About the author

Tom Broens has a master's degree in Telematics from the University of Twente, the Netherlands.
In 2004, Tom was awarded the KIVI/UT 2004 internship price for his internship at Twente Medical Systems International. Soon after that, he joined the University of Twente as a full-time researcher. From 2004 to 2008, he developed his Ph.D. work and participated in the European Amigo project and the Dutch AWARENESS project.
He has authored several international publications including journal, conference and workshop contributions.
Further, he has served as reviewer for international conferences and workshops. Since 2008, he works as scientific researcher at the Telematica Instituut, the Netherlands.

DYNAMIC CONTEXT BINDINGS

INFRASTRUCTURAL SUPPORT

FOR CONTEXT-AWARE APPLICATIONS

Tom Broens

Context-aware applications use context information, offered by context sources, to adapt to the situation at hand. The exchange of context information requires an association between the context consuming context-aware applications and suitable context producing context sources. We call these associations 'context bindings'.
This thesis provides insights in the generic characteristics of context-aware applications and their development process. We propose an abstraction, called the Context Binding Transparency, to mask the complexities of creating and maintaining context bindings for the application developer.
In this way, we facilitate the development process of context-aware applications. The responsibility for creating and maintaining context bindings is shifted from the application developer to a context binding infrastructure. This enables application developers to focus on the development of the primary application logic rather than the logic to create and maintain context bindings.
We propose a realization of a context binding infrastructure called the Context-Aware Component Infrastructure (CACI). This infrastructure realizes a context binding transparency and is composed of a context binding mechanism and a context discovery interoperability mechanism. CACI is prototyped using component middleware technology.
The feasibility and usefulness of the context binding infrastructure is evaluated using a case from the telemedicine domain.

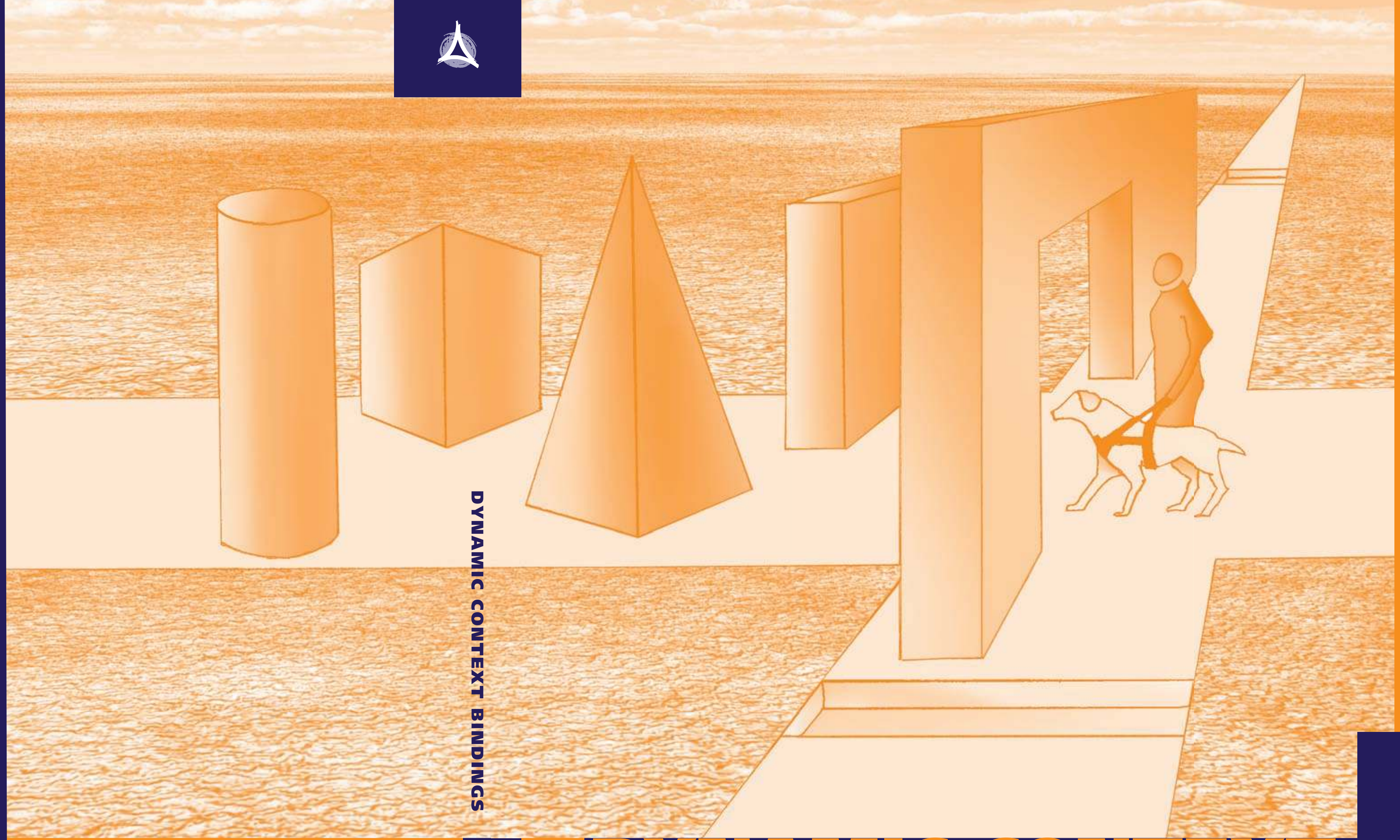
ISBN 978-90-75176-47-6

CTIT Centre for Telematics and Information Technology

Telematica Instituut

DYNAMIC CONTEXT BINDINGS

TOM BROENS



DYNAMIC CONTEXT BINDINGS

INFRASTRUCTURAL SUPPORT FOR CONTEXT-AWARE APPLICATIONS

TOM BROENS

Telematica Instituut
On top of technology.

This publication is a collaborative result of CTIT (Centre for Telematics and Information Technology) and the Telematica Instituut. It is published as part of the CTIT Ph.D.-Thesis Series and the Telematica Instituut Fundamental Research Series.
Part of the research presented in this thesis was done in the context of the AWARENESS (Context AWARE mobile NEtwork and ServiceS) project, a BSIK Freeband project, sponsored by the Dutch government. AWARENESS focuses on an infrastructure that enables rapid and easy development of context-aware and pro-active applications in a secure and privacy-conscious manner.
CTIT (www.ctit.utwente.nl) of the University of Twente is one of the largest academic ICT research institutes in Europe. It conducts research on the design of complex ICT systems and their application in a variety of domains. CTIT's unique multi-disciplinary approach makes it an attractive partner. The institute maintains an extensive international network of contacts and working relations with academia and industry. This network includes ICT and manufacturing companies, universities and research institutes, health care organisations, financial institutes, governmental organisations, and logistics service providers.
'Telematica Instituut' (www.telin.nl) is a unique partnership between the business community, research centres and government to perform telematics research for the public and private sectors.
The emphasis is on rapidly translating fundamental knowledge into marked-oriented applications. The institute's objective is to strengthen the competitiveness and innovative strength of Dutch business, as well as improving the quality of our society through the proper application of telematics. To achieve this, the institute brings together leading researchers from various institutions and disciplines. The Dutch government supports 'Telematica Instituut' under its 'leading technological institutes' scheme.

www.telin.nl

Dynamic Context Bindings:
Infrastructural Support for Context-Aware Applications

Telematica Instituut Fundamental Research Series

001 G. Henri ter Hofte, *Working Apart Together: Foundations for Component Groupware*
002 P. J.H. Hinssen, *What Difference Does It Make? The Use of Groupware in Small Groups*
003 D.D. Velthausz, *Cost Effective Network Based Multimedia Information Retrievel*
004 L. van de Wijngaert, *Matching Media: Information Need and New Media Choice*
005 R.H.J. Demkes, *COMET: A Comprehensive Methodology for Supporting Telematics Investment Decisions*
006 O. Tettero, *Intrinsic Information Security: Embedding Information Security in the Design Process of Telematics Systems*
007 M. Hettinga, *Understanding Evolutionary Use of Groupware*
008 A. van Halteren, *Towards an Adaptable QoS Aware Middleware for Distributed Objects*
009 M. Wegdam, *Dynamic Reconfiguration and Load Distribution in Component Middleware*
010 I. Mulder, *Understanding Designers, Designing for Understanding*
011 R. Slagter, *Dynamic Groupware Services – Modular Design of Tailorable Groupware*
012 N.K. Diakov, *Monitoring Distributed Object and Component Communication*
013 C.N. Chong, *Experiments in Rights Control: Expression and Enforcment*
014 C. Hesselman, *Distribution of Multimedia Streams to Mobile Internet Users*
015 G. Guizzardi, *Ontological Foundations for Structural Conceptual Models*
016 M. van Setten, *Supporting People in Finding Information: Hybrid Recommender Systems and Goal-Based Structuring*
017 R. Dijkman, *Consistency in Multi-viewpoint Architectural Design*
018 J.P.A. Almeida, *Model-Driven Design of Distributed Applications*
019 M.C.M. Biemans, *Cognition in Context: The effect of information and communication support on task performance of distributed professionals*
020 E.Fielt, *Designing for Acceptance: Exchange Design for Electronic Intermediaries*
021 P.Dockhorn Costa, *Architectural Support for Context-Aware Applciations: From Context Models to Services Platforms*

DYNAMIC CONTEXT BINDINGS: INFRASTRUCTURAL SUPPORT FOR CONTEXT-AWARE APPLICATIONS

Tom H.F. Broens

CTIT
Centre for Telematics and Information Technology

Telematica
Instituut

Enschede, The Netherlands, 2008

CTIT Ph.D.-Thesis Series, No. 08-125
Telematica Instituut Fundamental Research Series, No. 022 (TI/FRS/022)

Cover Design:	Studio Oude Vrielink, Losser and Jos Hendrix, Groningen
Cover Illustration:	"Guide-dog metaphor" by Adrie Broens
Cover Effects:	Gerhard de Groot
Book Design:	Lidwien van de Wijngaert and Henri ter Hofte
Printing:	Universal Press, Veenendaal, the Netherlands

CTIT Ph.D.-Thesis Series, No. 08-125
ISSN 1381-3617; No. 08-125
Centre for Telematics and Information Technology, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

Telematica Instituut Fundamental Research Series, No. 022
ISSN 1388-1795; No. 022
ISBN 978-90-75176-47-6
Telematica Instituut, P.O. Box 589, 7500AN Enschede, The Netherlands

DYNAMIC CONTEXT BINDINGS: INFRASTRUCTURAL SUPPORT FOR CONTEXT-AWARE APPLICATIONS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. W.H.M. Zijm,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 21 november 2008 om 15.00 uur

door
Tom Henri Ferdinand Broens
geboren op 29 juli 1981
te Enschede

Dit proefschrift is goedgekeurd door:
prof.dr.ir. L.J.M. Nieuwenhuis (promotor) en dr.ir. M.J. van Sinderen (assistent-promotor).

Abstract

The world is increasingly equipped with high-capacity, interconnected, mobile and embedded computing devices. Context-awareness provides an attractive approach to personalize applications such that they better suit the user's needs in this rich computing environment.

Context-aware applications use context information, offered by context sources, to adapt their behavior to the situation at hand. The exchange of context information requires an association between a context consuming context-aware application and suitable context producing context sources. We call these associations 'context bindings'.

Developing context-aware applications is complex due to some intrinsic characteristics of context sources. Firstly, context sources are distributed. Consequently, creating a context binding requires some form of discovery and selection of context sources. Secondly, context sources are arbitrary available during the life-span of the application. This makes a binding hard to maintain. Finally, context sources offer context information with a fluctuating quality. This makes a binding possibly unsuitable for the application. Currently, developers need to spend considerable effort to develop application code to create and maintain required context bindings, which can deal with these complexities.

This thesis provides insights in the generic characteristics of context-aware applications and their development process. We propose an abstraction, called the Context Binding Transparency. This transparency has as goal to mask the complexities of creating and maintaining context bindings for the application developer. In this way, we facilitate the development process of context-aware applications. The responsibility for creating and maintaining context bindings is relieved from the application developer and is shifted to a context binding infrastructure. This enables application developers to focus on the development of primary application logic rather than the logic needed to create and maintain context bindings.

The application developer interacts with the context binding infrastructure using context retrieval and publishing services, and a context requirement specification language. This language enables application developers to specify their requirement at a high level of abstraction rather than in programming code. In this thesis, we propose a realization of such a context requirement specification language, coined the Context Binding Description Language (CBDL). This language is developed to be generic for a broad range of context-aware applications.

Additionally, we propose a realization of a context binding infrastructure called the Context-Aware Component Infrastructure (CACI). This infrastructure realizes a context binding transparency and is composed of a context binding mechanism and a context discovery interoperability mechanism.

The context binding mechanism uses CBDL documents, specified by the application developers, to create and maintain context bindings on behalf of the application. The process of creating a binding consists of discovery of context sources at available context discovery mechanisms, selection of suitable context sources, establishment of a binding of the application to these context sources, and maintenance of this binding. Maintenance of a context binding includes re-binding to other suitable context sources in case of lost or (re-)appearing context sources and fluctuating quality of context. This thesis gives an example of a possible re-binding algorithm.

The context discovery interoperability mechanism enables context-aware applications to interoperate transparently with different context discovery mechanisms available in the application environment. The goal of the interoperability mechanism is to hide the heterogeneity and fluctuating availability of context discovery mechanisms for context-aware applications. The context discovery interoperability mechanism is a supporting mechanism for the context binding mechanism. It can also be used independently by context-aware applications that do not leverage from the context binding mechanism.

We have created a proof-of-concept prototype of CACI, using the OSGi component framework. The prototype includes implementations of the context binding mechanism and the context discovery interoperability mechanism.

Evaluation of the proposed context binding transparency and infrastructure consists of a user survey and a comparison on the development effort and software quality of a Telemedicine case implementation with and without CACI. The survey indicated a general interest of possible users in the features of the context binding transparency. The case implementations indicated a possible improvement in the development process of higher quality context-aware applications when using a context binding infrastructure.

This research stresses that the availability of context information and the quality of this information highly influences the development of context-aware applications. By using a middleware infrastructure to support the creation and maintenance of context bindings, the development of higher quality context-aware applications can be simplified.

Acknowledgements

'Makkelijk maakt lui'

Things and people are only meaningful in their context. For example, this thesis consists of around 524.000 characters. These characters are meaningless to a reader without their context. The reader should know the notion of a 'word' as being a sequence of characters divided by spaces to grasp its meaning. But often this is not enough; the meaning of a word becomes clear(er) when considering its surrounding sentences, paragraphs or maybe even chapters. This thesis consists of approximately 74.500 words. From the words in this thesis, 'context' is the most significantly occurring one (3699x). The occurrence of the word 'context' is in the majority of the cases in combination with other words like 'information', 'binding', '-aware application', 'source', 'discovery', '-awareness', etc. Hence, the context of the word 'context' is very important to understand this thesis.

Like the fact that words in a thesis only become meaningful in their context, the author's context is very important for the completion of a thesis. In my view, the quality level of this thesis is highly influenced by the people in my context. In the four years I spend at the University of Twente in the ASNA and IS group, I had a lot of good times, some bad times and I met a lot of nice and inspiring people. Without these people I could not have completed this thesis. I am very grateful that they occupy my context.

Let me start by thanking my promoter Bart Nieuwenhuis and supervisors Marten van Sinderen and Dick Quartel. After the ASNA reorganisation, there was a probability of me falling in a supervision void. However, they stood by my side and kept supporting me all the way. I am very grateful for this. Furthermore, their inspiring thoughts and quality remarks made the thesis as it is today. Thank you!

I want to thank the members of my promotion committee: professor Blair, professor Goebel, professor Konstantas, professor Hermens, professor van Hillegersberg and the people mentioned above, for their input to this thesis.

Parts of this work are done in close collaboration with bachelor and master students that I supervised. I want to thank Jasper Aarts, Jesper Jeeninga, Martijn Eenennaam, Peter Hoekerd and Tania Tariq for their contribution to this thesis. Additionally, I want to thank some people that helped to complete particular parts of the thesis: Aart van Halteren for his supervision in the beginning of my PhD, Klaas van den Berg for his help on software evaluations, Toni Piirainen for the integration of SimuContext with Vantagepoint, Ander de Keijzer for introducing me to Survay, and Rianne Huis in't Veld for helping me to get knowledgeable in the Telemedicine domain.

During my PhD I worked in two major projects. I want to thank the members of the AWARENESS and AMIGO project for the inspiring discussions and for offering me the opportunity to ventilate my research results and get valuable feedback. Especially, I want to thank my AMIGO collaborators at the Telematica Instituut: Henk Eertink, Remco Poortinga and Andrew Tokmakoff for the fun times and their quality feedback.

For me it is important that my working environment is besides intellectual challenging also socially engaged. This combination I found in the ASNA group and its surroundings. I want to thank the FASNA members (Former ASNA) and people I associate with FASNA: Annelies, Marlous, Maarten, Luis, Ricardo, Rodrigo, Luiz, Tiago, Lisandro, Eduardo, Laura, Teduh, Hailiang, Pravin, Kamran, Fadli, Remco Dijkman, Remco van der Meent, Giancarlo, Renata, Robert, Ing, Bert-Jan, Val, Bert, Richard, Kate. Some of you I am happy to call my friend! You made life most enjoyable on the fourth floor and later on the third and fifth floor. I had wonderful discussions and small-talk during the (regular) coffee breaks or during the 'appeltje' sessions. Also the parties at Macandra will stay in my memories. Additionally, I want to thank the members of the IS group for embracing me in their group, although I could not become a formal member. Especially, I want to thank Suse for her excellent support!

I also want to mention some special people. I want to thank my 'paranimfen' and best friends: Jeroen and Joris. Thank you for being my friend and helping me during the defense. I want to thank my soccer friends of the UT-kring for all the sportive highlights. Furthermore, I want to thank my TRMC model-flying friends for joining me in our great hobby. Especially, I want to thank Gerhard for his work on my cover. Although everything that has happened, I also want to thank Miriam and her family for their support and love. Things always happen unexpectedly, I want to thank Gertie for making me feel again. I hope to enjoy your company for a long time.

Finally, I want to thank my family which is the keystone of my life. Although you didn't always understood what I did, you had faith in the good ending and supported me throughout with all your hearth. I want to thank my sisters and their family for just being there: Marloes, Bas, Wouter, Leonie, and Sandra, Bas and Marit. Ofcourse, I want to express my gratitude, love and deepest respect for my parents: Adrie en Ria. Without their support and effort to enable the best possible opportunities for me, this result would not have been possible.

So … and now it is done. It is time to proceed. I always remember an old saying from Twente '*De geleardsten bint de wiesten nich*'. This saying means something like the most learned persons aren't always the wisest persons. In my view this touches a fundamental point. Not only studying but also experience and the people you encounter determine how 'wise' you are. After more than ten years of studying I want to explore and get wise(r). It is time for me to further enrich my context.

Tom Broens
Enschede, November 2008

Contents

Chapter 1

Introduction

This thesis proposes infrastructure-based mechanisms and abstractions to facilitate the development of *context-aware applications.* These are software applications that adapt their behaviour to the situation at hand. This chapter presents the motivation for this research and outlines the objectives and the adopted approach.

This chapter is organized as follows: Section 1.1 provides background information that is relevant for this research. Section 1.2 gives a problem description and analysis. Section 1.3 presents the objectives and research questions. Section 1.4 gives the scope of this research. Section 1.5 introduces the telemedicine case domain that is used to evaluate this research. Section 1.6 describes the adopted approach. Finally, Section 1.7 presents the structure of the remaining thesis.

1.1 Background

Humanity strives for constant innovation to improve the quality of life. Increasingly, computer systems are used for this purpose. For example, we observe that human users have increasingly access to, possibly multiple, computer systems in their environment[1]. With the user's environment, we denote the physical space in which the user lives such as a home, office and public spaces. Many users use a combination of computer systems, such as

[1] The number of Personal Computers in the world increased from 130 million in 1991 to 775 million in 2004 (source: ITU, http://www.itu.int/ITU-D/ict/statistics/). Access of all European households to a Personal Computer increased from 50% in 2002 to 64% in 2006 (source: EuroStat, http://ec.europa.eu/eurostat/). In the Netherlands this is even 84% of all the households in 2006 (source: CBS, http://www.cbs.nl). Households also increasingly use 'internet enabled' mobile computer systems like mobile phones, laptops, PDA's and palmtops. In Europe, access to such systems has increased from 13% in 2002 to 28% in 2006 (source: EuroStat). In the Netherlands this is 35% in 2006 (source: CBS).

