A. Schmidt (Editors), Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive '05). Lecture Notes in Computer Science. Springer Verlag, Munich, Germany.

Bardram, J.E., 2005c. *Tutorial for the Java Context Awareness Framework (JCAF), version 1.5*, Centre for Pervasive Healthcare, IT University of Copenhagen, Denmark.

Barkhuus, L. and Dey, A.K., 2003. *Is Context-Aware Computing Taking Control away from the User? Three Levels of Interactivity Examined*, Ubiquitous Computing, vol. 2864 (Ubicomp 2003). Lecture Notes in Computer Science. Lecture Notes in Computer Science. Berlin: Springer-Verlag, pp. 149-156.

Barrett, K. and Power, R., 2003. *State of the Art: Context Management*, M-Zones Research Programme.

Beigl, M., Gellersen, H.-W. and Schmidt, A., 2001. *Mediacups: Experience with Design and Use of Computer-Augmented Everyday Artifacts*. Computer Networks, 35(4): 401-409.

Bellotti, V. and Edwards, W.K., 2001. *Intelligibility and Accountability: Human Considerations in Context Aware Systems*. Human Computer Interaction, 16(2-4): 193-212.

Belotti, R., 2004. *SOPHIE - Context Modelling and Control*. Diploma Thesis Thesis, Swiss Federal Institute of Technology Zurich, Zurich, 59 pp.

Benerecetti, M., Bouquet, P. and Ghidini, C., 2000. *Contextual Reasoning Distilled*. Journal of Experimental and Theoretical Artificial Intelligence (JETAI), 12(3): 279-305.

Bennett, F., Richardson, T. and Harter, A., 1994. *Teleporting - Making Applications Mobile*, IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, US, pp. 82-84.

Beringer, J., 2004. *End-User Development: Reducing Expertise Tension*. Communications of the ACM, 47(9): 39-40.

Berners-Lee, T., Hendler, J. and Lassila, O., 2001. *The Semantic Web*, Scientific American.

Biegel, G. and Cahill, V., 2004. *A Framework for Developing Mobile, Context-aware Applications*, Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom'04). IEEE Computer Society, Washington, DC, USA, Orlando, FL, USA, pp. 361-365.

Booch, G., Rumbough, J. and Jacobson, I., 1999. *The Unified Software Development Process*. Object Technology, Massachusetts, 463 pp.

Bormans, J. and Hill, K., 2002. *MPEG-21 Overview v.5*. International Organisation for Standardisation, ISO/IEC JTC1/SC29/WG11/N5231.

Bregman, A.S., 1994. *Auditory Scene Analysis: The Perceptual Organization of Sound*. MIT Press, Cambridge, Massachusetts, 792 pp.

Brézillon, P., 1999. *Context in Problem Solving: A Survey*. The Knowledge Engineering Review, 14(1): 1-34.

Brézillon, P., 2002. *Modeling and Using Context: Past, Present and Future*, Laboratoire d'Informatique de Paris.

Brézillon, P., 2003. *Using Context for Supporting Users Efficiently*. In: R.H. Sprague (Editor), Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS '03). IEEE, Los Alamitos, pp. CD Rom.

BroadVision, 2007. *www.broadvision.com*. http://www.broadvision.com/ (access date: March 26th, 2007). BroadVision, Inc., Redwood City, California, US.

Brown, P.J., 1996. *The Stick-e Document: A Framework for Creating Context-Aware Applications*. In: J.W. Sons (Editor), Proceedings of the International Conference on Electronic Documents, Document Manipulation, and Document Dissemination (EP 96). Origination, Dissemination, and Design (EP-ODD), New York, Palo Alto, CA., pp. 259-272.

Brown, P.J., 1998. *Triggering Information by Context*. Personal Technologies, 2(1): 1-9.

Brown, P.J., Bovey, J.D. and Chen, X., 1997. *Context-aware Applications: from the Laboratory to the Marketplace*. IEEE Personal Communications, 4(5): 58-64.

Brusilovsky, B., 1996. *Methods and Techniques of Adaptive Hypermedia*. User Modeling and User-Adapted Interaction, 6(2-3): 87-129.

Bulander, R., Decker, M., Kölmel, B. and Schiefer, G., 2005. *Enabling Personalized and Context Sensitive Mobile Advertising while Guaranteeing Data Protection*. In: I. Kushchu, Kuscu, M. H. (Editor), EURO mGOV 2005. Mobile Government Consortium International LLC, Brighton, UK, pp. 455-454.

Byun, H.E. and Cheverst, K., 2001. *Exploiting User Models and Context-Awareness to Support Personal Daily Activities*, Proceedings of the International Workshop on User Modelling for Context-Aware Applications, Sonthofen, Germany.

Candolin, C. and Kari, H., 2003. *An Architecture for Context Aware Management*, Proceedings of the Military Communications Conference (MILCOM '03). IEEE, Boston, Massachusetts, USA.

Cardinaels, K., Duval, E. and Olivi, H.J., 2006. *A Formal Model of Learning Object Metadata*, Proceedings of the Innovative Approaches for Learning and Knowledge Sharing, 1st European Conference on Technology Enhanced Learning (EC-TEL '06). Lecture Notes in Computer Science. Springer, Crete, Greece, pp. 74-87.

Chen, G. and Kotz, D., 2000. *A survey of context-aware mobile computing research*, Computer Science Department, Dartmouth College, Hanover, New Hampshire.

Chen, G. and Kotz, D., 2002a. *Context Aggregation and Dissemination in Ubiquitous Computing Systems*, 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), Callicoon.

Chen, G. and Kotz, D., 2002b. *Solar: An Open Platform for Context-Aware Mobile Applications*, Short Paper Proceedings of the 1st International Conference on Pervasive Computing, Zurich, Switzerland, pp. 41-44.

Chen, G., Li, M. and Kotz, D., 2004a. *Design and Implementation of a Large-Scale Context Fusion Network*. In: I.C. Society (Editor), Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous), pp. 246-255.

Chen, H., 2004. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. Ph.D. Thesis, University of Maryland, Baltimore County.

Chen, H., Finin, T. and Joshi, A., 2004b. *A Context Broker for Building Smart Meeting Rooms*, Proceedings of the Autonomous Systems Symposium. AAAI Spring Symposium. AAAI Press, Menlo Park, CA, Stanford, CA, pp. 53-60.

Chitarro, L., Ranona, R. and Ieronutti, L., 2003. *Guiding Visitors of Web3-dimensional Worlds through Automatically Generated Tours*, Proceedings of the 8th International Conference on 3D Web Technology.

Ciavarella, C. and Paterno, F., 2003. *Supporting Access to Museum Information for Mobile Visitors*, Proceedings of the 10th International Conference on Human-Computer Interaction, Crete, Greece.