PCs, laptops, PDA's, mobile phones, mp3 players, media-centres and Personal Video Recorders (PVR) to perform their job or enjoy themselves. Even, traditional non-computerized objects are currently equipped with computer capabilities. Consider, for example, refrigerators that are equipped with an integrated TV and internet connection. Hence, many computer systems already reside in the user's environment. However, the majority does not cooperate to perform more useful functions for the user (Davies and Gellersen 2002).

Technological Developments

We distinguish the following technological developments that stimulate the integration of computer systems in the user's environment:

– *Increasing capacity with lower costs:* Computer systems offer increasing processing, memory, storage, communication bandwidth and battery capacity. For example when considering processing capacity, a commonly used estimation is 'Moore's law'. This law states that roughly every 18 months the transistor capacity on integrated circuits doubles. Similar trends are visible in memory and storage capacity. Currently, the only aspect that stays behind is battery capacity. Additionally, the costs of these high-capacity devices are decreasing.
– *Miniaturization:* Computer systems become increasingly smaller with similar or increasing capabilities. Miniaturization has two effects on the use of computer systems: (i) mobility and (ii) integration. The computer systems' size/capability ratio enables users to wear or carry useful computer systems. Additionally, computer systems become small enough to be 'hidden' in the user's daily environment and still be able to perform useful tasks (Bohn, Coroama et al. 2005).
– *Improved connectivity:* Computer systems become increasingly connected to each other and to the 'world' using the Internet. Mobile communication mechanisms like, amongst others, UMTS, Bluetooth and WLAN, enable computer systems to exchange information anywhere and at anytime. Consequently, the number of (mobile) internet users increased significantly[2].
– *Changing computing paradigm:* Users operate an increasing number of different 'personal' computer systems. For example, mobile phones and MP3 players. These computer systems are developed and configured with the purpose of being used by individual users rather then being generic for multiple users. Additionally, these computing systems are less recognizable as traditional computing systems. For example, internet-enabled wristwatches and refrigerators.

[2] The number of internet users increased from 4.4 million in 1991 to 863 million in 2004 (source: ITU, http://www.itu.int/ITU-D/ict/statistics/)

These developments lead to a growing awareness that human-computer interaction in future computer systems should be user-centric rather then, traditionally, technology-centric (Oulasvirta 2005). Rather that users are being forced to adapt to the computing system, the computing system should adapt to the users (Aarts and Marzano 2003). All these computing systems offer functionality and information to users. A real problem of such rich computing environments is that the user may be overloaded with information, leading to annoyance of the user and even a decrease of the user's efficiency to perform a certain task.

Calm Technology

In the 1990's, Weiser raises the need for technology to overcome the saturation of the user's attention, which he calls **calm technology** (Weiser 1991; Weiser and Brown 1998). He states that technology should both 'encalm' and inform. He therefore distinguishes the centre and periphery of the user's attention. The centre of attention is that what users explicitly focusing on. The periphery of attention is that what users are aware of without explicitly focussing on. For example, when driving a car, the road ahead is in the centre of attention, while the speedometer is in the periphery of attention.

Calm technology should overcome the saturation of the user's attention, making the daily life more enjoyable and effective (Ebling, Guerney et al. 2001). Weiser states this can be enabled in two ways, by: (i) technology that moves easily, and at the right moment, between the centre and periphery of attention and (ii) technology that enriches our peripheral reach. For example, the speedometer could blink when a user is speeding. In this way, the speedometer moves between the centre and periphery of attention of the user. Additionally, the user could be informed how much the fine will be when he is caught speeding. In this way, the user's periphery of attention is enriched.

Context-Aware Applications

The concept of calm technology has lead to developments enabling a future world that is filled with interconnected and high capacity (mobile) computing devices, which are integrated in the user's environment. These devices host applications that support users in their daily life. These applications should support the user unobtrusively by adapting to the situation at hand. For example, the user is listening to some music. While moving, the sound follows the user through his house. When the phone rings the music is turned to a low volume automatically. The music is moving from the centre to the periphery of attention, such that the user can speak to the person on the phone. In this way, the 'radio listening application' adapts to the situation of the user such as his current physical location and the fact that he receives an incoming phone call.

The information that characterizes the situation of an entity, for example a human user, is called **context information**. Software applications that use context information to adapt their behaviour are called **context-aware applications**. We distinguish the following advantages of using context information to adapt the behaviour of applications:

- *Tailored human-computing interaction:* Context information may be used to filter or personalize information delivered to the user and may limit or personalize the human-computer interaction between the user and application. In this way, possible decrease of effectiveness due to huge amount of available information, sometimes called 'information overload' (Maes 1994), can be countered. For example, coping with the available information coming available through the Internet[3], is a challenge that current users are facing. This information can be filtered to a more comprehensible set of information by taking into account the current user's situation.
- *Internal optimization:* The availability of context information may provide the application knowledge about its execution environment such as available bandwidth and CPU load. Hence, the internal behaviour of applications can be optimized by using this knowledge. For example, by additionally taking into account the available bandwidth information a context-aware application could optimize its transmission strategy. For example, the application could enable/disable compression of outgoing data.
- *Novel applications:* By using context information, novel types of applications can be created that may provide novel commercial opportunities. For example, in the Netherlands, multiple location-aware applications are currently being deployed such as museum tour guides[4], person trackers[5] or material trackers[6].

We identify three main current directions that research context-awareness to enable calm technology. These research directions are **ubiquitous computing**, **pervasive computing** and **ambient intelligence.** Ubiquitous computing is a concept, often used in the United States, to indicate research that originates and builds directly on the ideas of calm technology proposed by Weiser. Pervasive computing is a term more

[3] Available web sites grew from approximately 10 million in the beginning of 2000 to 100 million in the beginning of 2007 (source: Netcraft, http://news.netcraft.com/)

[4] N8 Museum gids, http://www.n8.nl/2006/mobiel http://www.zdnet.nl/news.cfm ?id=62145 (in Dutch)

[5] Waarbenik, http://waarbenik.nl/ , http://www.telecomwereld.nl/n0000278.htm (in Dutch)

[6] NS tracking and tracing, http://www.computable.nl/artikel.jsp?id=1377888 (in Dutch)

common in industry, which is proposed by IBM in the end of the 1990's (IBM 1999). Ambient intelligence originates from the European IST advice group (ISTAG) at the end of the 19th and the beginning of 20th century. The sketched directions have similar goals but slightly different focus. Ubiquitous and pervasive computing are directions, which are more device-oriented focussing on integrating and combining devices in the user's environment. Ambient intelligence combines this aspect with human-computer interaction aspects such as multimodal interactions (Shadbolt 2003; Svahn 2003).

This thesis should be read in the light of the developments and trends sketched in this section. The contribution that is presented in this thesis focuses on supporting the development of context-aware applications.

1.2 Problem Analysis

By nature, humans are context-aware. Humans are capable of sensing their environment and reacting correspondingly. For example, a human can adapt its conversation to the body language of the receiving person and the goal he wants to reach. However, for context-unaware computer applications to adapt to changing circumstances, to provide personalized and appropriate functionality to the user, is challenging. For example, current personal music applications are not designed to deal with interruptions such as an incoming phone call. The user has to manually operate the music application to adapt to the changing circumstances of an incoming phone call.

Limited availability of high capacity sensory devices stimulated applications to operate in static execution environments (Schilit 1995) and to be build context-independent (Lieberman and Selker 2000). Due to the sketched improving device capabilities, a broad spectrum of sensors is currently available. These sensors can sense all kind of context information, which is becoming available to applications. Together with the before mentioned trend of high capacity mobile devices integrated in the users environment and the need for user-centric applications, this lead to increasing interest in context-aware applications.

Context information and Context-Aware Applications

Context information is commonly defined as: "any information that can be used to characterize the situation of an entity, in which an entity can be a person, place, physical or computational object that is considered relevant to the interaction between an entity and an application, including the application and the user themselves" (Dey 2000). Examples of context information are location, mood, number of read emails, weather conditions etc. **Context-awareness** is commonly defined as: "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." (Dey 2000).

Context-Aware applications are particularly interesting for mobile and wearable applications (Satyanarayanan 1996). These applications operate in constant changing environments due to the movement of the user. For example, one of the first mobile context-aware applications is the Cyberguide application, which is a mobile tourist guide application that offers tailored information on points of interest, based on location and orientation of the tourist (Long, Kooper et al. 1996).

Context Consumers, Producers, and Context Bindings

Context-aware systems consist of software entities that can perform a context producer and/or context consumer role. **Context producers** are entities that acquire context from the physical environment and offer it to context consumers. Examples of context producers are software-wrapped sensors such as GPS, temperature sensor, ECG sensor or pure software producers like a software-wrapped Outlook calendar. In this thesis, software entities that perform solely a context producer role are called **context sources**. **Context consumers** are entities that use provided context information from context producers to adapt their behaviour. In this thesis, software entities that perform at least a context consumer role are called context-aware applications.

Summarizing, we model a comprehensive context-aware system as a composition of associated context producers and consumers, which exchange context information. The association between a context consumer and a context producer that is required for exchanging context information, is called a **context binding**. Transfer of context information consists of three phases:

1. Creation of context bindings between context consumers and producers.
2. Requesting and exchange of context information using an established context binding.
3. Releasing of established context binding.

Figure 1-1 presents an example of context-aware system. It depicts three context sources that produce context information and offer it to two context consumers. For context information to be transferred from a producer to a consumer, a context binding is required. The context information always flows from a context producer to a context consumer. A context-aware application adapts its behaviour based on received context information and may produce context information, which it can provide to other context consumers.

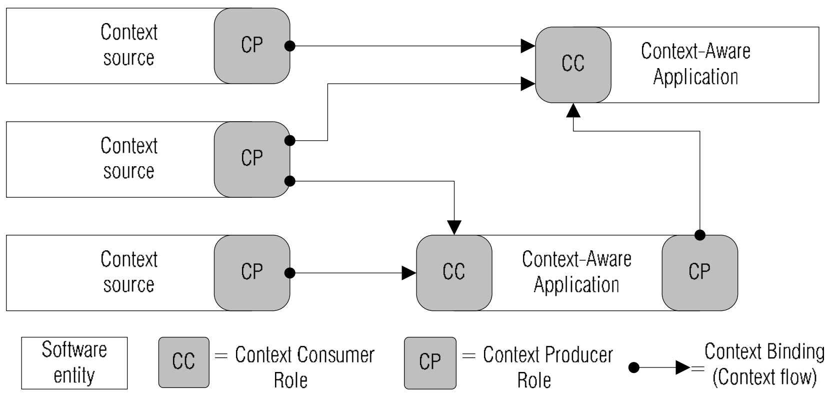

Context source
CP
Context source
CP
Context source
CP
CC
Context-Aware Application
CC
Context-Aware Application
CP
Software entity
CC = Context Consumer Role
CP = Context Producer Role
= Context Binding (Context flow)

Figure 1-1 Example of a context-aware system consisting of a composition of context producers and consumers.

Towards Third Generation Context-Aware Applications

We distinguish three generations of context-aware applications (see *Table 1-1*). Developers of **first generation context-aware applications** do not use middleware infrastructures in their development process (for examples of first generation context-aware application see (Chen and Kotz 2000; Korkea-aho 2000)). In these applications, context bindings are predetermined and hard-coded, in an ad-hoc and tightly coupled fashion, which is unique for their specific application (Dey 2000; Pascoe 2001). These developers choose specific context producers such as GPS location sensors, RFID sensors, and program the low-level interaction between the specific context producer and their application. Thereby, they create a tight coupling between their application and the used context producers. Reuse of the created application is limited and future evolutions, for example due to the upcoming of new technology, becomes difficult (Dey 2000; Ebling, Guerney et al. 2001). Additionally, as ubiquitous computing environments are highly dynamic in terms of availability and quality of context information, a loose coupling between context consumers and context producing entities offers clear advantages. A context-aware application should be able to find (i.e. discover), bind and use context producers, as they are available during the lifetime of the application (Davies and Gellersen 2002).

Currently, there is a trend towards using middleware infrastructures for facilitating the development of **second generation context-aware applications** (Henricksen, Indulska et al. 2005). These infrastructures offer solutions to recurring functions in the context-aware domain, like context discovery, reasoning and security. By applying run-time discovery, context producers and context consumers are increasingly decoupled and can be bound at run-time.

Table 1-1 Comparison of three generations of context-aware applications.

	1st Generation	2nd Generation	3rd Generation
Characterization	Application that adapts its behaviour to context information from predetermined and fixed context sources	Application that adapts its behaviour to context information from fixed at run-time discovered context sources	Application that adapt its behaviour to dynamically available context information with fluctuating quality from multiple context sources
Elements	Application + Context sources	Application + Context middleware + Context sources	Application + Context Binding Transparency / Context Binding infrastructure + Context sources
Binding time	Design-time	Run-time	Run-time
Binding management	Application managed	Application managed	Infrastructure managed
Binding type	Static and predetermined	Static, not-predetermined	Dynamic, based on availability and quality
Binding coding	Programmed binding	Programmed binding	Configurable binding in binding specification language

However, really establishing and maintaining context bindings is not supported by current context middleware infrastructures. Creation and maintaining context bindings is not trivial and still needs extensive programming effort (Banavar and Bernstein 2002). This originates from inherent characteristics of context sources:

- Context sources are distributed and a-priori unknown, so they need to be discovered before a context binding can be established. Furthermore, multiple context sources may be available (i.e. discoverable) in the environment, so selection is needed to bind to the most suitable context source.
- Context sources have dynamic availability, hence the persistence of an established context binding cannot be guaranteed.
- Context sources provide context information with different and changing qualities. This influences both the selection of context sources before the establishment of a context binding, and the decision to keep an established context binding or to replace it by another (better) one.

Hence, developers that use current context middleware infrastructures still need to create programming code to discover, select and bind to relevant context producers for every context consuming entity in their application. Furthermore, due to for instance the mobility of the user or possibly the context producer, the availability of the context producer for the context consumer is not guaranteed and reliable (Bellavista, Corradi et al. 2003).

Consequently, additional programming effort is needed to develop a flexible and robust context-aware system that can handle this dynamicity. For example when considering *Figure 1-2*, this figure depicts a user moving through different domains that offer multiple and different types of context information (e.g. presence, location, time). In domain A, certain context information may be available, while when the user moves to domain B, this context information becomes unavailable.

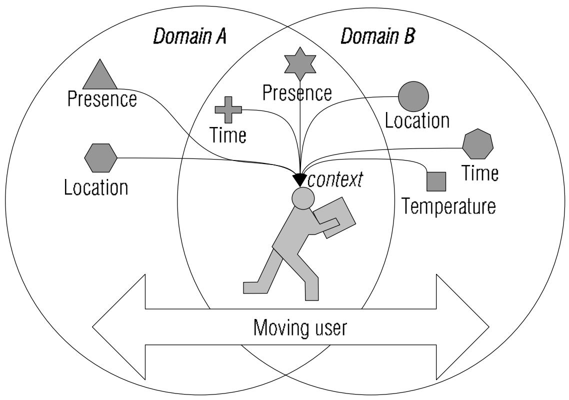


Figure 1-2 Context-Aware applications encounter different context information in different domains.

The hypothesis of this thesis is that the development process of **third generation context-aware applications** can be improved by offering a context binding infrastructure that realizes an abstraction, called the context binding transparency. In the remainder of this thesis, we develop the context binding transparency and infrastructure.

Context Binding Transparency and Context Binding Infrastructure

Hence, to facilitate the development process of third generation context-aware applications, we propose a contribution that consists of (i) an abstraction called the **context binding transparency** and (ii) a **context binding infrastructure** that realizes this abstraction. Correspondingly, our research consists of two perspectives:

– *Developer perspective*: this perspective refers to aspects concerning the developers of context-aware applications that should benefit from using the context binding infrastructure.
– *System perspective*: this perspective refers to aspects concerning the internal working of the context binding infrastructure that realizes the context binding transparency.

In general, we propose to shift the recurring problem of creating and maintaining a context binding from the application to the context binding infrastructure. The context binding transparency is realized by this infrastructure in terms of a **context retrieval and publishing service**. Application developers can use these services for easy exchange of context information. By using these services, the developer of a context-aware application becomes unaware of which context source is involved in creating a context binding, how this binding is created, and how this binding is maintained to overcome the dynamic availability and fluctuating quality of context sources. *Figure 1-3* represents the elements we propose to realize a context binding transparency and infrastructure:

1. A context requirement specification language that enables developers to specify their context requirement at an abstract level rather than directly programming these requirements (i.e. developer perspective).
2. A context binding mechanism that, based on a context requirement specification, creates and maintains context bindings, thereby hiding the distribution, heterogeneity and especially the dynamic availability and fluctuating quality of context producers for the application developer (i.e. system perspective).
3. A context discovery interoperability mechanism, which hides the heterogeneity and dynamic availability of context discovery mechanisms (i.e. system perspective).

In the remainder of this thesis, we elaborate on the design and implementation of these elements.

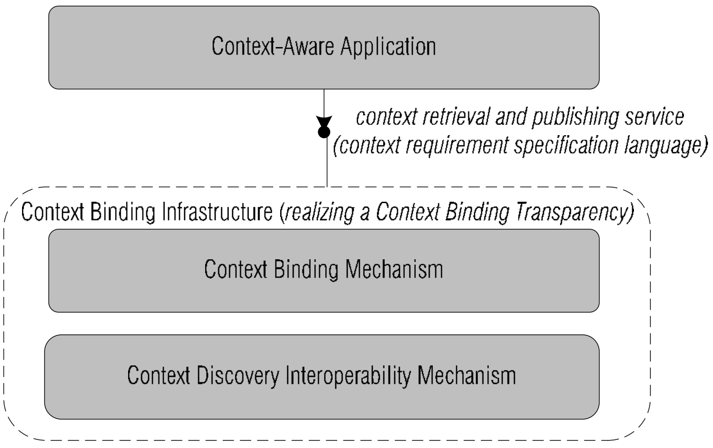

Context-Aware Application
context retrieval and publishing service
(context requirement specification language)
Context Binding Infrastructure (realizing a Context Binding Transparency)
Context Binding Mechanism
Context Discovery Interoperability Mechanism

Figure 1-3 High-level overview of the proposed contributions.

1.3 Objective and Research Questions

The generic goal of this thesis is to research infrastructure-based mechanisms and abstractions that support developers in creating context-aware applications. More specifically, the main objective of this research is:

Improve the development process of context-aware applications by applying a context binding infrastructure, realizing a context binding transparency that:

- *enables application developers to specify context requirements at an abstract level rather than directly programming these requirements;*
- *hides, whenever possible, the dynamic availability and quality of context producers for the application developers;*
- *interoperates with heterogeneous and dynamically available context discovery mechanisms;*

To reach this objective, we address the following research questions:

From the developer perspective, what does a context binding transparency look like for the application developer?

1. How do context-aware applications differ from non-context-aware applications and how does this influence the development process of these applications? How does the proposed context binding transparency influence the design of context producers and consumers?
2. What context requirements can application developers have? What elements are needed in a context requirement specification language such that applications developers are able to specify context requirements suitable for their context-aware applications?
3. What operational interfaces should a context binding mechanism offer, such that application developers can deploy and test their context-aware applications?
4. How configurable should a context binding mechanism be to enable application developers to develop flexible context-aware applications?

From the system perspective, what does the context binding infrastructure look like that realizes the context binding transparency?

5. How can a context binding mechanism create a suitable context binding based on a context requirement specification?
6. How can a context binding mechanism maintain a created context binding in an environment where context producers can appear, disappear, and have fluctuating quality?
7. How can a context discovery interoperability mechanism deal with multiple heterogeneous and dynamically available context discovery mechanisms offering context producers?

Evaluation of the context binding transparency and context binding infrastructure:

8. How can the telemedicine domain benefit from context-aware applications? How can the context binding infrastructure be used for developing context-aware telemedicine applications?

1.4 Scope

From the developer perspective, the context binding transparency and infrastructure facilitate a subset of the phases in a comprehensive development process of (context-aware) applications. If we consider the common waterfall or linear development process model (Pressman 2000), the starting point for using our context binding infrastructure is the situation where there exist requirements for context-aware applications, including requirements for context information. The scope of our context binding infrastructure ends by using it for testing a developed context-aware application. The defined scope, based on the waterfall model, is visually represented in *Figure 1-4*. Hence, we consider requirements engineering, deployment and maintenance activities out of the scope of this research.

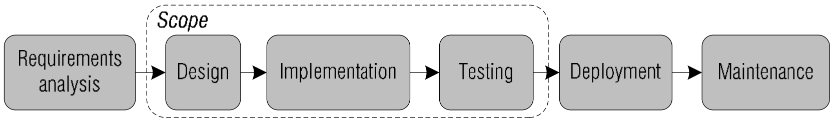


Figure 1-4 Positioning of the proposed contribution in the development process of (context-aware) applications.

From the system perspective, we assume the availability of IP-based communication mechanisms to invoke (remote) applications and services. On top of this, we assume the availability of (multiple) context discovery mechanisms that offers context discovery services. The development of context discovery mechanisms is out of the scope of this research.

Additionally, we assume that the quality information of the context information that is delivered by a particular context source is made available, either by the context source itself or by the corresponding context discovery mechanism. Determination of the actual quality values of context information is out of the scope of this research.

Furthermore, there are some aspects of context-aware systems related to context exchange, which we discuss briefly in the remainder of this thesis, but which we do not detail. We consider them out of the scope of this research:

– *Semantic interoperability:* The proposed context binding infrastructure tries to match descriptions of offered context information by context sources with requirements posed by the application developer. Both syntactic (i.e. syntax of the context request versus the syntax of the context offering) and semantic interoperability (i.e. the meaning of the request versus meaning of the context offering) influence the quality of this matching process.

- *Security issues:* When exchanging context information, security aspects such as enforcing the privacy of context consumers and producers, and the establishment of a trust relation between these parties, is key to successful operational context-aware system.
- *Business aspects:* Introducing context information provides opportunities for novel and enhanced applications and services. However, this may influence existing business processes or require novel and changed business models.

1.5 Case Domain: Telemedicine

The research described in this thesis is part of the Freeband AWARENESS project (Wegdam 2005). This project researches infrastructure-based mechanisms to support mobile context-aware applications (Sinderen, Halteren et al. 2006). The functions supported by these mechanisms consist of privacy enforcement, service discovery, context discovery and exchange, context modelling and context reasoning. The AWARENESS infrastructure is validated by developing *mobile context-aware healthcare* applications that use the AWARENESS infrastructure.

The research outlined in this thesis also uses healthcare as its case domain. In more detail, we focus on **Telemedicine** applications. The goal of telemedicine applications is to provide healthcare over distance using ICT (Tachakra, Wang et al. 2003). For example, applications that monitor and transfer vital signs of patients, who are living in their own home, to caregivers in a remote care institution.

We consider the Telemedicine domain as a valid application domain, because:

- *Social-economical trends:* Several social-economical trends such as aging and the increasing number of patients with chronic diseases require the future healthcare process to change to guarantee high quality healthcare. Therefore, researching ways to simplify the development of context-aware applications, which are envisioned to improve future healthcare processes, are relevant.
- *Specific requirements:* Due to the 'life-and-death' nature of healthcare applications, they have specific requirements that are more stringent than requirements for the majority of non-healthcare applications. Hence, especially for this type of applications, infrastructure

mechanisms to provide more reliable context information in terms of availability and quality, are potentially of great benefit.

1.6 Approach

Figure 1-5 presents the approach adopted in this research. Grey rounded rectangles depict phases in the research. White rectangles depict artefacts resulting from activities in a certain phase. These artefacts are used in other phases. The directed arrows present an input/result relation between the phases and artefacts.

The approach applied in this research is divided into four phases. The first phase consists of a *literature study* on the state-of-the art on:

– *Context, context-awareness and (context) middleware.* This results in (i) state-of-the-art (SOTA) and problem analysis on current context-awareness middleware approaches and (ii) a framework of basic concepts.
– *Telemedicine domain.* This results in the background information and motivation for the case study used for the evaluation.

The second phase is the *design* of the context binding infrastructure that realizes the context binding transparency. This includes the design of:

– *Context binding description language* that enables developers to specify their context requirements at a high level of abstraction.
– *Context binding mechanism* that hides the complexities of creating context-aware application that retrieve context information from dynamically available context sources with fluctuating quality.
– *Context discovery interoperability mechanism* that enables discovery of context sources from different domains that become available during the lifetime of the context-aware application.

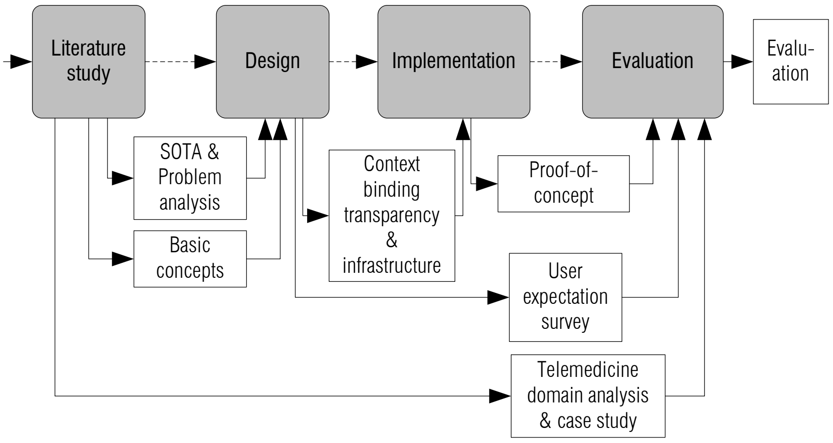


Figure 1-5 Research approach.

The third phase is the *implementation* of a proof-of-concept prototype based on the designed context binding transparency and context binding infrastructure. The final phase is an *evaluation* of the possible improvements in the development process of context-aware applications when using our context binding infrastructure. The evaluation is based on a:

- *User expectation survey* that determines the general interest of possible users (i.e. developers of context-aware applications) in the features of a context binding transparency and infrastructure.
- *Implementation of a Telemedicine case study with our context binding infrastructure* that shows the feasibility of the developed context binding transparency and infrastructure.
- *Comparison of a telemedicine case implementation with and without our binding infrastructure* that qualitatively compares and estimates the possible improvements in the development process of context-aware applications by using our context binding infrastructure.

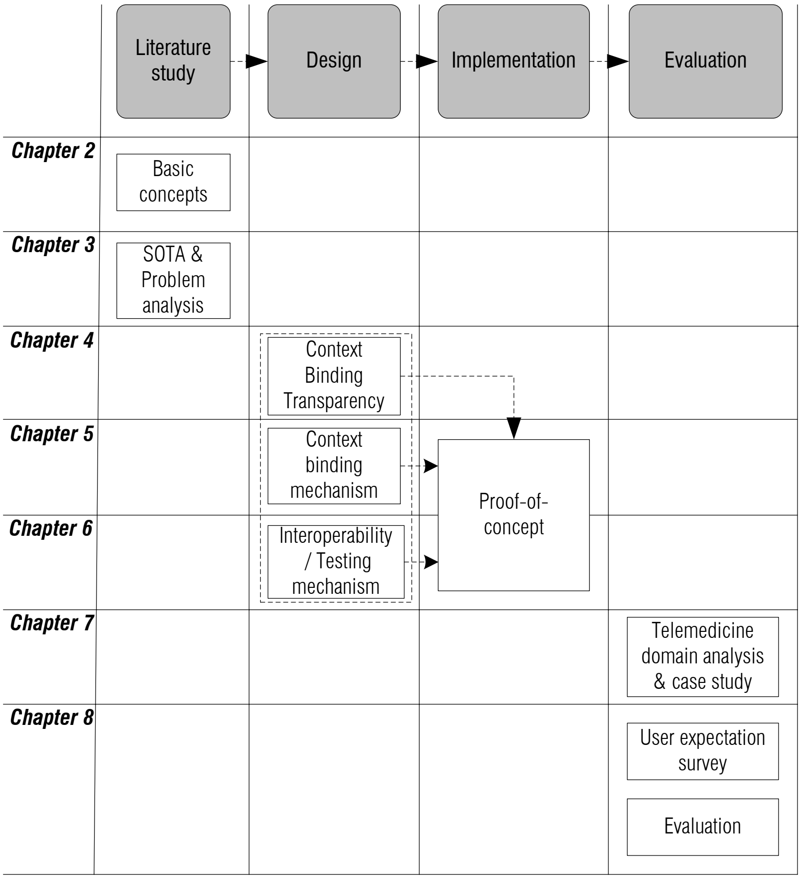
1.7 Structure

The structure of this thesis reflects the previously discussed approach. *Figure 1-6* correlates the structure of this thesis with the adopted approach. The remainder of this thesis is structured as follows:

- Chapter 2 discusses our framework of basic concepts, consisting of definitions, concepts and models used throughout this thesis.
- Chapter 3 presents the state-of-the-art in context middleware and further motivates our proposed context binding transparency and context binding infrastructure with a problem analysis.
- Chapter 4 reflects on the design process of context-aware applications from the perspective of the application developer (i.e. developer perspective) and presents the design of the context binding transparency.
- Chapter 5 presents the overall design of the context binding infrastructure. Additionally, it discusses the design and proof-of-concept implementation of the first part of the context binding infrastructure; the context binding mechanism.
- Chapter 6 presents the design and prototype implementation of the second part of the context binding infrastructure; the context discovery interoperability mechanism. Furthermore, it discusses a context simulation framework to facilitate testing of developed context-aware applications.
- Chapter 7 introduces the Telemedicine case domain and indicates the usefulness of applying context-awareness in this domain. Additionally, it

identifies key determinants influencing the success of telemedicine applications.
- Chapter 8 presents the evaluation of the proposed context binding transparency and infrastructure.
- Chapter 9 presents conclusions and future work.

Figure 1-6 Correlation of the adopted approach and structure of the thesis.



Chapter 2

Basic Concepts and Models

This chapter introduces basic definitions, concepts, and models needed to describe and reason about the proposed context binding transparency and infrastructure. Parts of this chapter are published in (Broens, Quartel et al. 2007).

The structure of this chapter is as follows: Section 2.1 presents the concept of systems and services. Section 2.2 describes applications and middleware. Section 2.3 elaborates on current definitions and characteristics of context and context information. Furthermore, it introduces our definitions of context and context information. Section 2.4 discusses current definitions and characteristics of context-aware applications. Furthermore, it introduces our definition of context-aware application. Section 2.5 describes design aspects of these applications. Section 2.6 presents a basic model of context-aware applications. Finally, Section 2.7 gives a generic discussion on the process of creating and maintaining context bindings.

2.1 Systems and Services

There exist several definitions of a **system**. For example, Merriam-Webster's dictionary defines a system as "a regularly interacting or interdependent group of items forming a unified whole". The oxford dictionary defines a system as "a complex whole; a set of things working together as a mechanism or interconnecting network" and Chambers technology dictionary defines a system as "anything formed of parts placed together or adjusted into a regular or connected whole". These definitions indicate the main characteristics of a system: (i) a system consists of collaborating parts and (ii) these collaborating parts form a unifying whole. There are many types of systems: mechanical systems, ecological systems, political systems, ICT systems etc. An example of a mechanical system is a car. A car consists of multiple parts such as an engine, gearbox, steering wheel, tires etc. These parts work together to form a car system.

In this thesis, we restrict ourselves to ICT systems. ICT systems are commonly used in everyday life. For example, persons can schedule appointments in a digital calendar, send emails and instant messages to other persons, and retrieve money from an ATM.

System perspectives, abstractions and services

In this thesis, we adopt the concepts proposed by (Vissers, Ferreira Pires et al. 2002) for designing systems. Additionally, we adopt their method of structured design of systems for the use of modelling context-aware applications. Hence, we distinguish the following two perspectives on systems:

- *External perspective:* considers the system as a "unified whole" or black-box, and views it from the perspective of the system's user that wants to use it for some purpose. This user can be a human or another computer system. This perspective shows "what" behaviour the system is capable of offering.
- *Internal perspective:* considers the system as an "independent group of items" or white-box. It reveals the internals of the system, showing "how" the system is capable of offering certain behaviour.

The concept of system can be used recursively. The 'items' of which the internal perspective of a system is composed can be again considered as systems which have an internal and external perspective. For example, the engine in a car system consists of valves, pistons, flywheel etc. These parts work together to form a propulsion system for the car system.

Developing systems using these perspectives is advantageous. It offers developers a way to develop systems in a step-wise manner using different levels of abstraction. **Abstraction** is the process of addressing only development aspects relevant at that particular point in time while ignored other aspects, which are not (yet) relevant. In this way 'complexities', that the developer is not (yet) interested in and that may distract him from his current primary goal, are hidden. For example, a developer that is creating a car system may first focus on designing the engine before designing the gearbox. Hence, the developer considers the engine from an internal perspective while considering the gearbox from an external perspective. At a later point in time, the perspectives may be switched.

We define that from an external perspective a system provides one or more services to users. The system that provides a service is called a **service provider.** The user of a service is called a **service user**. A **service** is defined as the external behaviour of a system that has some desired effect for the service user. A service is defined in terms of interactions and the relation between interactions. An **interaction** is the activity of two or more cooperating systems in which a common result is established. For example, a service user may invoke an information service from a system using simple synchronous interactions, in which a request interaction is directly followed by a response interaction.

Example: Epileptic Alarm System

To illustrate the concepts discussed in this section, we explain them with a simplified example of an epileptic alarm system. *Figure 2-1* presents the epileptic alarm system from an external and internal perspective.

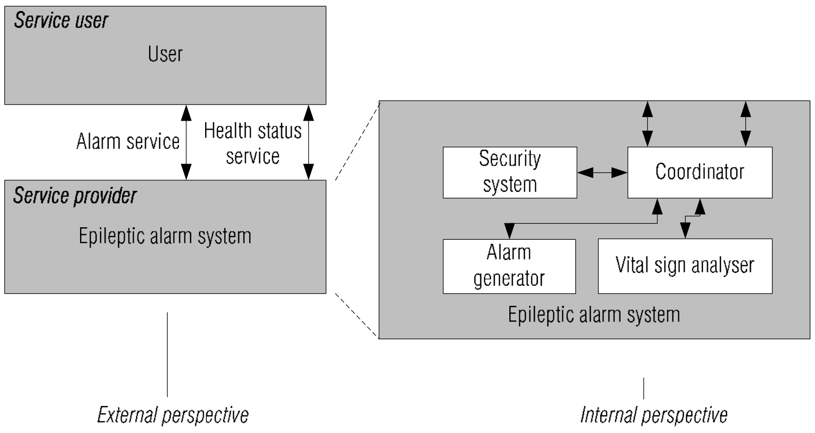


Figure 2-1 Example of an epileptic alarm system.

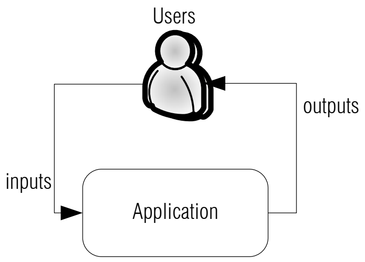
From an external perspective, the epileptic alarm system offers two services to service users: (i) 'alarm service' that informs on the alarm status of epileptic patients, and (ii) 'health status service' that informs on the health status of epileptic patients. The alarm service consists of a set of interactions that enable the service user to set a subscription to be notified of upcoming epileptic seizures. The health status service consists of two sets of interactions: (i) to request the status of individual patients that results in a direct response with the health status of that patient, and (ii) to subscribe to status changes of a patient which results in a notification of the health status of a patient when it changes.

We can take an internal perspective of the epileptic alarm system and further decompose it into several interacting sub-systems. For example, the epileptic alarm system is composed from a coordinator system that deals with the interaction with the user and delegates the responsibilities to other systems. Another sub-system is the security system that offers services to authenticate service users when they are trying to retrieve the health status of the patient. Additionally, there is a vital sign analyser and alarm generator system. The vital sign analyser offers services to collect vital signs from the patient and stores them in a database. The alarm generator provides services to analyse the collected data and it tries to infer if an epileptic seizure is likely to occur. If needed, the subsystems can be decomposed further.

2.2 Applications and Middleware

We define an **application** as a software system that offers a service to its users. This is schematically shown in *Figure 2-2*. From an external perspective, a user has interactions with an application. These interactions consist of a composition of inputs and outputs. From an internal perspective, an application has application behaviour that realizes the service.

Figure 2-2 Software application.



We distinguish two types of applications: (i) local applications and (ii) distributed applications. Local applications reside on a single execution environment. **Distributed applications** are applications that consist of multiple interacting application parts, which are located on different execution environments, connected by a communication platform consisting of communication software and hardware. In this thesis, we focus on distributed applications.

Developing distributed applications that consist of communicating application parts via heterogeneous communication platforms is complex. Middleware is introduced to limit these problems and to reduce development costs (in terms of time, money etc.) of distributed applications (Bernstein 1996; Alonso, Casati et al. 2004). **Middleware** is a software layer that provides supporting services for developing distributed applications. *Figure 2*-3 shows a non-middleware and middleware based application. Middleware acts as an intermediary between the application and the resources offered by operating systems such as the communication platform. Middleware has two characteristics: (i) it hides the complexities of the communication platform from the application in terms of so called **distribution transparencies** (Blair and Stefani 1998) and (ii) it shifts functions to deal with common complexities to the middleware layer.

Chapter 3 discusses (context) middleware in more detail and *Chapter 4* discusses transparencies in more detail.

Advances in device and wireless communication technology have lead to the development of mobile computing devices that can communicate everywhere and at anytime. Consequently, applications running on these devices also become mobile. Mobile applications reside and move with a human user. For example, current mobile devices, such as Personal Digital Assistants (PDA), have browser applications that enable the user to browse web pages while on the move. We define a **mobile application** as a distributed application of which one or more bearers of application parts can physically move.

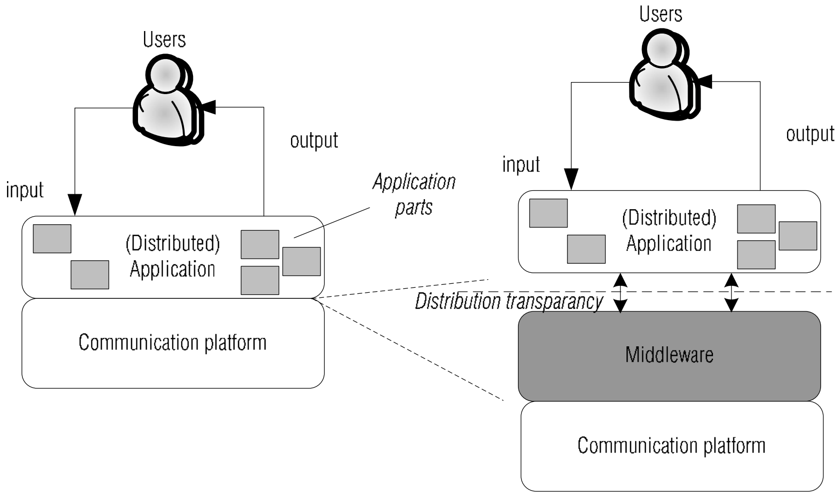


Figure 2-3 Distributed applications and middleware.

2.3 Context and Context Information

Many definitions of context have been proposed. However, creating a complete definition of context that fully captures its principles is complex (Bazire and Brezillon 2005). In this section, we discuss a small subset of context definitions, followed by our interpretation of the concept context and context information.

Definitions of Context

We observe that dictionaries define context from two perspectives: (i) the language perspective and (ii) the environmental perspective. For example, the Oxford dictionary defines context from the language perspective as "the parts that immediately precede and follow a word or passage and clarify its meaning". The essence of this type of definitions is that a part of a sentence can be explained by its surrounding parts (for an example of research on context from the language perspective see (Sowa 2003)). The Merriam-Webster dictionary defines context from the environmental perspective as "the interrelated conditions in which something exists or occurs". The essence of this type of definitions is that context describes a situation or event in the environment of something or someone. In this thesis, we view context from an environmental perspective and tailor it towards the computer science domain and then especially towards the ubiquitous computing domain.