Clark, H.H., 1996. *Using Language*, Cambridge, UK, 432 pp.

Clarke, P. and Cooper, M., 2000. *Knowledge Management and Collaboration*. In: U. Reimer (Editor), Proceedings of the 3rd International Conference on Practical Aspects of Knowledge Management, Basel.

Coutaz, J., Crowley, J.L., Dobson, S. and Garlan, D., 2005. *Context is key*. Communications of the ACM, 48(3): 49-53.

Coutaz, J. and Rey, G., 2002. *Foundations for a Theory of Contextors*, Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces (CADUI '02). Kluwer Academics Publishing, Valenciennes, France, pp. 283-302.

Crowley, J.L., Coutaz, J., Rey, G. and Reignier, P., 2002. *Perceptual Components for Context Aware Computing*, Proceedings of the International Conference on Ubiquitous Computing (UBICOMP '02), Goteborg, Sweden.

Cruz-Neira, C., Sandin, D.J. and Defanti, T.A., 1993. *Surroundscreen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*, Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques. ACM Press, New York, NY, USA, 135-142.

Culler, D.E. and Mulder, H., 2004. *Smart Sensors to Network the World*. Scientific American, 290(52-59).

Dagger, D., Wade, V. and Conlan, O., 2004. *Developing Adaptive Pedagogy with the Adaptive Course Construction Toolkit (ACCT)*, 2nd International Workshop on Authoring of Adaptive and Adaptable Educational Hypermedia (AH'04), Eindhoven, Netherlands.

Dayal, U. et al., 1988. *The HiPAC project: combining active databases and timing constraints*. ACM SIGMOD Record, 17(1): 51-70.

Dey, A.K., 2000. *Providing Architectural Support for Building Context-Aware Applications*. Ph.D. Thesis, Georgia Institute of Technology.

Dey, A.K., 2001. *Understanding and Using Context*. Personal Ubiquitous Computing, 5(1): 4-7.

Dey, A.K. and Abowd, G.D., 1999. *Towards a Better Understanding of Context and Context-Awareness*, College of Computing, Georgia Institute of Technology.

Dey, A.K. and Abowd, G.D., 2000. *CybreMinder: A Context-Aware System for Supporting Reminders*, 2nd International Symposium on Hendheld and Ubiquitous Computing. Lecture Notes in Computer Science. Springer, pp. 172-186.

Dey, A.K., Futakawa, M., Salber, D. and Abowd, G.D., 1999a. *The Conference Assistant: Combining Context-Awareness with Wearable Computing*, Proceedings of the 3rd International Symposium on Wearable Computers (ISWC), San Francisco, CA, USA, pp. 21-28.

Dey, A.K., Hamid, R., Beckmann, C., Li, I. and Hsu, D., 2004. *A CAPpella: Programming by Demonstration of Context-Aware Applications*, Proceedings of the Conference on Human Factors in Computing Systems (CHI '04). ACM Press, pp. 33-40.

Dey, A.K. and Mankoff, J., 2005. *Designing Mediation for Context-Aware Applications*. ACM Transactions on Computer-Human Interaction, 12(1): 53-80.

Dey, A.K., Salber, D. and Abowd, G.D., 2001. *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*. Human-Computer Interaction, Anchor article for Special Issue on Context-Awareness, 16(2-4): 97-166.

Dey, A.K., Salber, D., Futakawa, M. and Abowd, G.D., 1999b. *An Architecture To Support Context-Aware Applications*. GVU Technical Report GIT-GVU-99-23.

Dieterich, H., Malinowski, U., Kühme, T. and Schneider-Hufschmidt, M., 1993. *State of the Art in Adaptive User Interfaces*. In: M. Schneider-Hufschmidt, T. Kühme and U. Malinowski (Editors), Adaptive User Interfaces. North-Holland, Amsterdam, Netherlands, pp. 13-48.

Duda, R.O. and Hart, P.E., 1973. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, USA.

Ebling, M.R., Hunt, G.D.H. and Lei, H., 2001. *Issues for Context Services for Pervasive Computing*, Proceedings of the Workshop on Middleware for Mobile Computing, Heidelberg, Germany.

Eckel, G., 2001. *LISTEN - Augmenting Everyday Environments with Interactive Soundscapes*, Proceedings of the I3 Spring Days Workshop "Moving Between the Physical and the Digital: Exploring and Developing New Forms of Mixed Reality User Experience", Porto, Portugal.

Efstratiou, C., 2004. *Coordinated Adaptation for Adaptive Context-Aware Applications*. Ph.D. Thesis, Lancaster University, Lancaster, UK.

Efstratiou, C., Cheverst, K., Davies, N. and Friday, A., 2001. *An Architecture for the Effective Support of Adaptive Context-Aware Applications*, Proceedings of the 2nd International Conference on Mobile Data Management (MDM '01). Lecture Notes in Computer Science. Springer-Verlag, London, UK, pp. 15-26.

Felder, R.M. and Spurlin, J.E., 2005. *Applications, Reliability, and Validity of the Index of Learning Styles*. International Journal of Engineering Education, 21(1): 103-112.

Fink, J., 2004. *User Modeling Servers - Requirements, Design, and Evaluation*, 289. Akademische Verlagsgesellschaft Aka GmbH, Berlin.

Fischer, G., 2002. *Beyond 'Couch Potatoes': From Consumers to Designers and Active Contributors*. First Monday (Peer-Reviewed Journal on the Internet), 7(12).

Flintham, M. et al., 2003. *Where On-line Meets On-The-Streets: Experiences with Mobile Mixed Reality Games*, Proceedings of the Conference on Human Factors in Computing Systems (CHI '03), Ft. Lauderdale, Florida, USA, pp. 569-576.

FOLDOC, 2007. *FOLDOC - Computing Dictionary*. http://foldoc.org/ (access date: March 26th, 2007). Imperial College Department of Computing, London, UK.

Frese, M., 1987. *A Theory of Control and Complexity: Implications for Software-Design and Integration of Computer Systems into the Work Place*. In: M. Frese, E. Ulich and W. Dzida (Editors), Psychological Issues of Human-Computer Interaction in the Work Place. Elsevier Science (North Holland), Amsterdam, Netherlands, pp. 313-337.

Gardner, H., 1993. *Frames of Mind: Theory of Multiple Intelligences*. Fontana Press, 464 pp.

Gellersen, H.-W., Schmidt, A. and Beigl, M., 2002. *Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts*. ACM Journal on Mobile Networks and Applications (MONET), 7(5).

Gilb, T., 1998. *Principles of Software Engineering Management*, 464 pp.