Definitions of Context in the Ubiquitous Computing Domain

Context is defined in many different ways in the ubiquitous computing domain. One category of context definitions is definitions by example, often for the purpose of a particular application. For example, Schilit and Theimer (Schilit and Theimer 1994) define context as "location, identities of nearby people and objects, and changes to those objects". This definition is for the purpose of their location-based office application. Lamming and Flynn (Lamming and Flynn 1994) describe context as any information stored in a personal office-oriented "diary". This definition serves as a basis for their "forget-me-not" office application. Brown et al. (Brown, Bovey et al. 1997) define context as location, identities of the people around the user, the time of the day, season, temperature, as the basis for their stick-e notes application. Da Costa et al. (da Costa, da Silva Strzykalski et al. 2005) define context as a storage, network, power and memory parameters of the user's devices for the purpose of their supporting middleware for adaptive mobile applications. From a database perspective, van Bunningen et al. (Bunningen, Feng et al. 2005) define context as a "situation under which user's database access happens".

Another category of context definitions are definitions by example that introduce specific aspects of context. For example, Ebling et al. (Ebling, Hunt et al. 2001) define context as aspects of the physical world and conditions in a virtual world. This definition introduces the environment of the application in the scope of context. Similarly, Bradley and Dunlop (Bradley and Dunlop 2003) define that the scope of context includes the user and his applications. Wei et al. (Wei, Farkas et al. 2003) define context as any information concerning user's mobile device and its capabilities, as well as the networks used and their characteristics. Gray and Salber (Gray and Salber 2001) indicate that context is spatio-temporal information concerning a user, to indicate the importance of time and location for context. Similarly, Chalmers (Chalmers 2004) defines context as information relevant to the user along a timescale. Hence, they identify that context can be current as well as historic. Gloss (Gloss 2005) defines context as network's availability in space and time.

Definitions by example are difficult to apply to a broad range of applications (Dey 2000). Therefore, general definitions of context are proposed. For example, Schmidt and Beigl (Schmidt, Beigl et al. 1999) define context as a "the situation and the environment a device or the user is in". Dey et al. (Dey, Salber et al. 2001) define context as "any information that can be used to characterize the situation of an entity, where the entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves". Although there is no common consensus on the definition of context, Dey's definition is currently the most commonly referred definition of context[7].

The majority of current definitions mix, abstract from, or ignore the difference between context and context information. Additionally, these definitions ignore the perspective of the application that uses context information. Hence in the next section, we propose our interpretation of the concept context and context information.

Our Definition of Context and Context Information

We define **context** by adapting the definition of Schmidt et al. (Schmidt, Beigl et al. 1999). We generalize their definition by abstracting from the environment and generalize the concept of 'device and user' into entity.

Definition 1 Context. *Context is the set of situations an entity is in.*

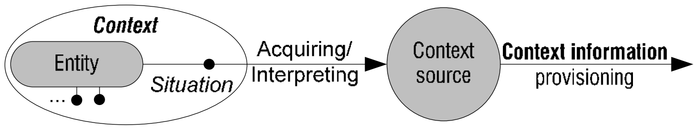
In the definition, **situation** denotes (based on (Dockhorn Costa 2007)) a particular state of affairs that is of interest. The definition of context describes that context is always related to an **entity.** Context does not exist by itself (Dockhorn Costa, Guizzardi et al. 2006). This entity can be a human, place, or physical/virtual object. The entity's context can consists of multiple situations. For example, a person (entity) can be in a room (location context) reading his email (activity context).

However, the majority of an entity's context cannot be used by applications because there are no ways to transform this context into digital information. Context sources may be deployed to acquire, for example by sensing the entity's environment, and interpret the context of an entity. *Figure 2-4* depicts a context source that aquires context information, which they provide to other systems, such as context-aware applications. Hence,

[7] (Dey, Salber et al. 2001) is cited more than 700 times according to Google scholar, http://scholar.google.com, visited 3 September 2007

context information is the representation of the real-world context of an entity.

Figure 2-4 Context vs. Context information.



In our view, context information can be defined from two perspectives: (i) the global perspective and (ii) the application perspective. The first definition considers context information from an abstract viewpoint in which the use of context information is not taken into account. The second definition considers the use of context information by a context-aware application as an important aspect of context information. From the global perspective, we define context information as follows:

Definition 2 Context information from a global perspective.

Context information is information that represents the context of an entity.

Figure 2-5 depicts that context information is related to the applications universe of discourse. From the perspective of the context-aware application, not all available global context information is relevant for the functioning of the application. The goal of the context-aware application determines the meaningfulness of the available global context information for that application. Only the information used for adapting the behaviour of a context-aware application to provide a higher quality service is what we call context information for that application.

Furthermore, for an application, context information is not always available (Dey 2000). Context sources can appear and disappear are arbitrary times during the lifetime of the application. For example, due to the movement of the application out of the range of a context source. Context information is used by the application to provide a higher quality service, which is tailored to the situation of the application or other relevant entities. However, without context information, a context-aware application should be able to function. In that case, it might offer a lower quality service. Hence, we define **context information** from an application perspective as:

Definition 3 Context information from an application perspective.

Context information is information that represents the context of an entity, which is optionally used by an application to adapt its behaviour to provide higher quality service.

Summarizing, the context of an entity consists of the set of all situations this entity is in. Context can be acquired by context sources and transformed into context information. Context information from a global perspective is a subset of an entities' context containing only the situations represented and offered by context sources. Context sources can offer one or more types of context information from one or more entities. Furthermore, multiple context sources can provide information on the same context of an entity. As can be seen from *Figure 2-5*, there exist context sources that provide context information that is outside of the universe of discourse of the application. Context information from an application perspective is a subset of global context information consisting only of information relevant for the application's universe of discourse. Additionally, when global context information is required for an application to function (i.e. when it cannot be omitted), we call this information application data rather then context information.

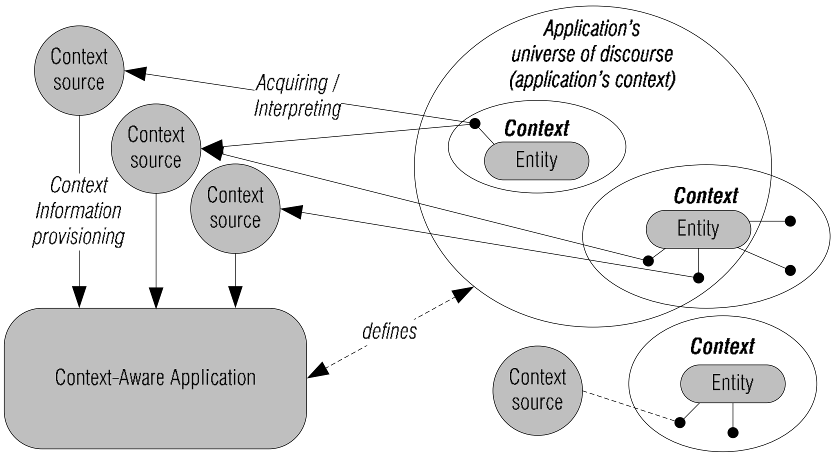


Figure 2-5 Relation of context, context information, context sources and the context-aware application.

High-level model of Context Information

The context information that represents an entities' context consists of different levels of information. *Figure 2-6*, presents a high-level model of context information (Broens, Halteren et al. 2006).

First, context information encapsulates information that describes an entities' context. This consists of an **element** that describes the context type. For example, context information can describe the physical location of a user. It consists of a **value**, for example 52.123/6.23123. Finally, it can consist of a **format** in which the value is expressed, for example 'Latitude/Longitude'.

Second, context information encapsulates information related to what it describes, denoted as relation information. This information consists of the **entity** (i.e. person, place, physical/virtual object) of which it is describing its context. For example, context information can describe the context of Person X, Table Y or Application Z.

Finally, on the meta-level, context information may contain information that describes characteristics of the context it is representing. This meta-information may consist of information on the quality of the context information (called Quality of Context – QoC) or security information (e.g., who is authorized to retrieve this context). Security and QoC related to context-awareness are discussed in more detail in Section 2.5.

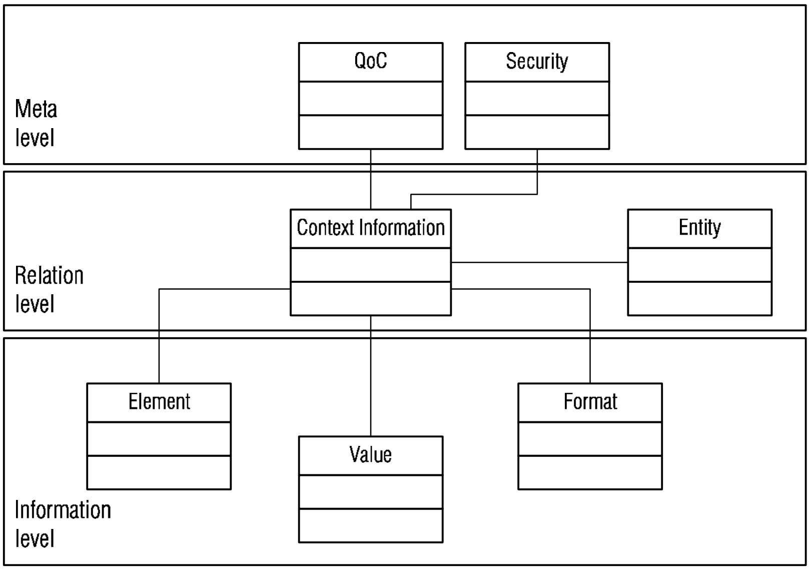


Figure 2-6 Overview of the taxonomy of context information.

2.4 Context-Aware Applications

Similarly to context, there exist many definitions of context-awareness and context-aware applications. In this section, we present a subset of these definitions and present our interpretation of the concept context-awareness and context-aware applications.

Definitions of Context-Awareness

Schilit et al. (Schilit, Adams et al. 1994) define context-awareness mainly as applications that adapt to the user location. This is often also denoted as location-awareness. Lamming and Flynn (Lamming and Flynn 1994) define it as applications that are aware of user's activities in an office environment. Schmidt and Beigl (Schmidt, Beigl et al. 1999) propose a more abstract definition of context-awareness - as an awareness of a "situation the user is in". Dey (Dey, Salber et al. 2001) provides a more generic definition of a context-aware system as "the system (that) uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". Henricksen et al. (Henricksen, Indulska et al. 2005) define context-aware applications as the applications that "adapt to changes in the environment and user requirements". Van Bunningen et al. (Bunningen, Feng et al. 2005) define context-awareness as an awareness of a "situation under which user's database access happens". Similarly, da Costa et al. (da Costa, da Silva Strzykalski et al. 2005) define context-awareness as system self-reflection in terms of as a storage, network, power and memory parameters.

Our Definition of a Context-Aware Application

Dey's definition offers a generic basis for our definition of context-aware applications. However, in our view, it lacks expressing major characteristics of context information relevant for defining context-aware applications.
First, context information should be treated as optional information. The default behaviour of a context-aware application (i.e. context-unaware behaviour) fulfils the basic user's need. However, by adapting to context information, the quality of the offered service improves. If inputted information to the application cannot be treated as being optional, this information is primary application data rather than context information.

Hence, we consider context-aware applications as an extension of non-context-aware applications. Context-aware applications have a basic non-context-aware behaviour, which is adapted when context is used. We assume that a context-aware application can function without context but can do its job better when considering context information.

Secondly, a context-aware application should react on fluctuating availability of context information as context sources are dynamically available during the lifetime of the application.

Finally, as described in Section 2.3, a context of an entity is transformed into context information by context source. A context source can be limited in its capabilities to transform context in context information or may introduce interpretation errors. Hence, context information is inherently imperfect (Dey, Mankoff et al. 2000; Henricksen and Indulska 2004) and has a quality that describes how well it represents the real-world context. A context-aware application should therefore react on the quality of the used context information. Hence, we define a **context-aware application** as:

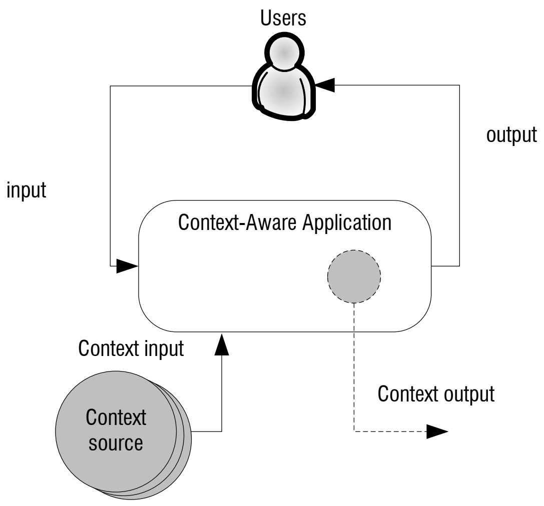
Definition 4 Context-Aware Application.

An application that improves its offered service quality, by executing default behaviour that can adapt, based on context information itself and the availability and quality of this information.

Characteristics of Context-Aware Applications

Figure 2-7 shows a basic model of a context-aware application. When compared to context-unaware applications (see *Figure 2-2*), context-aware applications are characterized by the use of additional **context inputs** to adapt their behaviour. Optionally, context-aware applications can produce context information, which can be made available to other applications via **context outputs**. Therefore, context-aware applications can also act as a context source for other applications when they produce and provide context information.

Figure 2-7 Context-Aware Application.



A context-aware application can adapt to context information from three types of entities: (i) information describing the context of the application itself, (ii) context information describing the context of the application's users and (iii) information describing the context of other entities.

For example, a context-aware follow-me music application can adapt music playback based on context information from all three types of entities. This application adapts to the location where music is played by taking into account the location of the application user (type ii), however the music is paused when other persons are present in the same location as the user (type iii). Additionally, the bit-rate of the played music can be lowered/increased based on the available bandwidth for retrieving the music stream (type i).

Adapting the application behavior to context information can have different forms, which can be combined in a context-aware application:

- Use context information to produce higher quality output. With higher quality, we mean outputs that better suit the goal of the user. For example in case of the follow-me music application, the presence of other users in a room pauses the playback of music.
- Use context information to replace, minimize or tailor the user input. For example, the follow-me music application uses the location of the user to play music only in the rooms he is physically in. The user does not have to switch the music on or off when he is moving between rooms.
- Use context information to adapt the internal behavior of the application. For example, to ensure the performance of the application. The follow-me music application uses different playback bit-rates when the available bandwidth fluctuates during the lifetime of the application.

Context offering, requirement and match

A context source offers context information in terms of **context offerings**. Context offerings contain information on the context information a context source is able to provide. This offering can consist of all or a subset of information described in *Figure 2-6,* such as entity, element and quality. Context-aware application requires context information in terms of **context requirements**. Context requirements specify similar information as a context offering, however then from the application perspective. One actual instance of context information that is exchanged between a context source and context-aware application is called a **context sample.** *Figure 2-8* visually represents these aspects.

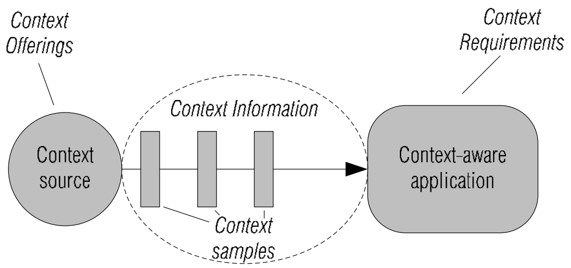


Figure 2-8 Context offerings, requests and context samples.

For a context-aware application to be able to use context information (i.e. consume context samples), a match has to be made between its context requirements and available context offerings. *Figure 2-9* visualizes the process of matching the context offerings and context requirements. Besides context requirements and offerings, the context owner may provide additional constraints on how and when its context is used. This is for example to enforce the privacy of the user. Together these parameters form the input on which a **context match** can be made between a context source and a context-aware application. Creating a context match can be the responsibility of the context-aware application or a third party such as a mediator/broker.

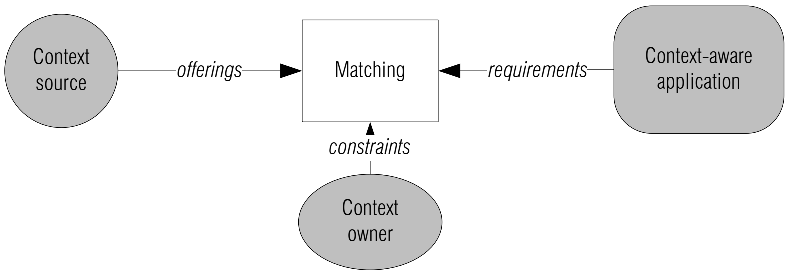


Figure 2-9 Matching of context offering with context requirements.

Example: Context-Aware Epileptic Alarm Application

To illustrate the concepts described in this section and to indicate the difference between a context-unaware and a context-aware application, we continue the example of a healthcare epileptic alarm application (Broens, Halteren et al. 2007). This application offers an alarm service to its user that notifies them of upcoming epileptic seizures. The user provides direct application data inputs to the application via sensors on his body that monitor vital signs such as heartrate and brain activity. Based on the vital sign inputs, the application reasons on possible upcoming seizures. When a seizure is detected by the application, it sends a notification output to the patient (end-user) and to the application part running in the hospital (also a user). Additionally it starts streaming the vital signs to the hospital.

By making this application context-aware, it uses context information to better detect alarms and to improve the interaction with the users. This context information is, for instance, the location of the patient and the communication bandwidth available. This context information is acquired from the context environment of the users. In this case, the location can be retrieved via a GPS context source sensing the patient's physical context environment while the bandwidth is provided by a bandwidth monitoring context source sensing the patient's computation context environment. The location is send together with the seizure notification to the hospital and is used by the doctors to send help to the patient at the right location. The bandwidth is used to tailor the vital signs to the capabilities of the communication platform, such as increasing data compression or decreasing sampling rates, to be able to guarantee the vital signs transfer to the hospital.

2.5 Design Aspects of Context-Aware Applications

In this section, we discuss four key design aspects of context-aware application: context-modelling, quality of context (QoC), context-awareness and security, and context reasoning.

2.5.1 Context Modelling

By introducing context-awareness, applications become increasingly complex and interconnected. This raises the need for context modelling. A **context model** is an information model that represents context information by describing context elements and their relationship. There are several reasons for introducing context models (Dockhorn Costa, Guizzardi et al. 2006):

– *Characterize the application's universe of discourse:* When developing context-aware applications, the used context information should be modelled in a context model. For example, in the case of the epileptic alarm system a nearby caregiver is send to the patient in need. The notion of 'nearby' should be captured in a context model to be able to develop high quality application logic that can deal with this context information.

– *Support common understanding, problem solving, and communication among the stakeholders:* Context information is exchanged between different stakeholders such as the context owner and the application user. For this exchange to be successful, all involved parties should have a common understanding on the shared context information. For example in (Broens, Pokraev et al. 2004), we discuss the need for such a common understanding in the area of service discovery. Here we argue that a high-quality service discovery result can only be achieved with a common context and service model shared between the service provider and service user.

– *Represent context unambiguously:* Context information can have multiple representations. For example, the concept of 'physical location' can be represented as 'location', 'place', 'position' etc. To be able to interpret and reason on context information, a context model is needed to capture concept unambiguously.

Current approaches for context modelling include, amongst others, conceptual modelling (Dockhorn Costa, Guizzardi et al. 2006; Guizzardi 2006), ontological modelling (e.g. with the OWL language) (Wang, Gu et al. 2004), meta-modelling (e.g. with UML) (Broens, Pokraev et al. 2004). Researching approaches to model context information is out of the scope of this thesis.

2.5.2 Quality of Context (QoC)

As identified in Section 2.3 and 2.4, quality of context is an intrinsic property of context and therefore influences the functioning of context-aware applications. **Quality of Context** (QoC) is the measure that indicates how well the offered context information represents the corresponding real-world situation.

In this thesis, we adopt the notions of QoC developed by (Bucholz, Kupper et al. 2003) and (Sheikh, Wegdam et al. 2007). They claim that there are three reasons that motivate the notion of QoC:

– *QoC-based application adaptation:* due the inherent imperfectness of context information (e.g. sensor inaccuracy, interpretation mistakes) the quality of context highly influences how well the application can adapt to context information. When taking an application specific viewpoint, QoC is the measure that indicates how well the application can use offered context information to adapt its behaviour. Therefore, besides the actual context information, QoC should also be available to the application to influence its behaviour.
– *Middleware efficiency:* multiple context sources can deliver similar context information (e.g. multiple location context sources). Based on the QoC the middleware can make selections on which context sources matches best with the application context requirements.
– *User's privacy enforcement:* artificially varying the QoC can be used to preserve the privacy of the entity (user) (see Section 2.5.3). The entity can specify constraints on the use of context. For example, the location of the user can be determined on room-level precision (e.g. Tom is in room 4126). To preserve the privacy of the user the QoC of the provided context samples can be lowered to city-level precision (e.g. Tom is in city Enschede).