Goiser, A.M.J., 1998. *Handbuch der Spread-Spectrum Technik*. Springer-Verlag Wien New York, pp. 152-158.

Gossmann, J. and Specht, M., 2002. *Location Models for Augmented Environments*. Personal and Ubiquitous Computing, 6(5-6): 334-340.

Gross, T. and Specht, M., 2001. *Awareness in Context-Aware Information Systems*. In: Oberquelle, Oppermann and Krause (Editors), Proceedings of the Mensch und Computer - 1. Fachübergreifende Konferenz. Teubner-Verlag, Bad Honnef, Germany, pp. 173-182.

Guha, R.V., 1995. *Contexts: A Formalization and Some Applications*. PhD thesis Thesis, Stanford University, Stanford, US.

Hansmann, U., Merk, L., Nicklous, M.S. and Stober, T., 2001. *Pervasive Computing Handbook*. Springer, Heidelberg, 409 pp.

Heckmann, D., 2005. *Ubiquitous User Modeling*. PhD Thesis, Saarland University, Saarbrücken, Germany.

Henderson, A. and Kyng, M., 1992. *There's no Place like Home: Continuing Design in Use*. In: J. Greenbaum and M. Kyng (Editors), Design at work: Cooperative Design of Computer Systems. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, pp. 219-240.

Henricksen, K., 2003a. *A Framework for Context-Aware Pervasive Computing Applications*. Ph.D. Thesis, University of Queensland, Queensland, Queensland.

Henricksen, K. and Indulska, J., 2006. *Developing Context-Aware Pervasive Computing Applications: Models and Approach*. Pervasive and Mobile Computing, 2(1): 37-64.

Henricksen, K., Indulska, J., McFadden, T. and Balasubramaniam, S., 2005. *Middleware for Distributed Context-Aware Systems*, International Symposium on Distributed Objects and Applications (DOA'05). Lecture Notes in Computer Science. Springer, pp. 846-863.

Henricksen, K., Indulska, J., Rakotonirainy, A., 2002. *Modeling Context Information in Pervasive Computing Systems*, Proceedings of the 1st International Conference on Pervasive Computing. Lecture Notes in Computer Science. Springer-Verlag, Zurich, pp. 167-180.

Henricksen, K., Indulska, J., Rakotonirainy, A., 2003b. *Generating Context Management Infrastructure from Context Models*, Proceedings of the 4th International Conference on Mobile Data Management (MDM) - Industrial Track, Melbourne, Australia.

Hertz, J., Krough, A. and Palmer, R.G., 1991. *Introduction to the Theory of Neural Computation*. Perseus Books Group, Redwood City, CA.

Hodgins, W. and Duval, E., 2002. *Draft Standard for Learning Object Metadata*. Learning Technology Standards Committee, IEEE.

Holmquist, L.E. et al., 2001. *Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts*. Lecture Notes in Computer Science, 2201: 116-?

Hong, J. and Landay, J., 2001. *An Infrastructure Approach to Context-Aware Computing*. Human-Computer Interaction, Special Issue on Context-Awareness, 16(2-4): 287-303.

Hopper, A., 1999. *Sentient Computing*. The Royal Society Clifford Paterson Lecture, 1999. Technical Report 1999.12., AT&T Laboratories, Cambridge.

Hull, R., Neaves, P. and Bedford-Roberts, J., 1997. *Towards Situated Computing*. In: B. Krulwich (Editor), The First International Symposium on Wearable Computers (ISWC '97), Cambridge, MA.

IRCAM, 2007. *WWW IRCAM: The Institute*. http://www.ircam.fr/institut.html (access date: March 26th, 2007). Institut de Recherche et Coordination Acoustique/Musique, Paris, France.

IrDA, 2007. *IrDA the Secure Wireless Link*. http://www.irda.org (access date: March 26th, 2007). IrDA, the Infrared Data Association, Walnut Creek, California, USA.

ISAS, 2007. *Institute of Sensor and Actuator Systems - Home*. http://www.isas.tuwien.ac.at (access date: March 26th, 2007). Institute of Sensor and Actuator Systems, Vienna University of Technology, Vienna, Austria.

ISO9241-11, 1998. *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) - Part 11: Guidance on Usability*. International Organization for Standardization.

ISO13407, 1999. *Human-Centred Design Processes for Interactive Systems*. International Organization for Standardization.

Jabber, 2007. *Jabber: Open Instant Messaging and a Whole Lot More, Powered by XMPP*. http://www.jabber.org (access date: March 26th, 2007). Jabber Software Foundation, Denver, US.

Jameson, A., 2003. *Systems that adapt to their users: An integrative Overview*, Tutorial presented at 9th International Conference on User Modelling, Johnstown, PA, USA.

Jarke, M., Klemke, R. and Nick, A., 2001. *Broker's Lounge - An Environment for Multi-Dimensional User-Adaptive Knowledge Management*, Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34), Maui, Hawaii.

Java, 2007. *Java Technology*. http://java.sun.com/ (access date: March 26th, 2007). Sun Microsystems, Inc., Santa Clara, CA, USA.

Jot, J.-M., 1999. *Real-Time Spatial Processing of Sounds for Music, Multimedia and Interactive Human-Computer Interfaces*. Multimedia Systems, 7: 55-69.

Kagal, L., Finin, T. and Joshi, A., 2003. *A Policy Language for a Pervasive Computing Environment*, IEEE 4h International Workshop on Policies for Distributed Systems and Networks (POLICY'03). IEEE Computer Society, Villa Olmo, Lake Como, Italy, pp. 63-74.

Karger, C. and Oppermann, R., 1991. *Empirische Nutzungsuntersuchung Adaptierbarer Schnittstelleneigenschaften*. In: Ackermann and Ulrich (Editors), Software-Ergonomie 91. Benutzerorientierte Softwareenwicklung, Sturrgart, Germany, pp. 272-280.

Kari, H. and Candolin, C., 2003. *Context Aware Management Architecture*, Commercial Information Technology for Military Operations (CITMO 2003), Ronneby, Ruotsi.

Katz, R.H., 1994. *Adaptation and Mobility in Wireless Information Systems*. Personal Communications, IEEE (first quarter), 1(1): 6-17.

Kiszka, J., 2004. *IrCOMM2k*. http://www.ircomm2k.de/ (access date: March 26th, 2007). Kiszka, Jan (GNU Public Licence). Free Software Foundation, Inc., Boston, MA, USA.

Klann, M., Eisenhauer, M., Oppermann, R. and Wulf, V., 2003. *Shared Initiative: Cross-Fertilisation Between Aituation Aware and Tailorable Systems*. In: C. Stephanidis (Editor), Universal Access in HCI. Lawrence Erlbaum Associates Mahwah, pp. 562 - 566.

Klemke, R., 2002. *Modelling Context in Information Brokering Processes*. PhD Thesis, RWTH Aachen, Aachen.

Klemke, R. and Koenemann, J., 1999. *Supporting Information Brokers with an Organisational Memory*, 5. Deutsche Tagung Wissensbasierte Systeme - Bilanz und Perspektiven, Workshop Wissensmanagement und Organisational Memory (XPS-99), Würzburg, Germany.

Kobsa, A., 1993. *User Modeling: Recent Work, Prospects and Hazards*. In: M. Schneider-Hufschmidt, T. K?hme and U. Malinowski (Editors), Adaptive User Interfaces: Principles and Practice. North-Holland, Amsterdam, Netherlands, pp. 111-128.

Kobsa, A., 2001. *Generic User Modeling Systems*. User Modeling and User-Adapted Interaction, 11(1-2): 49-63.

Kowalsky, R., 1986. *A Logic-Based Calculus of Events*. New Generation Computing, 4: 67-95.

Kravcik, M., Kaibel, A., Specht, M. and Terrenghi, L., 2004. *Mobile Collector for Field Trips*. Educational Technology and Society, 7(2): 25-33.

Kriste, T., 2001. *Situation-Aware Mobile Assistance*. In: R. Earnshaw, R. Guedj, A. van Dam and J. Vince (Editors), Frontiers of Human-Centred Computing, Online Communities and Virtual Environments. Lecture Notes in Computer Science. Springer-Verlag, London, UK, pp. 99-115.

Kröner, A., 2001. *Adaptive Layout of Dynamic Web Pages*, 248. Akademische Verlagsgesellschaft Aka GmbH, Berlin.

Lave, J. and Wenger, E., 1991. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, Cambridge, UK, 138 pp.

Lei, H., Sow, D.M., Davis, J.S., Banavar, G. and Ebling, M.R., 2002. *The Design and Applications of a Context Service*. ACM SIGMOBILE Mobile Computing and Communications Review, 6(4): 45-55.

Lenat, D.B., 1998. *The Dimensions of Context-Space*, Cycorp, Austin (Texas), US.

Lieberman, H., 2001. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann; 1st edition (February 27, 2001), San Francisco, 448 pp.

Lieberman, H., Paternó, F. and Wulf, V., 2006. *End User Development*. Springer, Berlin, Germany, 492 pp.

Lieberman, H. and Selker, T., 2000. *Out of context: Computer Systems that Adapt to, and Learn from, Context*. IBM Systems Journal, 39(3&4): 617-631.

LISTEN, 2006. *LISTEN - Augmenting Everyday Environments Through Interactive Soundscapes*. LISTEN Project Consortium 2001.

Long, S., Aust, D., Abowd, G.D. and Atkeson, C.G., 1996. *Cyberguide: Prototyping ContextAware Mobile Applications*, Proceedings of the Conference on Human Factors in Computing Systems (CHI '96). ACM Press, NewYork, Vancouver, British Columbia, Canada, pp. 293-294.

Lorenz, A., 2003. *Agent-Based User Modeling for Ubiquitous Computing*. In: L. Ardissono, P. Brna and A. Mitrovic (Editors), Proceedings of the 10th International Conference on User Modeling (UM2005). Lecture Notes in Computer Science. Springer, Edinburgh, UK, pp. 512-514.

Lorenz, A., 2005. *A Specification for Agent-Based Distributed User Modelling in Ubiquitous Computing*. In: P. Dolog and J.I. Vassileva (Editors), Proceedings of the 1st Workshop on Decentralized, Agent Based and Social Approaches to User Modelling (DASUM2005). Lecture Notes in Computer Science. Springer Verlag Berlin Heidelberg, Edinburgh, UK, pp. 31-40.

Lorenz, A., Schmitt, C., Oppermann, R., Eisenhauer, M. and Zimmermann, A., 2005. *Location and Tracking in Mobile Guides*, Proceedings of the 4th Workshop on HCI in Mobile Guides, Salzburg, Austria.

Marmasse, N. and Schmandt, C., 2000. *Location-Aware Information Delivery with ComMotion*, 2nd International Symposium on Handheld and Ubiquitous Computing, Bristol, UK, pp. 157-171.

Mathias, A., 2001. *SmartReminder: A Case Study on Context-Sensitive Applications*, Dartmouth College Computer Science, Hanover, NH, USA.

McCarthy, B. and McCarthy, D., 2005. *Teaching Around the 4MAT® Cycle: Designing Instruction for Diverse Learners with Diverse Learning Styles*. Corwin Press, 120 pp.

McCarthy, J., 1993. *Notes on Formalizing Contexts*. In: T. Kehler and S. Rosenschein (Editors), Proceedings of the 5th National Conference on Artificial Intelligence (IJCAI '93). Morgan Kaufmann, Los Altos, California, pp. 555-560.

McCarthy, J. and Buvac, S., 1994. *Formalizing Context (Expanded Notes)*, Stanford, CA, USA.

Meier, R. and Cahill, V., 2003. *Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications*, Proceedings of the 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03). Lecture Notes in Computer Science. Springer-Verlag Heidelberg, Germany, Paris, France, pp. 285-296.

Mitchell, K., 2002. *A Survey of Context-Awareness*, University of Lancaster, Lancaster, UK.

Mørch, A., 1997. *Three Levels of End-user Tailoring: Customization, Integration, and Extension*. In: M. Kyng and L. Mathiassen (Editors), Computers and Design in Context. MIT Press, Cambridge, MA, USA, pp. 51-76.

MySQLCC, 2007. *MySQL AB: MySQL Control Center*. http://www.mysql.com (access date: March 26th, 2007). MySQL Inc., Cupertino, CA, USA.

Newberger, A. and Dey, A.K., 2003. *Designer Support for Context Monitoring and Control*, Intel Research Berkeley Technical Report IRB-TR-03-017.

Oppermann, R., 1994. *Adaptively supported Adaptability*. International Journal of Human-Computer Studies, 40(3): 455-472.

Oppermann, R., 2005. *From User-Adaptive to Context-Adaptive Information Systems*. i-com Zeitschrift für interaktive und kooperative Medien, 4(3): 4-14.

Oppermann, R. and Specht, M., 1999. *A Nomadic Information System for Adaptive Exhibition Guidance*. Archives and Museum Informatics. Cultural Heritage Informatics Quarterly, 13(2): 127 - 138.

Oppermann, R. and Specht, M., 2000. *A Context-sensitive Nomadic Information System as an Exhibition Guide*, Proceedings of the Second Symposium on Handheld and Ubiquituous Computing (HUC 2000). Lecture Notes In Computer Science. Springer-Verlag, London, UK, Bristol, UK, pp. 127-142.