Let's reconsider *Figure 2-9*, which indicates that the use of context information requires a match between the context offerings, requirements and constraints. Part of the context requirements, offerings and constraints can be related to QoC and can therefore be mapped onto quality of context concept.

Figure 2-10 shows graph of the QoC of context information over time. QoC can be specified in two ways: (i) specification related to context samples, and (ii) specifications related to context offerings/requirements. The first is what we denote as **actual QoC.** This measure specifies the quality of a single instance of context information and therefore vary over time. The second is what we call **offered QoC** and respectively **required QoC.** These measures specify the potential quality range that can be offered or respectively, the quality range that is required for the application to function. These measures are semi-static and do not change often.

Hence, the context requirements of the application specify a lower limit on the acceptable actual QoC of the offered context information. It does not specify an upper limit because we assume that better QoC results in at least the same but possibly better application behaviour. The context offerings of a context source form a QoC range in which context can be offered. The upper limit of the context offerings minus the constraints (i.e. constraints limit the QoC) form the real upper limit on the offered QoC of the context information. If the actual QoC of the offered context information by a context source is between these limits this context is useful for the application and can be consumed from the context source. We call these limits the **QoC Matching Area**. Outside of these limits, negotiations may take place to lower the requirements of the application or to lower the constraints of the entity. This widens the matching area in which the offered context is useful for the application.

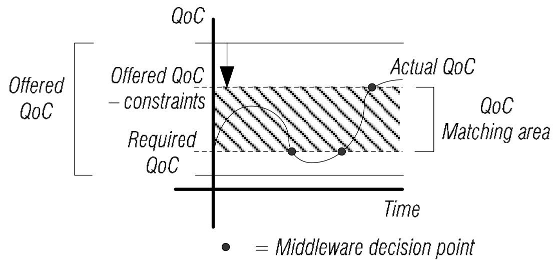


Figure 2-10 influence of QoC on the matching between a context-aware application and a context source.

To explain these concepts we present an example of a person locator application, which indicates the route to a person inside a building based on the location of the searched person and the location of the user. The application specifies that the location has to be minimally precise on 50m. The middleware finds a context source that can offer location with minimum precision of 100m and a maximum precision of 1m. This results in a matching area ranging from 1-50m precision. In the area between 50-100m, negation may take place or the actual context may be rejected by the application or the middleware may find another context source capable of offering the required context.

We can further refine the presented QoC notion by introducing application depended QoC levels as depicted in *Figure 2-11*. These levels determine the type of behaviour of a context-aware application based on the actual QoC. For example, the person locator application can provide a directional arrow to the searched person when the precision is between 5-50m. When the precision is higher than 5m, it can show a map with the possible route to the searched person.

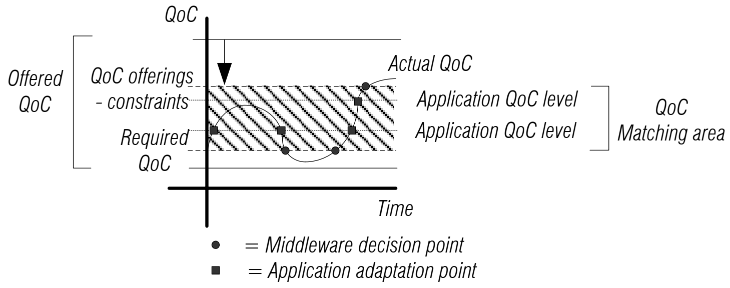


Figure 2-11 influence of QoC on a context-aware application.

The abstract notion of QoC can be refined into the following QoC indicators (Sheikh, Wegdam et al. 2007):

- *Precision:* "granularity with which context information describes the real world situation." For example, location of 'Tom' can be determined with a precision of 10m.
- *Freshness:* "time that elapses between the determination of context information and its delivery to a requestor." For example, location of 'Tom' is determined 5 minutes ago.
- *Temporal Resolution:* "the period of time to which a single instance of context in formation is applicable." For example, the location of 'Tom' is valid for the coming half hour.
- *Spatial Resolution:* "the precision with which the physical area, to which an instance of context information is applicable, is expressed." For example, Tom's location is expressed on the room-level or Tom's location is expressed on the city-level.
- *Probability of Correctness:* "the probability that an instance of context accurately represents the corresponding real world situation, as assessed by the context source, at the time it was determined." For example, the probability of correctness of Tom's location is at least 80%.

In this thesis, we mainly focus on the role of QoC for the application adaptation and then specifically on how does the QoC influence the matches between context sources and context-aware applications, and how to facilitate applications to adapt to QoC. The application strategy to really adapt its behaviour to QoC is out of the scope of this thesis.

Additionally, we recognize two main challenges in dealing with QoC, however consider them out of scope for this thesis:

- *Representation of QoC:* to be able to use QoC measures, the application needs to understand the QoC indicators and values specified by the context source, and visa versa. This corresponds with the common problem of semantic interoperability discussed earlier.
- *Determining QoC measurements:* determining actual QoC measures of context information at run-time is challenging.

2.5.3 Context-awareness and Security

In a context-aware system, multiple physical entities are involved. *Figure 2-12* shows the relationships of the different entities. First, there are the entities that own the context information (**context owner**) that is distributed to context-aware applications via context sources. Often this context information specifies the situation of this owner, however also other entities may own context information of other entities. Secondly, there are the users of the context-aware application that, however implicitly, use context information.

Both parties are subject to security risks. For example, the privacy of the owner of the context information is at risk because of the possible malicious use of their context information. Examples of privacy risks are unauthorized tracking of the user's location, profiling and identity theft of the user. On the other hand, the user of the context-aware application is also at risk because it can use a context-aware application that adapts to context information from un-trusted context sources.

Hence, on the one hand, context-aware environments have the opportunity to deliver users with higher quality services. However, on the other hand, it is also recognized that using context information may introduce a security risk for the users (Campbell, Al-Muhtadi et al. 2002; Hong and Landay 2004; Robinson, Vogt et al. 2004; Brey 2005; Neisse, Wegdam et al. 2007). In the remainder of this section, we discuss two main security aspects in more detail: privacy and trust.

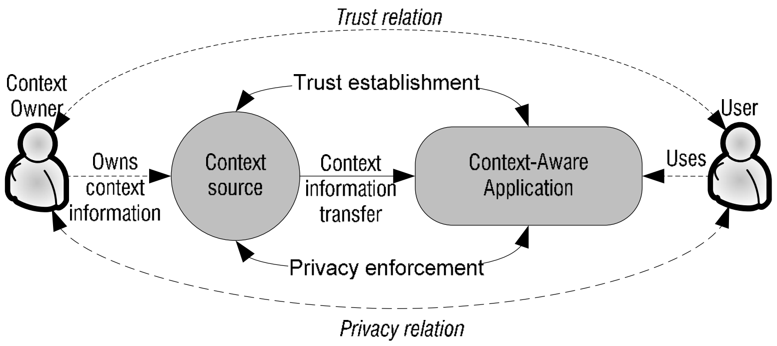


Figure 2-12 Relating context-awareness with privacy and trust.

Context-Awareness and Privacy

The Oxford dictionary defines privacy as "a state in which one is not observed or disturbed". When transformed to the context-awareness domain, **privacy** can be described as the state in which the context information of an entity, which may be exploitable information, cannot be disclosed unauthorized.

Therefore, in a privacy-based context-aware system, the owner of the context information should be aware of how their context is being acquired (access control) and how it is going to be used (context handling). However, users should not be disrupted too much from using legitimate and desirable context-aware systems. The challenge is to find a balance between the controlled release of context information and the usability of controlling the user's privacy. For example, by enabling the user to set privacy policies. Too much user privacy control may result in inflexible and invasive context-aware systems, while too little privacy control may result in loss of privacy (Wegdam, van Bemmel et al. 2006).

So, on the one hand, disclosure of context needs privacy enforcement mechanisms but on the other hand context information can be used to make the context privacy control less obtrusive (e.g. context-aware access control (Hulsebosch, Salden et al. 2005)). Overcoming the privacy sensitive nature of context information is out of the scope of this thesis.

Context-Awareness and Trust

Chamber's dictionary defines **trust** as the "belief or confidence in, or reliance on, the truth, goodness, character, power, ability, etc of someone or something". For a successful context-aware application, there has to be a trust relation between the involved entities. A trust relation can be defined as a subjective measure to represent the believe of a '**trustor'** concerned with a certain '**trustees'** behaviour and focused on a certain trust aspect (Abdul-Rahman and Hailes 2000). Trust aspect in context-awareness can mainly be divided into three aspects (Neisse, Wegdam et al. 2007):

- *Identity provisioning:* trust of the trustor in the identity of the thrustee
- *Context information provisioning:* trust of the trustor in the quality of the offered context information.
- *Privacy enforcement:* trust of the trustor in the quality of the privacy enforcement by the trustee.

For example, a trust relation between Tom (trustor) and Ricardo (trustee, context owner) may contain all the previously mentioned aspects. Tom may trust that Ricardo is who he says he is (identity provisioning). Tom may also trust Ricardo to provide high quality context information (context information provisioning) while not using this context for malicious use (privacy enforcement). Establishing trust relations for exchanging context information is out of the scope of this thesis.

2.5.4 Context Reasoning

Another property of context information is that it can be used to deduce other context. The process of deducing entailed context information from different sources of context is called **context reasoning** (Benerecetti, Bouquet et al. 2000; Kranenburg, Salden et al. 2005). There are two types of context reasoning, which may be combined:

- *Vertical reasoning:* deduce higher-level context information from more primitive context sources. For example, 'speed' can be deduced from combining 'travelled distance' and 'Elapsed time'.
- *Horizontal reasoning:* deduce higher-quality context information. For example, improve the precision of 'Location' by reasoning (e.g. averaging) on location information from multiple location sources.

The process of context reasoning is represented in the commonly adapted layered model of a context-aware system (Ailisto, Alahuhta et al. 2002). *Figure 2-13* presents this layered model. The model consists of five layers: physical, context data, semantic, inference and application layer. On the physical layer context sensors produce raw context data. On the context data layer this raw context data is processed into context information. On the semantic level annotate the context information with semantic information such it can be used for further reasoning. This includes storing it for further use. On the inference level the semantic annotated data is used to deduce entailed context information. On the application level this deduced information can be used for tailoring the application behaviour. We have to note that different application parts encompassing a context-aware system may support different combination of layers from this model. Furthermore, other variations (three/four layered models) of the presented model exists, for example (Baldauf, Dustdar et al. 2004; Henricksen, Indulska et al. 2005). Context reasoning is out of the scope of this thesis.

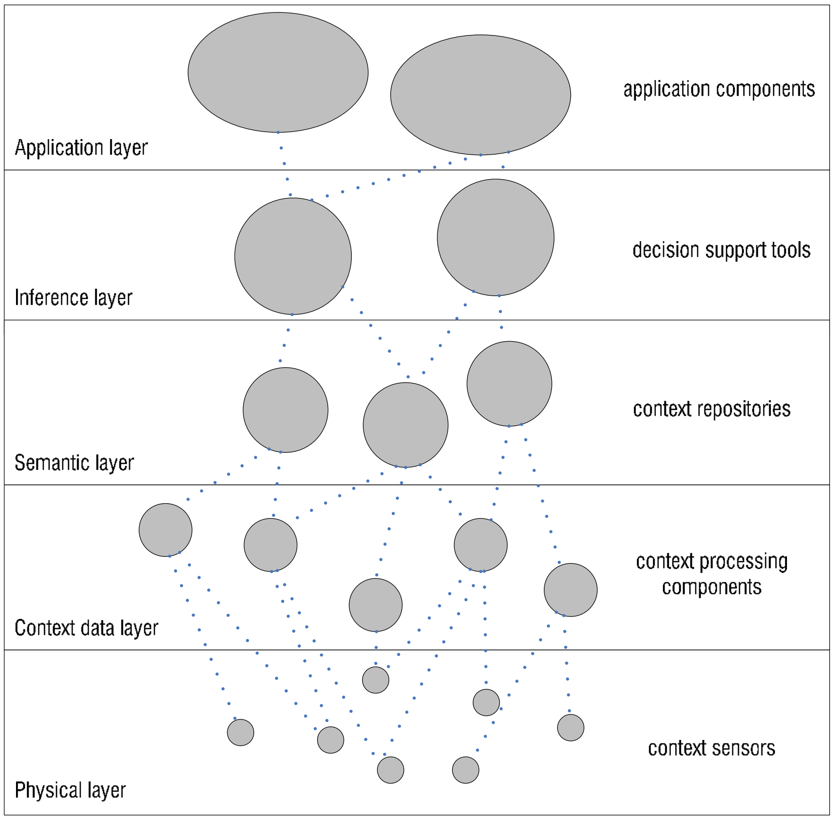


Figure 2-13 Layered model of a context-aware system.

We explain reasoning and the layered model with a healthcare example. *Figure 2-14* presents the reasoning process of this example. Consider a system that measures heart activity. Electro Cardio Gram (ECG) sensors measures physical signals and provide the system with the result in volts (layer 1). The system can now, by aggregating this voltage and time (layer 2), construct an ECG diagram (layer 3). The next step could be to use this diagram to infer the heartbeat in beats per minute (layer 4). For instance, by combining the heartbeat, sweat production, and the activity the user is doing (i.e., watching TV) it can be inferred if a patient is suffering from an epileptic seizure, or it can even be predicted if a patient suffers from an epileptic seizure (layer 4). The application can notify caregivers based on this seizure context (layer 5).

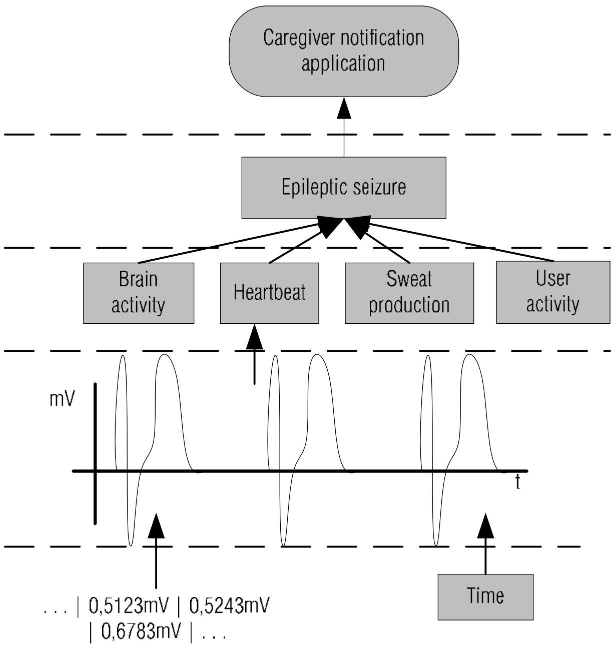

Caregiver notification application
Epileptic seizure
Brain activity
Heartbeat
Sweat production
User activity
mV
t
. . . | 0,5123mV | 0,5243mV | 0,6783mV | . . .
Time

Figure 2-14 Reasoning example in the healthcare domain.

2.6 Modelling Context-Aware Applications

In this section, we present basic models of context-aware applications. These models are further refined in the remainder of this thesis.

Context-Aware Applications

In this section we take a top-down approach. *Figure 2-15* presents a high-level black-box model of a context-aware application and its supporting context middleware.

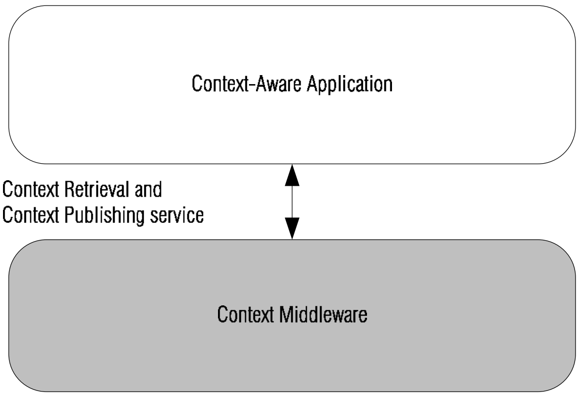


Figure 2-15 Context-aware application and its supporting context middleware.

A context-aware application uses context information to adapt its behaviour. Furthermore, it can produce context information. The context middleware facilitates these aspects by offering a context retrieval and context publishing service. The **context retrieval service** facilitates the context-aware application to retrieve context. The **context publishing service** facilitates the context-aware application to publish its context to the context environment. For example, other context-aware applications.

The context-aware application is further detailed in *Figure 2-16*. It consists of two main functional elements: (i) application logic and (ii) context logic. **Application logic** is the behaviour of the application that fulfils the users need. This behaviour can adapt to context information it consumes and possibly can produce context. **Context logic** is the behaviour needed for the application logic to retrieve the required context information or publish its offered context information.

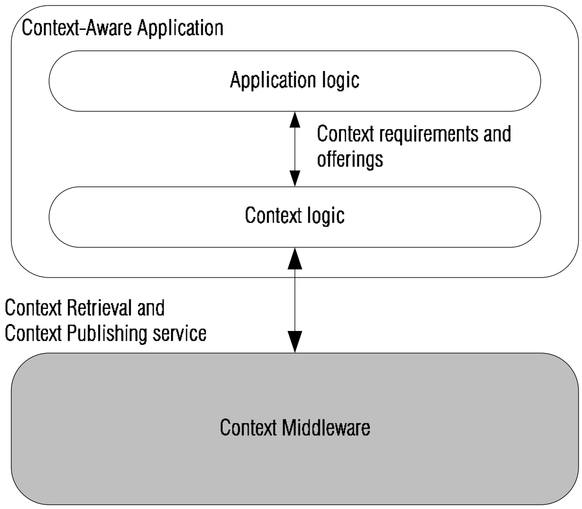
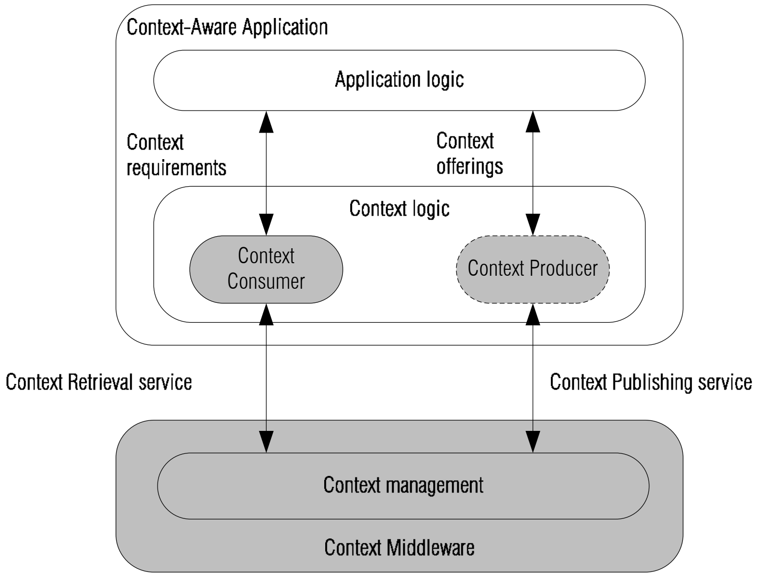

Context-Aware Application
Application logic
Context requirements and offerings
Context logic
Context Retrieval and
Context Publishing service
Context Middleware

Figure 2-16 Detailing the context-aware application.

Figure 2-17 details the context logic. The context logic consists of two functional elements. First, the **context consumer element** consists of behaviour to retrieve context required by the application logic. For an application to be context-aware, it requires to have a context consumer functional element. Secondly, the context logic can consist of a **context producer element**. The context producer element is optional and consists of behaviour to publish the offered context information of the application logic. In this thesis, we also use the term **context consumer and producer role**. With this we indicate that a context-aware application consists of a context consumer and respectively context producer element, or indicate a context source when it only has a context producer element.

Figure 2-17 Detailing the context logic.



Both the context retrieval service and the context publishing service are provided by the context middleware. We denote the specific middleware functionality that provides these services as **context management**. Context middleware may also consist of other elements like communication and security mechanisms. These are out of the scope of the model presented in this chapter.

Context Sources and Context Binding

Figure 2-18 shows a model of a context source. Context information is offered by context sources. We model these **context sources** similarly to context-aware applications. It consists of specific application logic responsible for sensing, acquiring and processing context information into context offerings. The context logic has a mandatory context producer element that is responsible to publish the offered context produced by the application logic.

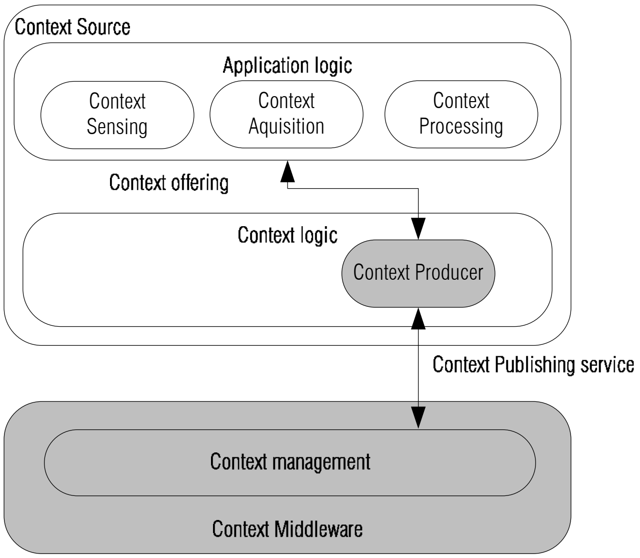


Figure 2-18 Model of a context source.

A context-aware application X can appear as a context source for context-aware application Y that is using the context of context-aware application X. However, there also exist non-context-aware applications that are context sources. They have as sole purpose producing context. For example, an application part that wraps a GPS to produce location information.

Figure 2-19 shows the relationship of a context-aware application and a context source. A context-aware application has context requirements that are fulfilled by its context consumer functional element, using the context retrieval service. A context source offers its created context via the context publishing service to the context middleware. The middleware is responsible for facilitating the association (i.e. determine a context match) of a context consumer (context-aware application) with a suitable context producer (context source). We define a context binding as:

Definition 5 Context Binding.

A context binding is the required association between a context consumer and a context producer, which is needed for context information exchange, resulting from a context binding process.

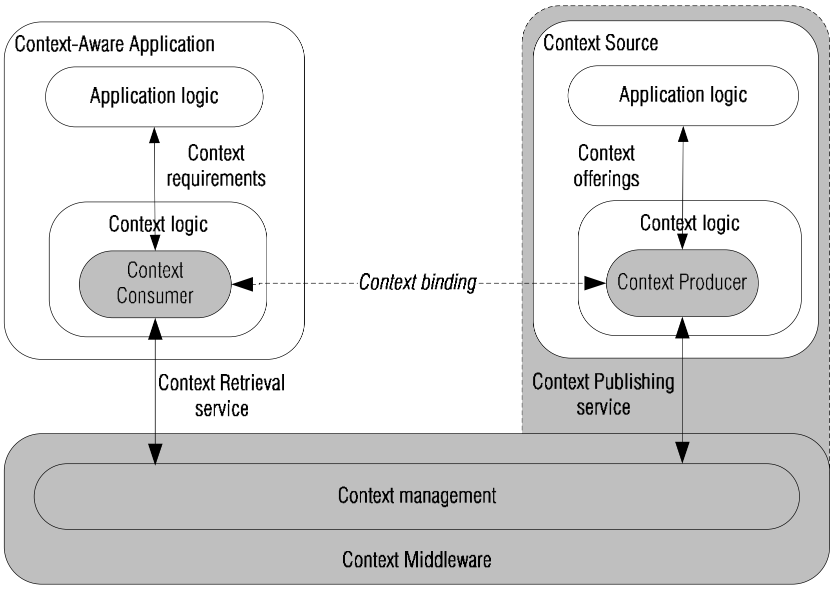


Figure 2-19 Relation of a context-aware application with a context source.

2.7 Context Binding Process

Creation and maintenance of a context binding requires a comprehensive binding process. First, we identify the phases in this binding process. Additionally, we identify several challenges that influence the availability and quality of a possible binding. We related the importance of the phases with the identified challenges.

2.7.1 Phases in a binding process

We distinguish several phases in a generic binding process to create and maintain context bindings. *Figure 2-20* discusses the phases and artefacts in this binding process. Grey rounded rectangles represent phases of the binding process. White rectangles represent artefacts used by or resulting from the functions execution in the phases.

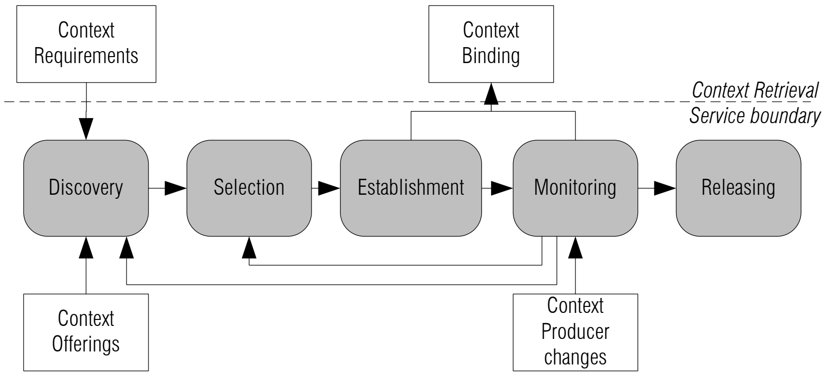


Figure 2-20 Context Binding process.

The following phases and corresponding functions can be distinguished:

- *Discovery:* Functions that have as goal to find context producers that match with the user's context requirements. This includes:
  - Processing context requirements from the user.
  - Extract basic context requirements (e.g. Location of Tom) and possible QoC criteria on the required context information (e.g. precision > 2m).
  - Find suitable context producers, advertised with their context offerings, from available context producers in local, remote or federated repositories.
- *Selection:* Functions that select suitable context producers. This includes:
  - Rank the set of context producers resulting from the discovery phase.
  - Select one or more suitable context producers based on pre-defined or user-based criteria.
- *Establishment:* Functions that create a context binding with a selected context producer and makes this binding available to the user. This includes:
  - Create a context binding to the selected context producer. This may include creating an intermediary proxy for controlling certain aspects of the binding. For example, including functions to enforce privacy, ensure QoS, context reasoning, or start of a rebinding process in case of disappearing context producers.
  - Make the created context binding available to the user. For example by notifying the user.
- *Monitoring:* Functions that actively monitor a created binding (i) on fluctuating availability and QoC and (ii) newly appearing context producers. This includes:
  - Monitor an established binding on disappearing of bound context producers and possibly initiate another discovery phase.

  - Monitor the availability of new context producers and possibly initiate another selection phase to compare the old bound producer and the newly available producer.
- *Releasing:* Functions that destroy an established binding. This can be explicit releasing in case of a destroy request or implicit releasing when the application of the user terminates.

In the model, we assume that the context offerings are already influenced by possible context constraints (see Section 2.4). Consequently, the set of context offerings available for discovery, excludes offerings of producers that, due to constraints of the owner, cannot be used.

2.7.2 Characteristics of context bindings

A binding process that takes into account all the previous mentioned functions is important, due to some inherent challenges for creation and maintenance of context binding:

1. Variety of distributed context sources:
   - The environment of the context-aware application may contain multiple distributed context sources that are capable of offering the required context information.
2. Heterogeneity of context sources:
   - Context sources may offer (similar) context using different representations (i.e. context models), and access mechanisms.
3. Dynamic availability of context sources:
   - After establishing a binding between a context-aware application and a context source, the mobility of the user and consequently of its context-aware application may result in the unavailability of bound context sources.
   - After binding of a context-aware application with a context source, the mobility of context sources may result in the unavailability of bound context sources.
4. Dynamic availability of the context information a context source can offer:
   - Sensor failure may result in the unavailability of context information a context source can provide.
   - Effectuation of context-aware privacy access mechanism may result in the unavailability of context information delivered by bound context sources.
   - Context-aware applications may require context information from other entities than itself. Due to the mobility of these entities, bound context sources may not be able to offer context related to these entities.
5. Fluctuating quality of context sources:

- Sensing and acquisition errors of raw context information, or misinterpretation of the raw context information, yields imperfect context information that has fluctuating QoC.
- Multiple context sources may provide similar context with different offered quality of context.
- After binding, the quality of context samples a context source provides may differ over time.
- After binding, new context sources may appear in the context environment with possibly better quality of context.

Table 2-1 relates these challenges to the distinguished binding phases. Discovery is required to enable applications to find a suitable context source from the variety of available distributed context sources. This includes a selection from a set of suitable context sources. A binding has to be established between an application and one or more context sources. Establishment of a binding has to ensure that these two can interoperate.

Overcoming dynamic availability and fluctuating QoC requires a complete binding process. After establishment the binding has to be monitored. In case of a failing binding, for example due to loss of a context source or degrading QoC, a new source has to be found and hence a (re-)binding process is stared. The final three challenges are the motivation for this research. Aspects to deal with these challenges are elaborated further in the remainder of this thesis.

Table 2-1 Rating the importance of the binding phases to overcome binding challenges.

Binding phase / Binding challenge	Discovery	Selection	Establishment	Monitoring	Releasing
Variety of distributed context sources	●	●	-	-	-
Heterogeneity of context sources	-	-	●	-	-
Dynamic availability of context sources	●	●	●	●	●
Dynamic availability of context information	●	●	●	●	●
Fluctuating quality of context sources	●	●	●	●	●
Legend: '●', '-' ='important', 'not important'					

Chapter 3

State-of-the-Art on Context Middleware

This chapter discusses the state-of-the-art in context middleware. We focus especially on middleware mechanisms that facilitate the creation and maintenance of context bindings. With this we mean mechanisms that (partially) support the discovery, selection, establishment, monitoring and releasing of context bindings. Parts of this chapter are published in (Broens, Halteren et al. 2006).

This chapter is organized as follows: Section 3.1 presents a general overview of middleware, especially focussing on component-based middleware. Section 3.2 discusses a relevant subset of currently available context middleware systems. Section 3.3 gives a general overview of the AWARENESS middleware architecture and discusses its four proposed context discovery mechanisms. Finally, Section 3.4 gives conclusions on the current capabilities of context middleware systems for the creation and maintenance of context binding. Furthermore it discusses how we can leverage from these capabilities and where extensions can be made.

3.1 Middleware

Distributed applications consist of application parts that are connected by communication platforms. With communication platforms we denote hardware and software that enable interactions amongst application parts. **Middleware** is commonly referred to as a software layer between applications and communication platforms with the general goal of facilitating the development of distributed applications. This goal can be divided in (Emmerich, Aoyama et al. 2007):

– Provide interoperability between distributed applications across heterogeneous communication platforms;

– Support programming abstractions that hide complexities of building distributed applications.
– Offer common building blocks that relieve the application developer from solving recurring problems.

Evolution of Middleware

Remote Procedure Call (RPC) can be seen as is the foundation of the majority of current middleware technologies. RPC is introduced in the time of imperative programming (1980's) by Birell and Nelson (Birrel and Nelson 1984). RPC hides for the developer the fact that an invoked procedure call is handled by a remote party instead of a local party. They propose a two-tier system that consists of a client, which is a program that calls a remote procedure, and a server, which is a program that implements the invoked remote procedure. Additionally, they introduce many concepts used in current middleware technologies, like Interface Definition Language (IDL), Name and Directory service, service interface and stub. A recent example of pure RPC-based middleware is XML-RPC (Apache Webservices project 2003). XML-RPC implements the transport of the remote procedure call with HTTP and uses XML as the data format to encapsulate a procedure call.

There are several enhancements to RPC-based middleware (Emmerich, Aoyama et al. 2007). Traditional RPC based middleware supports only synchronous communication. To support asynchronous communication, Message Oriented Middleware (MOM) uses messages and message queues to transfer RPC's. Another enhancement is Transaction Processing Monitors (TP-Monitors), which extend traditional RPC with transaction capabilities.

With the evolution from the imperative programming paradigm to the Object-Oriented programming (OO) paradigm (1990's), RPC based middleware is extended with the notion of objects. This type of middleware is called Object-Oriented middleware. Although the goal of OO middleware is similar to RPC middleware, the client does not invoke a procedure but a method of an object that is possibly exposed with an interface. Due to the characteristics of OO such as inheritance and polymorphism, the function the server actually performs depends on the object that implements the remote method. Well known examples of OO middleware are RMI (Sun 2003) and CORBA (OMG 2004). Currently, middleware is evolving into component-based middleware, which we discuss in more detail in the next section.

For a more thorough discussion on the evolution of middleware for distributed systems, we refer to (Emmerich, Aoyama et al. 2007) and (Alonso, Casati et al. 2004).

Component-based Middleware

Component-based middleware views an application as a composition of components. The main idea behind components is re-use and composition of application code, which can be recognized in the Latin word '*componere*' meaning "to put together". A **component** is defined by Szyperski (Szyperski 1998; Szyperski, Gruntz et al. 2002) as "a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties".

When analyzing the presented definition, we distinguish the following characteristics of a component (Wegdam 2003):

- *Explicit context dependencies:* specifies what the deployment environment needs to provide to allow the component to function, including required interfaces from other components. For example, an epileptic detection component could specify that it needs a vital sign analyzing component to function.
- *Contract and interfaces:* specifies functional and non-functional aspects of a component. A contract is typically an interface that specifies operations annotated with pre- and post-conditions and possibly invariants. For instance, an epileptic detection component could specify an operation "detectSeizure" which requires an integer ECG signal in the range of -10 mV to 10mV.
- *Unit of deployment:* a component is an application part that is actually deployed. This requires an environment including lifecycle management functionality to (un)deploy components.
- *Third party composition:* a component can be composed by third parties. For example, an epileptic detection component can be used in a patient application but also in the application of a healthcare professional.

Figure 3-1 presents an overview of a generic component-based middleware architecture. An application is a composition of components. These components are encapsulated by **containers** that offer an execution environment. Basic functions a container provides are lifecycle management functions such as installing, starting and stopping of components. An application can consist of multiple distributed components residing in different containers.

Components are deployed in the container using a component descriptor. The **component descriptor** indicates the capabilities the component can offer (i.e. component interface) and the capabilities it requires from the middleware or other components. The component descriptor is used during the deployment of the components. Components interact with the container using the **container interface**. The container can offer certain **middleware services** such as the advertising and discovery of component services. The container (and hence also components) interacts with clients using some type of communication platform. Examples of currently available component-based middleware technologies are Corba Components (OMG 2002), J2ME (Sun 2005), J2EE (Sun 2005) and OSGi (OSGi Alliance 2004).

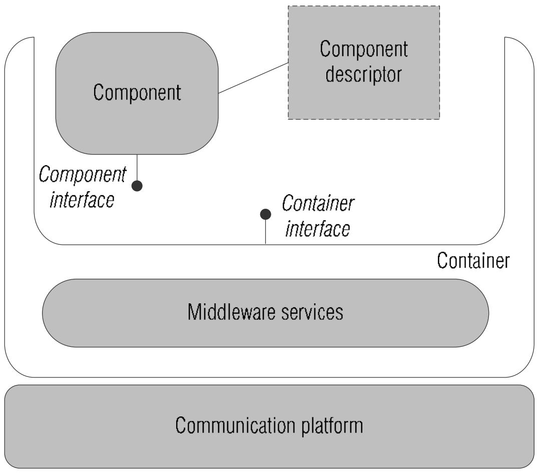


Figure 3-1 Generic component-based middleware architecture.

Figure 3-2 relates the component-based middleware paradigm with context-awareness. If we consider an application as application behaviour that based on user inputs creates user outputs, context-aware applications additionally use context inputs offered by a context producer to provide higher quality output. If we apply the component-based paradigm, a context-aware application becomes a composition of context-unaware and context consuming and/or context producing components that may have individual context requirements and context offerings.

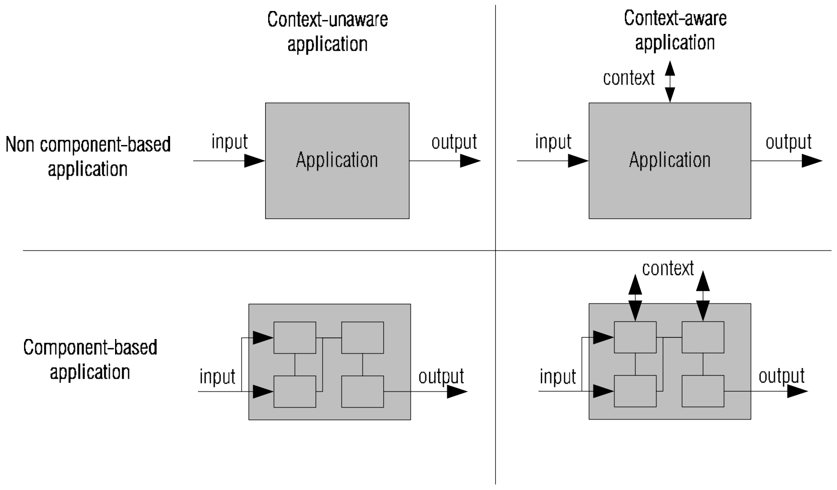

Context-unaware application
Context-aware application
context
Non component-based application
input
Application
output
input
Application
output
context
Component-based application
input
output
input
output

Figure 3-2 Component-based context-aware applications.

Middleware for Context-Aware Applications

Middleware has the potential to overcome challenges that developers of context-aware applications face:

- Context producers and context consumers are distributed on possible heterogeneous communication platforms. Additionally, context producer and consumers are heterogeneous in the way they offer and transfer context information. Middleware infrastructures can facilitate the exchange of context information by providing interoperability between heterogeneous context producer, consumer and communication platforms.
- There is a need for common abstractions that hide the complexity of developing context-aware applications (e.g. binding, security, context reasoning). These abstractions can be offered in the form of context middleware.
- Recurring problems like context discovery, securing context information, context-based adaptation and binding can be bundled into generic middleware building blocks.

We define **context middleware** as an extension of the earlier presented middleware definition:

Definition 6 Context middleware.

Context middleware is an intermediary software layer, between context consumers and producers, and communication platforms, which has as goal to reduce the complexities of distributing context information to facilitate the development of context-aware applications.

Current context-aware middleware infrastructures are mainly OO-based solutions. We claim that a direction towards component-based middleware approaches is beneficial for the easy development of context-aware applications. The general advantages of component-based development of applications also apply for the development of context-aware applications:

- *Reusability of components:* Components are well-defined encapsulated units of programming that can be easily reused. Hence, context-aware application development can benefit from this aspect.
- *Third party composition:* When developing a context-aware system, context consumers and context producers are typically distributed and subject to third party composition. Components are well suited for this third party composition.
- *Unit of deployment:* components execute in a run-time environment this means that on deploy-time this environment can execute certain functionality. For context-aware applications this could for example includes initializing context bindings and setting security policies. This may results in a possible decreasing amount of explicit interactions of the application with the middleware.

3.2 Current Context Middleware Systems

There exist many context middleware systems that (partially) facilitate the creation and maintenance of context bindings. For the interested reader, we refer to the following papers for an extensive overview of current context middleware systems (Chen and Kotz 2000; Henricksen, Indulska et al. 2005; Baldauf, Dustdar et al. 2007).

In this section, we discuss a small subset of context middleware systems, which we consider representative for the spectrum of context middleware systems. We discuss successively: the Context Toolkit (Dey, Salber et al. 2001), Solar (Chen and Kotz 2002), Pace, (Henricksen, Indulska et al. 2005), Java Context-Awareness Framework (JCAF)(Bardram 2005) and the Context Management System (CMS) (Ramparany, Poortinga et al. 2007).

For every system, we provide a high-level architectural overview followed by a discussion on their context binding capabilities. These capabilities are identified using the binding phases as identified in Section 2.7.

3.2.1 Context Toolkit

The Context Toolkit (Dey 2000; Dey and Abowd 2000; Dey, Salber et al. 2001) has pioneered in providing generic support for the exchange of context information. Main elements in the context toolkit architecture are: context widgets, sensors, aggregators, interpreters and discoverers (see *Figure 3-3*).

A context widget encapsulates a single physical sensor that produces context information. It offers an abstraction to enable applications to uniformly retrieve context information, independently from the specific sensor technology. A context aggregator can be used to perform horizontal reasoning by aggregating multiple context samples from different context widgets. A context interpreter can be used to do vertical reasoning by providing higher-level context information based on lower-level context information provided by context widgets (e.g. speed, based on time and distance). A discoverer can be used to locate a specific context widget/aggregator or interpreter. Furthermore, it enables applications to be notified of appearing and leaving context widgets.