Oppermann, R. and Thomas, C.G., 1996. *Supporting Learning as an Iterative Process in a Social Context*. In: P. Brna, A. Paiva and J. Self (Editors), Proceedings of the European Conference on Artificial Intelligence in Education, Lisboa, Portugal, pp. 150-156.

Öztürk, P. and Aamodt, A., 1998. *A Context Model for Knowledge-Intensive Case-Based Reasoning*. Special Issue on Using Context in Applications. International Journal on Human-Computer Studies, 48(3): 331-355.

Padovitz, A., Loke, S.W., Zaslavsky, A., Bartolini, C. and Burg, B., 2005. *An Approach to Data Fusion for Context Awareness*, Proceedings of the International and Interdiciplinary Conference on Modeling and Using Context. Lecture Notes in Computer Science. Springer, Berlin, Germany, Paris, France, pp. 353-367.

Pascoe, J., 1997. *The Stick-e Note Architecture: Extending the Interface Beyond the User*, Proceedings of the 2nd International Conference on Intelligent User Interfaces. ACM Press, New York, NY, USA, Orlando, Florida, USA, pp. 261-264.

Pascoe, J., 1998. *Adding Generic Contextual Capabilities to Wearable Computers*, 2nd IEEE International Symposium on Wearable Computers (ISWC'98). IEEE, Pittsburgh, PA, pp. 92-99.

Pazzani, M. and Billsus, D., 1997. *Learning and Revising User Profiles: The Identification of Interesting Web Sites*. Machine Learning, 27(3): 313-331.

Petrelli, D. and Not, E., 2006. *User-centred Design of Flexible Hypermedia for a Mobile Guide: Reflections on the HyperAudio Experience*. User Modeling and User-Adapted Interaction (UMUAI), 15(3-4): 303-338.

RAFT, 2003. *Remotely Accessible Field Trips. http://www.raft-project.net/*. RAFT 2003.

Rey, G. and Coutaz, J., 2004a. *The Contextor Infrastructure for Context-Aware Computing*, Proceedings of the 18th European Conference on Object-Oriented Programming (ECOOP04), Workshop on Component-Oriented Approach to Context-Aware Systems, Oslo, Norway.

Rey, G. and Coutaz, J., 2004b. *Contextor: Capture and Dynamic Distribution of Contextual Information*, Proceedings of the 1st French-Speaking Conference on Mobility and Ubiquity Computing (UbiMob '04). ACM Press, New York, NY, USA, Nice, France, pp. 131-138.

Rhodes, B.J., 1997. *The Wearable Remembrance Agent: A System for Augmented Memory*. Personal Technologies Special Issue on Wearable Computing, 1(1): 218-224.

Rich, E., 1989. *Stereotypes and User Modeling*. In: A. Kobsa and W. Wahlster (Editors), User Models in Dialog Systems. Springer Verlag, Berlin, Heidelberg, pp. 35-51.

Richter, M.M., 1995. *The Knowledge Contained in Similarity Measures*, Invited Talk at the First International Conference on Case-Based Reasoning (ICCBR'95), Sesimbra, Portugal.

Ryan, N., 1999. *ConteXtML: Exchanging Contextual Information between a Mobile Client and the FieldNote Server*.

Ryan, N., Pascoe, J. and Morse, D., 1998. *Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant*. In: V. Gaffney, M. van Leusen and S. Exxon (Editors), Computer Applications in Archaeology 1997, British Archaeological Reports, Oxford.

Salber, D., 2000. *Context-Awareness and Multimodality*, Proceedings of First Workshop on Multimodal User Interfaces, Grenoble, France.

Salber, D., Dey, A.K. and Abowd, G.D., 1999. *The Context Toolkit: Aiding the Development of Context-Enabled Applications*, Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '99). ACM Press, New York, Pittsburgh, Pennsylvania, USA, pp. 434-441.

Satyanarayanan, M., 2003. *Of Smart Dust and Brilliant Rocks*. Pervasive Computing, 2(2-4).

Schilit, B.N., 1995. *A System Architecture for Context-Aware Mobile Computing*. Ph.D. Thesis, Columbia University, New York, NY, US.

Schilit, B.N., Adams, N.I. and Want, R., 1994. *Context-Aware Computing Applications*, Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, pp. 85-90.

Schilit, B.N. and Theimer, M., 1994. *Disseminating Active Map Information to Mobile Hosts*. IEEE Network, 8(5): 22-32.

Schilit, B.N., Theimer, M. and Welch, B.B., 1993. *Customizing Mobile Applications*, USENIX Symposium on Mobile & Location-Independent Computing. USENIX Association, pp. 129-138.

Schmidt, A., 2000. *Implicit Human Computer Interaction Through Context*. Personal Technologies, 4(2&3): 191-199.

Schmidt, A., 2002. *Ubiquitous Computing Computing in Context*. Ph.D. Thesis, Lancaster University, Lancaster, U.K.

Schmidt, A. et al., 1999. *Advanced Interaction in Context*. In: H.W. Gellersen (Editor), Proceedings of the First International Symposium on Handheld and Ubiquitous Computing (HUC '99). Lecture Notes in Computer Science. Springer Verlag, Heidelberg, Karlsruhe, Germany, pp. 89-101.

Schmidt, A., Beigl, M. and Gellersen, H.-W., 1999. *There is more to Context than Location*. Computers & Graphics Journal, Elsevier, 23(6): 893-902.

Schmidt, A., Takaluoma, A. and Mäntyjärvi, J., 2000. *Context-Aware Telephony Over WAP*. Personal Technologies, 4(4): 225-229.

Schmidt, A. and van Learhoven, K., 2001. *How to Build Smart Appliances?* IEEE Personal Communications, Special Issue und Pervasive Computing, 8(4): 66-71.

Smith, M.K., Welty, C. and McGuinness, D., 2004. *OWL Web Ontology Language Guide*. http://www.w3.org/TR/owl-guide/ (access date: March 26th, 2007). World Wide Web Consortium (W3C), Cambridge, MA, USA.

Sohn, T. and Dey, A.K., 2003. *iCAP: An Informal Tool for Interactive Prototyping of Context-Aware Applications*, Extended Abstracts of Computer-Human Interaction on Human Factors in Computing Systems (CHI 2003). ACM Press, Fort Lauderdale, Florida, US, pp. 974-975.

Sørensen, C.-F. et al., 2004. *A Context-Aware Middleware for Applications in Mobile ad-hoc Environments*, 2nd Workshop on Middleware for Pervasive and ad-hoc Computing (MPAC ’04). ACM Press, New York, NY, USA, pp. 107-110.