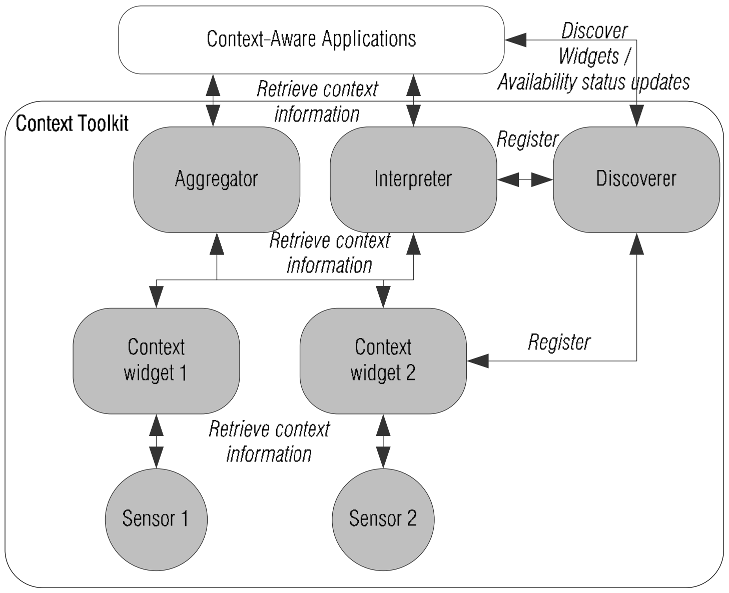

Context-Aware Applications
Discover Widgets / Availability status updates
Retrieve context information
Context Toolkit
Register
Aggregator
Interpreter
Discoverer
Retrieve context information
Context widget 1
Context widget 2
Register
Retrieve context information
Sensor 1
Sensor 2

Figure 3-3 Overview of the of the Context Toolkit architecture.

Discussion

In *Table 3-1*, we identify the context binding capabilities of the Context Toolkit, based on the identified phases and functions in a generic context binding process (see Section 2.7).

Table 3-1 Context binding capabilities of the Context Toolkit.

	Discovery	**Selection**	**Establishment**	**Monitoring**	**Releasing**
Context Toolkit	●	x	●	●	x
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

The context toolkit was one of the first middleware mechanisms to offer support for context information exchange. It offers a basic discovery mechanism to locate context widgets, without support for QoC criteria. Selection of widgets is the responsibility of the application. Establishment of a binding is performed by providing a reference of a widget to the application. The context toolkit monitors for the availability of context widgets. Applications can register for notifications of leaving (i.e. unregistering) and appearing (registering) widgets. However, decisions (i.e. re-binding) on how to react on these situations are the responsibility of the applications. Statements on releasing of context bindings are not explicitly mentioned.

3.2.2 Solar

Solar (Chen and Kotz 2002; Chen and Kotz 2003; Chen, Li et al. 2004) is a mechanism for distribution of context information in large-scale peer-to-peer sensor network. Elements in the Solar architecture are: sensors, planets, operators, channels and directories (see *Figure 3-4*).

Sensors may connect to so-called planets to advertise their context offerings. Planets form an execution environment for operators, which are data processing blocks, which together form a peer-to-peer network. Context-aware applications can also connect to planets to retrieve context information. Exchange of context information between sensors, operators and applications is done via channels. A planet offers generic services to its connected sensors and applications. One of them is a directory service which a context-aware application can use to locate operators that can provide certain context information.

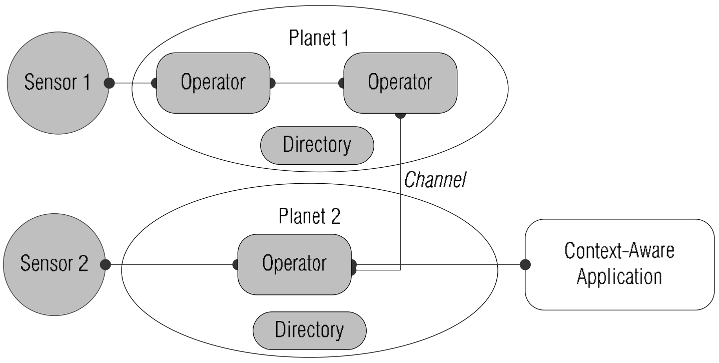


Figure 3-4 Overview of the Solar architecture.

Discussion

In *Table 3-2*, we identify the context binding capabilities of Solar, based on the identified phases and functions in a generic context binding process (see Section 2.7).

Table 3-2 Context binding capabilities of Solar.

	Discovery	Selection	Establishment	Monitoring	Releasing
Solar	●	●	●	x	x
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

Solar offers basic support for applications to discover operators that can produce the required context information. The concept QoC is out of the scope of Solar. When applications request context information, Solar selects an operator path (i.e. composition of operators) that is suitable for the context requirements of the application. Establishment of a context binding is done by providing a reference to an operater that can traverse context information from the determined operator path. Switching of planets (i.e. operator paths) when planets disappear or new planets become available is the responsibility of the application. Releasing of context bindings are not explicitly mentioned.

3.2.3 Pace

The Pace middleware (Henricksen and Indulska 2004; Henricksen, Indulska et al. 2005) offers supporting mechanisms and tools for the development of context-aware applications. The Pace middleware consists of (see *Figure 3-5*):

– *Context management system:* functions to aggregate and store context information. Additionally, it provides functions to discover context information. The context management system consists of multiple distributed context repositories, which a context-aware application can query or subscribe to. Usage of access control mechanism on the repository can be configured.
– *Preference management system:* functions to store user preferences. Based on these preferences and context information, stored in the context management systems, it triggers certain actions.
– *Programming toolkit:* functions that enable application developers to easily specify actions that should be triggered by the preference manager.
– *Messaging framework*: functions to facilitate the communication between the different components of the middleware and context-aware applications.
– *Schema compiler toolset:* tools that enable application developers to generate code, based on application specifications, that eases the interaction with the Pace middleware.

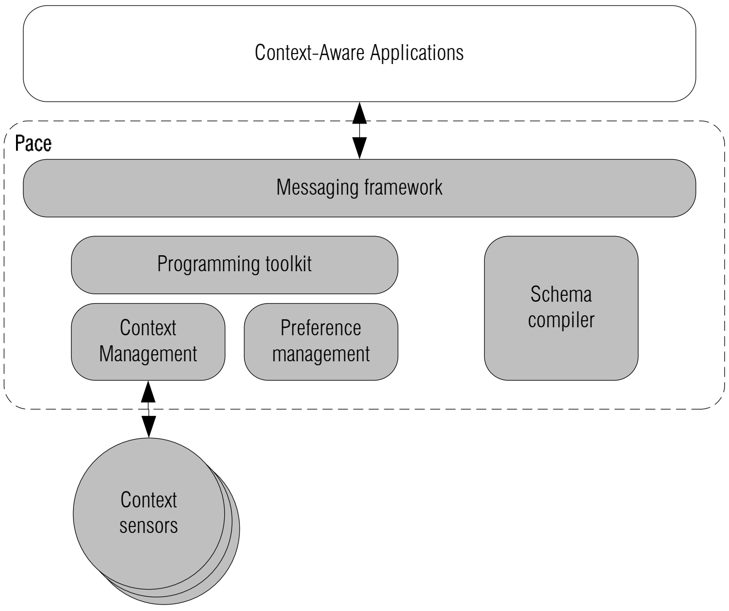


Figure 3-5 Overview of the Pace architecture.

Discussion

In *Table 3-3*, we identify the context binding capabilities of Pace, based on the identified phases and functions in a generic context binding process (see Section 2.7).

Table 3-3 Context binding capabilities of Pace.

	Discovery	Selection	Establishment	Monitoring	Releasing
Pace	●●	●●	●	x	x
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

The Pace middleware offers support for the discovery of context information via the context management functions. The context model they use to describe context information contains the notion of QoC. However, it is unclear how this is used in the discovery and selection process. Selection and establishment is implicitly performed by querying the databases that are filled with context information. Appearing and disappearing context producers are out of the scope of Pace. Releasing of context bindings or not explicitly mentioned.

3.2.4 JCAF

The Java Context-Awareness Framework (JCAF) offers a light-weight programming framework for the development of context-aware applications. Elements in JCAF are (see *Figure 3-6*): sensors, entities, context clients, context services, context monitor, entity repository and transformer repositories.

There are two types of context clients: the first retrieves context information (i.e. context-aware application) and the second produces context information (i.e. sensor). The latter type wraps a sensor in a software component, together with a context monitor that acquires context information from the sensor. This context information is exposed to an application as context service. Every context type is represented as an 'entity' and stored in the entity repository. Additionally, a context service can consist of transformers that aggregate context information from multiple entities or infer context information stored under a new entity. Access control mechanisms can be implemented in a context service. The context consuming context client can request context information with a request/response mechanism or via a subscribe/notify mechanism.

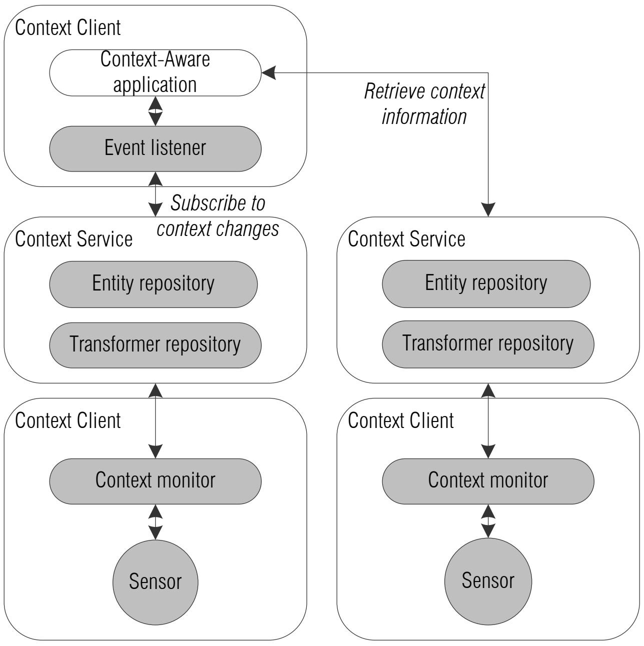


Figure 3-6 Overview of the JCAF architecture.

Discussion

In *Table 3-4*, we identify the context binding capabilities of JCAF, based on the identified phases and functions in a generic context binding process (see Section 2.7).

Table 3-4 Context binding capabilities of JCAF.

	Discovery	Selection	Establishment	Monitoring	Releasing
JCAF	●	x	●	x	x
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

The JCAF middleware provides a generic programming framework for handling context information. However, it does not provide a specific discovery mechanism to discover context services. For this purpose it reuses the Java RMI registry facility. In the discovery, QoC criteria are not dealt with. Establishment of a context binding is done by providing a reference to a context client. Additionally, fluctuating availability of context services and the quality of context information is not taken into account. Explicit statements on releasing a context binding are not made.

3.2.5 CMS

The Context Management System (CMS) (Ramparany, Poortinga et al. 2007) is an infrastructure for managing context information. Elements in CMS are (see *Figure 3*-7): sensors, context wrappers, context broker, context store, and context interpreter.

A context-aware application can request (i.e. request/response and subscribe/notify) context information via the context broker. The context broker stores the capabilities of sources of context that advertise their offerings via a context wrapper. A context wrapper is a software component that interacts with a sensor and that can transfer context information to a context-aware application. Additionally, CMS supports the storage of context information in the context store. This store can be queried to retrieve context history or can be used as a basis for context interpretation (i.e. context reasoning) by the context interpreter.

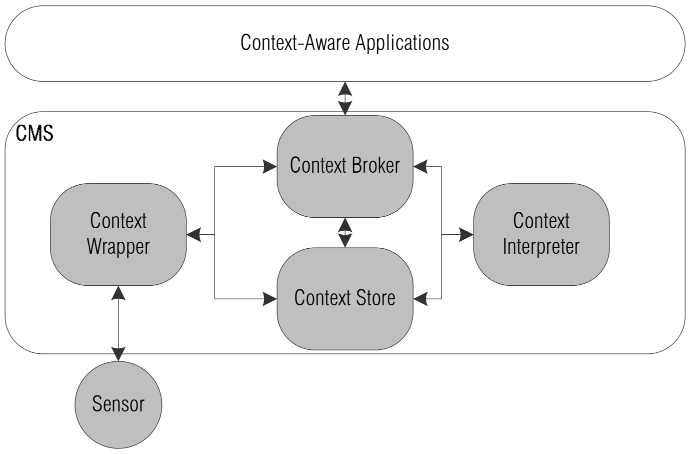


Figure 3-7 Overview of the CMS architecture.

Discussion

In *Table 3-5*, we identify the context binding capabilities of CMS, based on the identified phases and functions in a generic context binding process (see Section 2.7).

Table 3-5 Context binding capabilities of CMS.

	Discovery	Selection	Establishment	Monitoring	Releasing
CMS	●●	x	●	x	x
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

CMS offers discovery of context wrappers via the context broker. In the discovery request there is basic support for QoC criteria. Selection of a suitable wrapper is the responsibility of the application. Establishment of a binding is performed by providing a reference to the wrapper. Fluctuating availability and quality of wrappers is out of the scope of the CMS. Explicit statements on releasing are not made.

3.2.6 Discussion

The discussed context middleware systems offer partial support for a comprehensive context binding process. Their focus is mainly on the discovery of context sources. Establishing a context binding and reacting on possible fluctuating availability and quality of context information is left to the developer of context-aware applications. Our research focuses on tackling these issues as part of the AWARENESS project (see the next section). However, AWARENESS also developed several other context management mechanisms, which we discuss and incorporate in the previously started comparison.

3.3 Awareness Context Middleware

The goal of the Awareness project (Sinderen, Halteren et al. 2006; Wegdam, Sinderen et al. 2008) is to develop a middleware-based infrastructure to facilitate the development of context-aware, mobile applications. In this section, we first give an overview of the AWARENESS architecture and position our work in this architecture. Additionally, we discuss the four context management mechanisms developed in AWARENESS, which we believe give a representative and innovative overview of current context management solutions.

3.3.1 Overall AWARENESS Architecture

Figure 3-8 shows the basic layered architecture of the AWARENESS infrastructure. Two layers can be distinguished: (i) the context-aware mobile application layer (white), and (ii) the context infrastructure layer that offers generic support functions (grey) to the application layer.

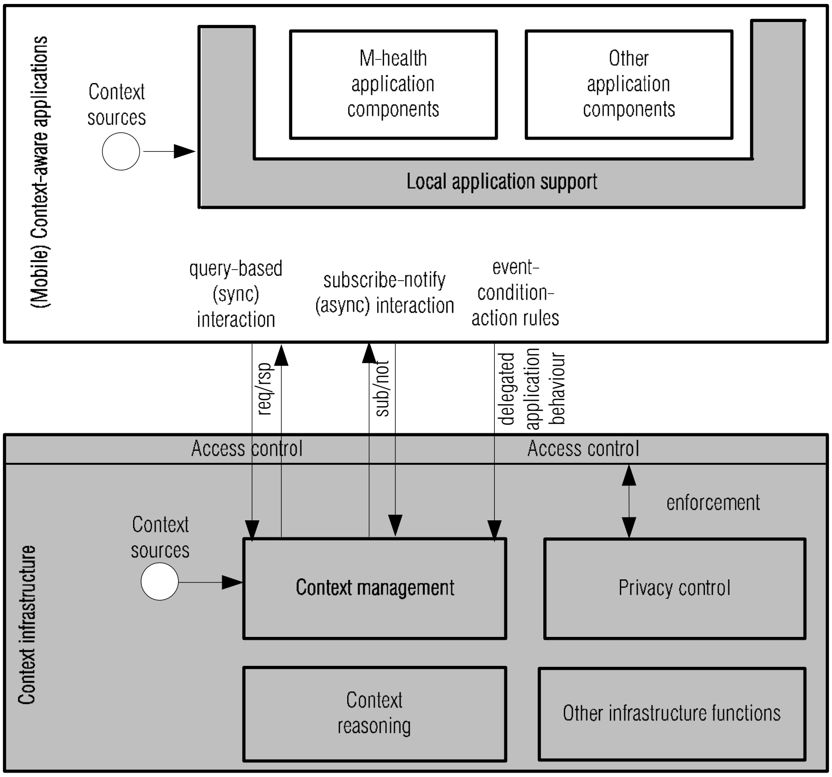

(Mobile) Context-aware applications
Context sources
M-health application components
Other application components
Local application support
query-based (sync) interaction
subscribe-notify (async) interaction
event-condition-action rules
req/rsp
sub/not
delegated application behaviour
Access control
Access control
enforcement
Context infrastructure
Context sources
Context management
Privacy control
Context reasoning
Other infrastructure functions

Figure 3-8 Overall awareness architecture (Sinderen, Halteren et al. 2006; Wegdam, Sinderen et al. 2008).

The following infrastructure functions are identified:

- *Context reasoning:* The context reasoning functionality (Sinderen, Verheijen et al. 2007) is responsible for combining context information from different sources. In this way the quality of the context information can be improved (horizontal reasoning), or 'higher level' context information from more primitive context information can be derived (vertical reasoning).
- *Context management:* The context management functionality (Benz, Hesselman et al. 2006) is responsible for discovering and exchanging context information. It offers request-response and publish-subscribe interaction patterns to the application layer to allow access to context information. In addition, application developers can delegate parts of the application behaviour by issuing application-specific behaviour rules that are executed in the infrastructure.
- *Privacy control:* The privacy control functionality (Wegdam, Brok et al. 2007) enables end-users to control their privacy. It ensures that privacy sensitive information that is trusted to the infrastructure can only be accessed after the user has given consent for this. The context information can be anonymized or obfuscated (i.e. reducing the Quality

of Context) before providing it to the application layer. In addition, it provides identity management, context-aware trust management and adaptive security functions.

– Local application support: The local application support functions are co-located with the application components, and provide a local container to facilitate the development of context-aware application components. Although the local application support is part of the application layer, it is not application dependent. The mechanism proposed in this thesis are part of the local application support functions. Another local application support function that is researched is the dynamic reconfiguration of health signal processing tasks on available processing nodes (Mei and Widya 2007).

In the remainder of this section, we elaborate on the different context management solutions developed in the Awareness project (Benz, Hesselman et al. 2006). This includes the Context Management Framework (CMF), the Cumular Context Server (CCS), the Context Distribution Framework (CDF) and the JXTA based infrastructure (JEXCI).

3.3.2 CMF

The Context Management Framework (CMF) (Benz, Hesselman et al. 2006; Hesselman, Tokmakoff et al. 2006; Kranenburg, Bargh et al. 2006) enables context-aware applications to discover context information of an entity based on *identities*. An identity is a name that uniquely identifies an entity in a pervasive environment (e.g. user@domain). This identity is coupled to a context agent that aggregates all context information available from the entity that is represented by the identity. The context agents are the single point of access to context information for context-aware applications.

Figure 3-9 presents a general overview of the CMF architecture. An application can, in three steps, obtain context information of an entity. First it queries the CMF with the identity (e.g. Tom@UT) of the entity from which it requires context information. The CMF returns a context agent that has gathered all the context sources that offer context information from that entity. Secondly, the application queries the context agent for the context information that he requires (e.g. location) and optionally specifies QoC requirements. The context agent creates a context proxy that acts as a middleman between the application and the context source, which is responsible for delivering the context information to the user and enforcing privacy policies. Finally, the application can retrieve context information by querying the context proxy.

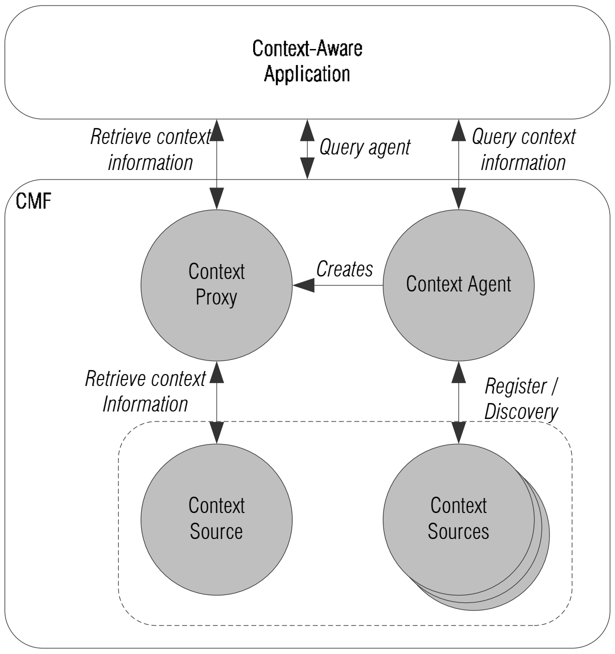

Context-Aware Application
Retrieve context information
Query agent
Query context information
CMF
Context Proxy
Creates
Context Agent
Retrieve context Information
Register / Discovery
Context Source
Context Sources

Figure 3-9 Overview of the CMF architecture.