Specht, M., Kaibel, A. and Apelt, S., 2005. *Extending LCMS for Remote Accessible Field Trips in RAFT*, Proceedings of the 3rd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops) on Pervasive eLearning - PerEL '05. IEEE Computer Society, Kauai Island, HI, USA, pp. 302-306.

Specht, M. and Kravcik, M., 2006. *Authoring of Learning Objects in Context*. International Journal on E-Learning, 5(1): 25-33.

Specht, M., Lorenz, A. and Zimmermann, A., 2006. *An Architecture for Contextualized Learning Experiences*, Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies. IEEE Computer Society, Washington, DC, USA, Kerkrade, Netherlands, pp. 169-173.

Specht, M. and Oppermann, R., 1998. *ACE - Adaptive Courseware Environment*. The New Review of Hypermedia and Multimedia, 4(1): 141-161.

Stahl, C. and Heckmann, D., 2004. *Using Semantic Web Technology for Ubiquitous Location and Situation Modeling*. Journal of Geographic Information Sciences CPGIS, 10(2): 157-165.

Strang, T. and Linnhoff-Popien, C., 2004. *A Context Modelling Survey*, Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004, Nottingham, UK.

Thesaurus, 2007. *thesaurus.com*. http://www.thesaurus.com (access date: March 26th, 2007). Lexico Publishing Group, LLC, Los Angeles, CA, USA.

Tramberend, H., 1999. *Avango: A Distributed Virtual Reality Framework*, Proceedings of the IEEE Virtual Reality '99 Conference, Houston, Texas, US, pp. 14-21.

Unnützer, P., 2001. *LISTEN im Kunstmuseum Bonn*. KUNSTFORUM International, 155: 469-470.

van Learhoven, K., Aidoo, K.A. and Lowette, S., 2001. *Real-Time Analysis of Data from Many Sensors with Neural Networks*, 5th International Symposium on Wearable Computing (ISWC), Zürich, pp. 115-122.

Vassileva, J.I., 1996. *A Task-Centered Approach for User Modeling in a Hypermedia Office Documentation System*. User Modeling and User-Adapted Interaction (UMUAI), 6: 185-223.

Véron, E. and Levasseur, M., 1983. *Ethnographie de l'expositionl'exposition: l'espace, le corps, le sens*, Paris, Bibliothèque publique d'Information, Centre Georges Pompidou.

Wahlster, W. and Kobsa, A., 1989. *User Models in Dialog Systems*. In: A. Kobsa and W. Wahlster (Editors), User Models in Dialog Systems. Springer-Verlag, Heidelberg-Berlin, pp. 472.

Wakkary, R. and Hatala, M., 2006. *ec(h)o: Situated Play in a Tangible and Audio Museum Guide*, Proceedings of the 6th ACM conference on Designing Interactive systems, Symposium on Designing Interactive Systems. ACM Press, New York, NY, USA, University Park, PA, USA, pp. 281-290.

Want, R., Hopper, A., Veronica, F. and Gibbons, J., 1992. *The Active Badge Location System*. ACM Transactions on Information Systems, 10(1): 91-102.

Watson, I., 1998. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 289 pp.

Weber, G., 2002. *Interview mit Gerhard Weber*, Künstliche Intelligenz (KI), Bremen, pp. 37-41.

Weiser, M., 1991. *The Computer for the Twenty-First Century*. Scientific American, 265(3): 94-104.

Weiser, M., 1994. *The World is not a Desktop*. Interactions, 1(1): 7-8.

Weka, 2007. *Weka 3 - Data Mining with Open Source Machine Learning Software in Java*. http://www.cs.waikato.ac.nz/~ml/weka/index.html (access date: March 26th, 2007). The University of Waikato, Department of Computer Science, Hamilton, New Zealand.

Wenzel, E., 1998. *The Impact of System Latency on Dynamic Performance in Virtual Acoustic Environments*, 5th International Congress on Acoustics and 135th Meeting of the Acoustical Society of America, Seattle, WA, pp. 2405-2406.

Wikipedia, 2007. *Wikipedia, the free encyclopedia*. http://en.wikipedia.org/ (access date: March 26th, 2007). Wikimedia Foundation, Inc., St. Petersburg, FL, USA.

Winograd, T., 2001. *Architectures for Context*. Human-Computer-Interaction, Special Issue on Context-Aware Computing, 16(2-4).

Wulf, V., 2000. *Exploration Environments: Supporting Users to Learn Groupware Functions*. Interacting with Computers, 13(2): 265-299.

Wulf, V. and Golombek, B., 2001. *Direct Activation: A Concept to Encourage Tailoring Activities*. Behaviour and Information Technology, 20(4): 249-263.

Xerces, 2007. *xml.apache.org*. http://xml.apache.org/ (access date: March 26th, 2007). The Apache Software Foundation, Forest Hill, MD, USA.

Yan, H. and Selker, T., 2000. *Context-Aware Office Assistant*, Proceedings of the 5th International Conference on Intelligent User Interfaces. ACM Press, New York, NY, USA, New Orleans, Louisiana, US, pp. 276-279.

Zadeh, L.A., 1965. *Fuzzy Sets*. Information and Control, 8(3): 338-353.

Zimmermann, A., Lorenz, A. and Specht, M., 2002. *Reasoning From Contexts*. In: N. Henze (Editor), Personalization for the Mobile World: Workshop Proceedings on Adaptivity and User Modeling in Interactive Systems (ABIS2002), Hannover, Germany, pp. 114-120.

Zimmermann, A., Lorenz, A. and Specht, M., 2003a. *The Use of an Information Brokering Tool in an Electronic Museum Environment*. In: D. Bearman and J. Trant (Editors), Proceedings of the International Conference about Museums and the Web (MW2003). Archives & Museum Toronto, Canada, Charlotte, NC, USA, pp. 217-225.

Zimmermann, A., Lorenz, A. and Specht, M., 2003b. *User Modeling in Adaptive Audio-Augmented Museum Environments*. In: P. Brusilowsky, A. Corbett and F. de Rosis (Editors), Proceedings of the 9th international Conference on User Modeling (UM-03). Lecture Notes in Computer Science. Springer Verlag Berlin Heidelberg, Johnstown, PA, USA, pp. 403-407.

Appendix A

EBNF Notation of the Design View

This appendix defines the syntax of the markup language used for the configuration of the adaptable context-aware application constructed with the Context Management System. A syntax comprises a set of rules that guide the construction of correct language expressions and that in turn allows a verification of such expressions. As a type of meta-language, the Extended Backus-Naur Form (EBNF) constitutes a popular means of the definition of syntaxes for programming and markup languages. The EBNF bases the syntax definition on so-called productions, i.e. rules that describe a specific fragment of the syntax. The totality of all such syntactical rules describes a language. The general principle of these productions consists in the replacement of elements of a string, words or sentences by other strings, words or sentences.

The markup language employed for the configuration of the software components of an adaptable context-aware application rests upon the syntax of XML. This basis allows the application of standard parsers such as Xerces (2007) by the Apache Software Foundation for the parsing of the markup documents. The remainder of this section specifies the EBNF grammar of the configuration markup language of the Context Management System.

Sensors

```
sensor-document ::= “<” “SENSORS” [sensor-fabric-def] “>”
                          {sensor-def}
                    “</” “SENSORS” “>”;
sensor-fabric-def ::= class-def;
sensor-def ::= “<” sensor-name sensor-attributes “/>”;
sensor-name ::= symbol;
sensor-attributes ::=  sensor-class-def [entity-id-def] [device-ip-def]
                       [sensor-server-ip-def] {optional-attributes-def};
entity-id-def ::= “ENTITY_ID” “=” “”” symbol “””;
device-ip-def ::= “DEVICE_IP” “=” “”” ip-address “””;
```

```
optional-attributes-def ::= symbol "=" "”" string "”";
sensor-server-ip-def ::= "SENSOR_SERVER_IP" "=" "”" ip-address "”";
sensor-class-def ::= class-def;
```

Attributes

```
attribute-document ::= "<" "ATTRIBUTES" [attribute-fabric-def] ">"
                          {attribute-def}
                       "</" "ATTRIBUTES" ">";
attribute-fabric-def ::= class-def;
attribute-def ::= attribute-def-simple | attribute-def-complex;
attribute-def-simple ::= "<" attribute-name attribute-properties "/>";
attribute-def ::= "<" attribute-name attribute-properties ">"
                          value-providers
                  "</" "attribute-name" ">";
attribute-name ::= symbol;
attribute-properties ::=  attribute-class-def
                          ["CONTEXT_CHANGE" "=" "”" boolean "”"]
                          ["INIT_VALUE" "=" "”" string "”"]
                          ["USER_PREFERENCE" "=" "”" number "”"];
attribute-class-def ::= class-def;
value-providers ::= sensor-providers | attribute-providers;
sensor-providers ::= "<" "SENSORS" ">" {sensor-ref}+ "</" "SENSORS" ">";
sensor-ref ::= "<" sensor-name "/>";
attribute-providers ::=  "<" "ATTRIBUTES" ">"
                               {attribute-ref}+
                         "</" "ATTRIBUTES" ">";
attribute-ref ::= "<" attribute-def "/>";
context-name ::= symbol;
```

Context Collection

```
context-collection-document ::=
                    "<" context-collection-name [context-collection-fabric-def] ">"
                          {context-def}
                    "</" context-collection-name ">";
context-collection-name ::= symbol;
context-collection-fabric-def ::= class-def;
context-def ::=  "<" context-name ">"
                     [history] attribute-document
                 "</" context-name ">";
```

History

```
history ::= persistence-def cache-def history-def;
persistence-def ::= “<” “PERSISTENCE” adapter-def “/>”;
adapter-def ::= “ADAPTER” “=” “"” class-def “"”;
cache-def ::= “<” “CACHE” cache-size-def “/>”;
cache-size-def ::= “SIZE” “=” “”” digit “””;
history-def ::= “<” “HISTORY” history-attributes “/>”;
history-attributes ::= “PERSISTENCE” “=” “”” persistence-options “””
                       “EVENT_TYPE” “=” “”” event-type-options “””
                       “TICK” “=” “”” milliseconds “””;
persistence-options ::= “FULL” | “ON_OVERFLOW” | “NONE”;
event-type ::= “SYNCHRONOUS” | “ASYNCHRONOUS” | “ON_CONTEXT_CHANGE”;
milliseconds ::= digit;
```

References

```
reference ::= entity-reference | context-attribute-reference | memory-reference |
value-reference;
entity-reference ::=
             “<” “REFERENCE” “ENTITY” “=” “”” (“QUERY” | “RESULT” | entity-def) “””;
entity-def ::= entity-type “.” entity-id;
entity-type ::= symbol;
entity-id ::= symbol;
context-attribute-reference ::=
             “<” “REFERENCE”  “ATTRIBUTE” “=” “”” attribute-def “”” “/>”;
attribute-def ::=
        [entity-def “.”] [context-name “.”] [history-position “.”] attribute-name;
history-position ::= number;
memory-reference ::=   “<” “REFERENCE” “MEMORY” “=”
                          “”” memory-type “.” variable-name “””
                       “/>”;
memory-type ::= “LOCAL” | “GLOBAL” | “SYSTEM”;
value-reference ::= “<” “REFERENCE” “VALUE” “=” “”” value “”” “/>”;
value ::= number | symbol | string;
variable-name ::= symbol;
```

Boolean Qualifier

```
boolean-qualifier-document ::=  “<” “BOOLEAN_QUALIFIERS” “>”
                                     {boolean-qualifier-def}
                                “</” “BOOLEAN_QUALIFIERS” “>”;
boolean-qualifier-def ::=       “<” boolean-qualifier-name “>”
                                     boolean-qualifier
                                “</” boolean-qualifier-name “>”;
boolean-qualifier-name ::= symbol;
boolean-qualifier ::= concatenation | predicate-def | boolean-qualifier-ref;
boolean-qualifier-ref ::= “<” boolean-qualifier-name [inversion] “/>”;
concatenation ::= conjunction | disjunction;
conjunction ::=         “<” “AND” “>”
                           boolean-qualifier {boolean-qualifier}+
                        “</” “AND” “>”;
disjunction ::=         “<” “OR” “>”
                           boolean-qualifier {boolean-qualifier}+
                        “</” “OR” “>”;
predicate-def ::= predicate-start predicate-references predicate-end;
predicate-start ::= “<” predicate predicate-attributes “>”;
predicate-attributes ::= predicate-type [inversion];
predicate-type ::= “TYPE” “=” “” type “”;
type ::= “NUMBER” | “SYMBOL” | “BOOLEAN” | “TIME”;
inversion ::= “INVERT” “=” “” boolean “”;
predicate-end ::= “</” predicate-name “>”;
predicate ::= “EQUALS” | “GREATER” | “LESS” | “GREATER_OR_EQUAL” | “LESS_OR_EQUAL”
| “IS_LIKE” | “NOT_EQUAL” | “CONTAINS” | “CASE_SENSITIVE_LIKE” | “IS_NULL”;
predicate-references ::= left-reference right-reference;
left-reference ::= reference;
right-reference ::= reference;
```

Control Rules

```
control-rules-document ::=   “<” “CONTROL_RULES” “>”
                                {rule-def}
                             “</” “CONTROL_RULES” “>”;
rule-def ::= rule-ref | parameterized-rule;
rule-ref ::= “<” rule-name “/>”;
parameterized-rule ::= “<” rule-name “>” rule “</” rule-name “>”;
rule-name ::= symbol;
rule ::= precondition-def actions-def;
```

```
precondition-def ::= “<” “PRECONDITION” “>” {precondition} “</” “PRECONDITION” “>”;
precondition ::= boolean-qualifier;
actions-def ::= “<” “ACTIONS” “>” {action}+ “</” “ACTIONS” “>”;
action ::= “<” action-name “>” parameter-def “</” action-name “>”;
parameter-def ::= “<” “PARAMETERS” “>” {parameter}+ “</” “PARAMETERS” “>”;
parameter ::= “<” parameter-name “>” reference “</” parameter-name “>”;
```

Actuators

```
actuator-document ::=  “<” “ACTUATORS” [actuator-fabric-def] “>”
                               {actuator-def}
                       “</” “ACTUATORS” “>”;
actuator-fabric-def ::= “CLASS” “=” “”” class-def “””;
actuator-def ::= “<”actuator-name actuator-attributes “/>”;
actuator-name ::= symbol;
actuator-attributes ::= actuator-class-def
                     [entity-id-def]
                     [device-ip-def]
                     [actuator-server-ip-def]
                     [parameters-def]
                     {optional-attributes-def};
entity-id-def ::= “ENTITY_ID” “=” “”” symbol “””;
device-ip-def ::= “DEVICE_IP” “=” “”” ip-address “””;
actuator-server-ip-def ::= “ACTUATOR_SERVER_IP” “=” “”” ip-address “””;
parameters-def ::= “PARAMETERS” “=” “”” symbol {“,” symbol} “””;
optional-attributes-def ::= symbol “=” “”” string “””;
actuator-class-def ::= class-def;
```

Filters

```
filter-document ::= “<” “FILTERS” “>” {filter-def} “</” “FILTERS” “>”;
filter-def ::= “<” filter-name “>” boolean-qualifier “</” filter-name “>”;
filter-name ::= symbol;
```

Situations

```
situation-document ::= “<” “SITUATIONS” “>” {situations-def} “</” “SITUATIONS” “>”;
situation-def ::=      “<” situation-name trigger-def “>”
                               boolean-qualifier
                       “</” situation-name “>”;
trigger-def ::= “TRIGGERS” “=” “”” attribute-ref-simple “””;
situation-name ::= symbol;
```

Stereotypes

```
stereotype-document ::=   “<” “STEREOTYPES” “>”
                               {stereotype-hierarchy-def}
                          “</” “STEREOTYPES” “>”;
stereotype-hierarchy-def ::=   “<” stereotype-hierarchy-name “>”
                                    {stereotype-def}
                               “</”stereotype-hierarchy-name “>”;
stereotype-def ::=     “<” stereotype-name [extends]“>”
                          boolean-qualifier
                       “</”stereotype-name “>”;
extends ::= “EXTENDS” “=” “”” stereotype-name “””;
stereotype-name ::= symbol;
```

Atoms

```
boolean ::= true | false;
true ::= “TRUE”;
false ::= “FALSE”;
number ::= [“+” | “-”] {digit}+ [“.”{digit}+];
digit ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7”| “8” | “9”;
ip-address ::= {digit}+ “.” {digit}+ “.” {digit}+ “.” {digit}+;
string ::= {char-extended}+;
char ::= digit | “A” | … | “Z” | “a” | … | “z”;
char-extended ::= char | “ ” | “,” | “*” | ... ;
symbol ::= {char | “-” | “_” }+;
class-def ::= symbol { “.” symbol };
```

Appendix B

Curriculum Vitae

Particulars

Name:	Andreas Zimmermann
Place of Residence:	Verdistr. 35 67227 Frankenthal
Telephone:	+49 179 7824988
E-Mail:	Andreas.Zimmermann@fit.fraunhofer.de
Date of Birth:	07/19/1973
Place of Birth	Ludwigshafen am Rhein
Nationality:	German
Marital Status:	Unmarried

Education

1993-2001	Diploma in Computer Science and Electrical Engineering, University of Kaiserslautern, Germany (A / 1.0)
1984-1993	Grammar school in Frankenthal (Karolinengymnasium), Germany A-Levels in Mathematics, Physics, Latin (B / 2.0)
1980-1984	Primary school in Frankenthal/Eppstein, Germany

Professional Experience

Since March 2002	Research associate for the *Fraunhofer Institute for Applied Information Technology* in the department "Information in Context" in Sankt Augustin, Germany

10/23/2001 - 02/28/2002	Freelance consultant for *T-Systems* in Aachen, Germany
04/01-10/22/2002	Product-Innovation Manager for *TRAIAN Internet Products* in Bonn, Germany (Petition in bankruptcy in October 2002)

Practical Experience

01/08 - 04/23/1999	Internship at *Siemens Business Solutions* in Boston, Massachusetts, US
09/15 - 12/31/1998	Internship at *SAP AG* Walldorf in the department "German Marketing"
04/22 - 07/10/1997	Practical work "Artificial Intelligence" at the research group for *Artificial Intelligence and Knowledge-Based Systems* of the University of Kaiserslautern
04/15 - 07/15/1997	Practical work "Developing Communication Systems" at the research group for *Computer Networks* of the University of Kaiserslautern
04/17 - 07/10/1997	Practical work "Computer Design" at the research group for *VLSI-Design and Architecture* of the University of Kaiserslautern
10/20/1996 - 02/15/1997	Practical work "Case-Based Reasoning" at the research group *Artificial Intelligence and Knowledge-Based Systems group* of the University of Kaiserslautern
08/01/1996 - 03/01/1999	Student assistant for the *University of Kaiserslautern* in the research group *Artificial Intelligence and Knowledge-Based Systems* - Project "INRECA"
09/02 - 10/15/1996	Practical work "Automation Technique and Robotics" in the department "Automation" of the *BASF Ludwigshafen*, Germany

Extracurricular Jobs

1997 - 2002	Tourists' guide around Europe for *RuF-Reisen*, Germany
06/27/1997 - 07/19/2000	Employee for *DO & CO Partyservice and Catering Ges.m.b.H*, Austria
07/17 - 08/17/1996	Tourists' guide in France for *Ragazzi-Reisen*, Germany
04/01 - 04/09/1995 & 03/27 - 04/05/1996	Organization and realization of tennis camps in Italy
since 1994	Regular coaching and teaching of small groups at a several tennis clubs and promoting the youth work
07/11 - 08/31/1994	Working student for the *BASF Ludwigshafen*, Germany