Discussion

In *Table 3-6*, we identify the context binding capabilities of the CMF, based on the identified phases and functions in a generic context binding process (see Section 2.7).

Table 3-6 Context binding capabilities of CMF.

	Discovery	Selection	Establishment	Monitoring	Releasing
CMF	●●	x	●	x	●●
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

CMF offers discovery capabilities based on identities and context types. Context sources can be registered to context agents but context agent can also discovery context sources. The selection of a suitable context source is the responsibility of the application. When a selection is made by the application, the CMF associates the context source with the application via a context proxy in which privacy enforcement is taken into account. Monitoring of the availability and quality of context producers is also the responsibility of the application. Additionally, detection of newly appearing context sources is the responsibility of the application. Releasing of a context binding is implicitly done when the context proxy is not used anymore.

3.3.3 CCS

The goal of the Cumular Context Service (Benz, Hesselman et al. 2006; Brok 2006) is to offer a scalable solution for context information collection and reasoning. It offers a centralized solution targeted towards the mobile phone operator domain, based on database technology. Elements in the CCS architecture are the CCS core, context sources, and northbound and southbound adapters (see *Figure 3-10*).

The CCS core consists of a database to store context information. Northbound adapters are components that offer context-aware applications access to the CCS core, to enable them to retrieve context information. Southbound adapters are components that offer access to the CCS core, to context sources, to enable them to publish context information. North and southbound adapters are custom made components to be able to communicate with specific applications and context sources.

The main functions that the CCS core supports are: (i) context information storage, (ii) selection of a suitable context source based on QoC requirements specified in the context information request, (iii) access control based on privacy policies, (iv) notifications of context change, (v) heuristic based reasoning and aggregation, and (vi) buddy management for policy selection.

Context sources continuously fill the database of the CCS core with context information (via the southbound adapters) independently from context-aware applications. A context-aware application is coupled to an application specific northbound adapter, which can be used the retrieve context information (i.e. either with a request/response or subscribe/notify interaction mechanism). The northbound adapter transforms the request of the application towards SQL statements required for the CCS core. The CCS core queries its context tables and returns the most suitable context information (based on QoC requirements).

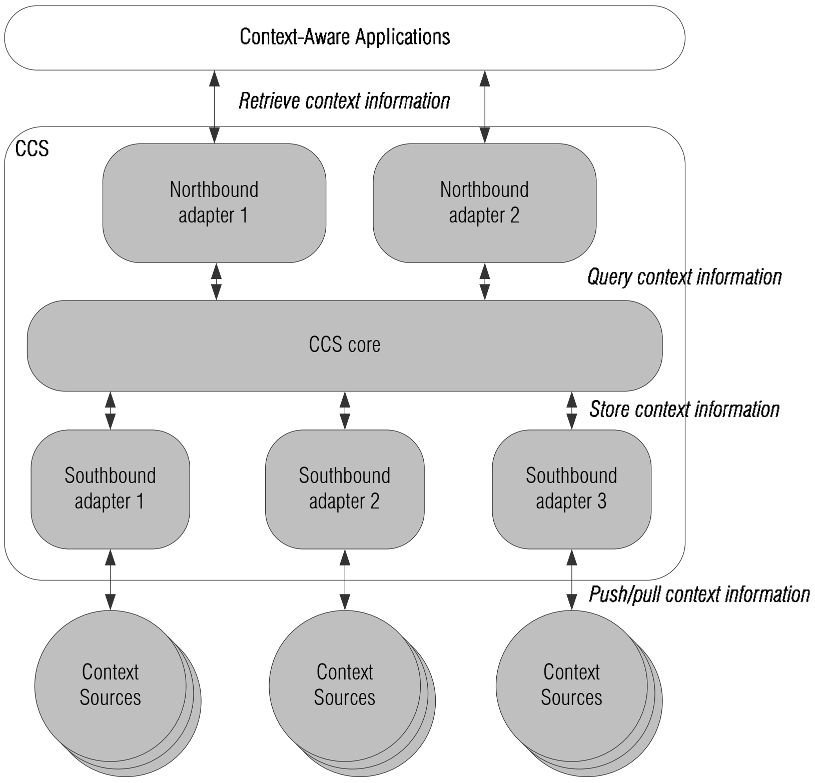


Figure 3-10 Overview of the CCS architecture.

Discussion

In *Table 3-7*, we identify the context binding capabilities of the CCS, based on the identified phases and functions in a generic context binding process (see Section 2.7).

Table 3-7 Context binding capabilities of CCS.

	Discovery	Selection	Establishment	Monitoring	Releasing
CCS	●●	●●	●	x	●●
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

The CCS enables context-aware applications to discover context information based on entity, context type and optionally QoC requirements. Based on the QoC requirements, it selects suitable context information from the available context sources in the CCS core. The CCS checks for access restrictions defined by the context owner. The CCS considers statically connected context sources. Hence, appearing and disappearing context sources (i.e. relevant in the monitoring phase) are out of its scope. Explicit statements on releasing are not made.

3.3.4 CDF

The Context Distribution Framework (CDF) (Benz, Hesselman et al. 2006; Pawar, Halteren et al. 2007) provides a framework for service oriented context-aware applications that are hosted on mobile devices. Features this framework offers are: (i) off-loading of resource intensive context computations from the mobile device, (ii) selection of suitable context sources based on QoC requirements and (ii) modelling of mobile context sources as services. Elements in the CDF architecture are context sources, context services, service directory and the context distribution service (see *Figure 3-11*).

Context sources are represented in the CDF as context services. These context services (de)register their context offerings to a service directory. The context distribution framework, discovers and selects the most suitable context source (ranking algorithm based on a Euclidian distance function using the provided QoC parameters) on behalf of the user (i.e. the context-aware application). Alternatively, a user can request a reference to a context source, and then he is responsible for selecting from a ranked set of sources. When a context source disappears, its reference is removed from the registry. If the user requests context information, the CDS employs a fail-safe mechanism. When the top ranked source is not available the CDS returns context information from the second best context source, and so on. In case of a request for a context source, handling disappearing context sources is the responsibility of the user. When new context sources appear, they are registered in the service registry and notified to the CDS, which ranks them in the set of possible context sources.

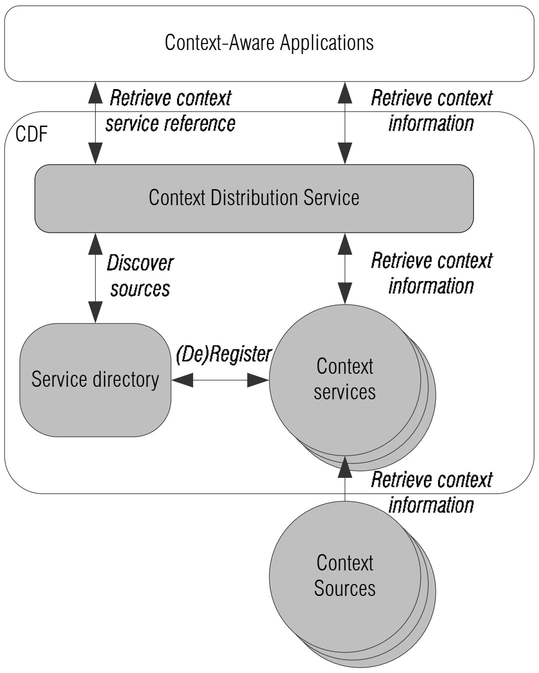

Context-Aware Applications
Retrieve context service reference
Retrieve context information
CDF
Context Distribution Service
Discover sources
Retrieve context information
Service directory
(De)Register
Context services
Retrieve context information
Context Sources

Figure 3-11 Overview of the CDF architecture.

Discussion

In *Table 3-8*, we identify the context binding capabilities of the CDF, based on the identified phases and functions in a generic context binding process (see Section 2.7).

Table 3-8 Context binding capabilities of CDF.

	Discovery	Selection	Establishment	Monitoring	Releasing
CDF	●●	●●	●	●	●●
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

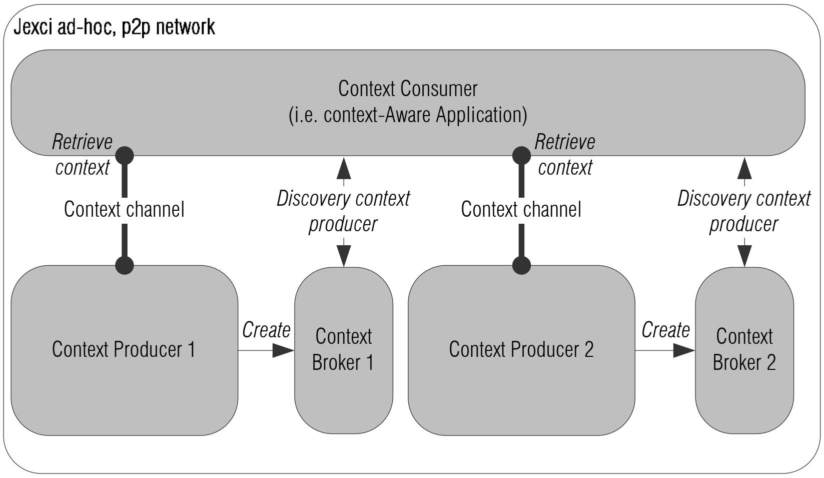
CDF enables applications to discover context information based on entity, type and optionally QoC requirements. The CDF deploys a ranking algorithm to order context sources. The application can chose to do the selection itself or let CDF select a suitable context source. CDF does not create an intelligent establishment (i.e. access control, QoC monitoring) between a context-aware application and context source. When the application chooses to let CDF select a suitable context source, disappearing of this context source is monitored and, based on the previously established ranking, context from another context source is returned. However, when selection is performed by the application also monitoring of disappearing context sources becomes its responsibility.

Additionally, monitoring and rebinding in case of dropping QoC is out of the scope of CDF.

3.3.5 Jexci

The JXTA based infrastructure (Benz, Hesselman et al. 2006) is an infrastructure to facilitate the distribution of context information in ad-hoc and peer-to-peer networks. Core technology used by this mechanism is the JXTA framework[8] that let nodes communicate in peer-to-peer networks. Elements in the JEXCI architecture are context consumers, context producers, context brokers and context channels (see *Figure 3-12*).Context producers create a context broker which is registered as a service to the JXTA network. A context consumer discovers context brokers by issuing a JXTA discovery request to the JXTA network. The consumer can request a single context information value or subscribe to context changes at the context broker. For the transfer of context information, the context broker creates a context channel between the context consumer and context producer, if access control policies allow this.

Figure 3-12 Overview of the Jexci architecture.



Discussion

In *Table 3-9*, we identify the context binding capabilities of Jexci, based on the identified phases and functions in a generic context binding process (see Section 2.7).

[8] http://www.sun.com/software/jxta/

Table 3-9 Context binding capabilities of Jexci.

	Discovery	Selection	Establishment	Monitoring	Releasing
Jexci	●	X	●	X	●●
Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase					

Jexci enables applications to discover context information using entity and type. However, QoC requirements cannot be specified. The selection of suitable context producers is the responsibility of the context-aware application. Jexci creates a context channel and broker as middleman between a context consumer and producer, which performs access control. Appearing and disappearing context producers are out of the scope of Jexci.

3.4 Conclusions

This section gives conclusions on the state-of-the-art analysis presented in this chapter. We give a categorization of the analyzed context middleware systems and compare them based on the identified phases in a context binding process. Additionally, some final remarks are discussed in which we identify capabilities of current context middleware systems. We reflect on these capabilities to determine where these systems can be improved to better facilitate the creation and maintenance of context bindings.

3.4.1 Categorization of Current Context Middleware Systems

Based on the analyzed context middleware systems, we observe two dimensions along which context middleware systems can be categorized: (i) type of context discovery mechanism and (ii) application scope. The first dimension refers to the interaction of the context-aware application with the context middleware system to find context sources. The second dimension refers to the type of context-aware applications the context middleware system supports.

The first dimension (context discovery mechanism) can be divided into the following two categories: (i) information-based and (ii) proxy-based context middleware systems. The first category consists of context middleware systems that interact with the context-aware applications in terms of a context information request, which results in direct retrieval of context information (i.e. Pace, CCS, CDF). Often, these mechanisms consist internally of (multiple distributed) databases which are filled with context information, independently of the requirements of context-aware applications. Major advantage of this approach is that histories of context information can be easily collected. Additionally, discovery, selection and establishment of context sources are performed on behalf of the application. Hence, the application is shielded from a major part of the context binding process.

The second category (proxy-based) consists of context middleware systems that interact with the context-aware application in terms of a context information request, which results in retrieval of one or more context source proxy objects (i.e. Context Toolkit, Solar, JCAF, CMS, CMF, CDF, Jexci). The proxy object can be used to request or subscribe to context information. This approach gives more control over the context information exchange process to the application, compared to the information-based context management systems. However, it is up to the specific implementation of the context middleware what additional steps of the context binding process are hidden from the application.

The second dimension (application scope) can be divided in the following categories: (i) infrastructure-based and (ii) peer-to-peer context management systems. Infrastructure-based context management systems (e.g. Pace, CCS, CDF, Context Toolkit, JCAF, CMS, CMF) are build to support applications that operate in a managed communication environment (e.g. ethernet, WLAN). Peer-to-peer context middleware sytems (e.g. Solar, Jexci) support applications in an ad-hoc communication environment. Main difference between these categories of context middleware systems is that infrastructure-based context middleware systems have the availability of centralized repositories to register and advertise the offering of context sources. In peer-to-peer context middleware systems repositories are local at the peer nodes and advertised to available neighbors. Hence, this makes the discovery process of both types of context middleware systems fundamentally different.

In *Table 3-10*, we summarize the analyzed context management systems according to the discussed dimensions and categories.

Table 3-10 Categorizations of the analyzed context-management systems.

Context Discovery Mechanism / **Application Scope**	**Information-based**	**Proxy-based**
Infrastructure-based	Pace, CCS, CDF	Context Toolkit, JCAF, CMS, CMF, CDF
Peer-to-Peer	-	Solar, Jexci

3.4.2 Comparison of Current Context Middleware Systems

In *Table 3-11*, we summarize the context binding capabilities of the discussed context middleware systems, based on the identified phases and functions in a generic context binding process (see Section 2.7).

All discussed context middleware systems have basic mechanisms to directly discover (i) context information (i.e. information-based) or (ii) proxy objects (i.e. proxy-based) that can be used to retrieve context information. The majority enables the application developer to specify QoC criteria that influence the discovery and possibly the selection process. Especially, the information-based context middleware systems offer selection mechanism to select suitable context sources on behalf of the application. The majority of the proxy-based mechanisms leave the responsibility of context source selection to the application. The responsibility of establishing a binding between a context-aware application with a suitable context source is mainly left to the application. Although many discussed discovery mechanisms acknowledge the importance of dealing with appearing and disappearing context sources, active monitoring of established bindings on disappearing and appearing context sources is left a responsibility of the context-aware application. Additionally, fluctuating QoC of retrieved context samples is not taken into account by the context middleware systems.

Table 3-11 Comparing binding capabilities.

	Discovery	Selection	Establishment	Monitoring	Releasing
Context Toolkit	●	x	●	●	x
Solar	●	●	●	x	x
Pace	●●	●●	●	x	x
JCAF	●	x	●	x	x
CMS	●●	x	●	x	x
CMF	●●	x	●	x	●●
CCS	●●	●●	●	x	●●
CDF	●●	●●	●	●	●●
Jexci	●●	x	●	x	●●

Legend: '●●', '●', 'x' ='comprehensively', 'partially', 'does not' implement functions from this phase

3.4.3 Final Remarks

From the previous analysis, we conclude that current context middleware systems partially offer support for a complete context binding process. Current systems focus primarily on the discovery phase of this process and offer basic abstractions for the establishment of a context binding. However, we claim that, the largely ignored, selection, monitoring and releasing phases are equally important (see *Section 2.7.2*) to overcome the dynamic availability of context sources and information, and the fluctuating quality of context.

Hence, we see an opportunity for a comprehensive context binding infrastructure that besides discovery also covers the selection, establishment, monitoring and releasing phases of the context binding process. However, the fact that retrieving context information encompasses a comprehensive context binding process should ideally be hidden as much as possible for the application developer. When the application developer is enabled to specify his context requirements, we believe that the responsibility of retrieval of this context information can be shifted to an infrastructure-based context binding mechanism (see Chapter 5) and can be made transparent for the developer (see Chapter 4).

The discussed context middleware systems provide generic solutions for context discovery from which we can benefit. However, the majority of these systems are infrastructure-based which means that they only offer support for context-aware applications that operate in the scope of the infrastructure. These context-aware applications can only use context information available in that infrastructure. We believe that different environments require different context middleware mechanisms to exchange context information (Hesselman, Benz et al. 2008). Consequently, during the life-span of, especially a mobile, context-aware application, this application moves between different environments and may encounter different context middleware systems from which it should be able to retrieve context information. Hence, we see an opportunity for a context discovery interoperability mechanism (see Chapter 6) that facilitates context-aware applications to use different context middleware systems, which they encounter, to retrieve context information.

Chapter 4

Context Binding Transparency

This chapter describes the Context Binding Transparency (CBT) and gives an overview of the services and language that we propose to realize this transparency. We discuss the context retrieval and publishing services which expose the CBT to application developers. Additionally, we present the context binding description language (CBDL), which can be used to specify context requirements. Parts of this chapter are published in (Broens, Quartel et al. 2007).

This chapter is structured as follows: Section 4.1 discusses the concept of 'transparency'. Section 4.2 gives a high level overview of the features that CBT offers to developers of context-aware applications. Section 4.3 presents the context retrieval and publishing services. Section 4.4 presents the Context Binding Description Language (CBDL). Finally, Section 4.5 discusses the development of context-aware applications using the aforementioned mechanisms.

4.1 Transparency

Intuitively, transparency denotes that something is transparent, meaning that it can be seen through. For example, transparency can be witnessed when looking through a glass door, or by looking through an oven window that shows the food that is being prepared.

Nevertheless, the semantics of the concept transparency is overloaded. It has different meanings when viewed from different perspectives. For example, from the optical perspective, transparency is defined by the Oxford dictionary as "the condition of allowing light to pass through so that objects behind can be distinctly seen". From an organizational perspective, a transparent organization denotes an organization in which its internal products and process can be inspected. From a computer science perspective transparency is defined by the Open Distributed Processing

Reference Model (ODP-RM) as "the property of hiding from a particular user the potential behaviour of some parts of a distributed system" (Blair and Stefani 1998; Joaquin.net 2007).

When comparing these definitions, we distinguish a contradicting interpretation of the concept transparency. On the one hand the concept transparency focuses on *revealing* something (optics, organisational perspective), while on the other hand transparency focuses on *hiding* something (computer science perspective). Hence, the computer science perspective on transparency can be perceived as counter intuitive and requires some additional elaboration.

Transparency in Computer Science

As presented before, transparency in computer science is introduced in the ODP-RM. The purpose of ODP-RM is to define standards for the design and development of open distributed systems. ODP-RM considers distributed objects that interact via heterogeneous communication platforms. This raises all sorts of distribution-related development problems for application developers such as locating objects, failure of objects and consistency of objects.

These problems have to be dealt with such that distributed applications can function properly. Dealing with these problems could be done purely at the application level. However, some of the solutions for these problems are not application-specific and apply for a range of applications. Consequently, such functions may be shifted to generic infrastructures such as middleware systems like the ones discussed in Chapter 3. Advantages of shifting functionality to a generic infrastructure can be: decreasing development complexity, time, costs and fault rate.

Infrastructures can realize transparencies for application developers. When an infrastructure takes over the responsibility of dealing with a particular development problem, this problem is (partially) hidden for the application developer. Application developers can focus primarily on their application-specific problems at hand. Their application becomes 'transparent' for the hidden development problem.

The ODP-RM defines eight distribution transparencies that have as a goal to reduce development effort by hiding complexities of interacting distributed objects (Blair and Stefani 1998):

- *Access:* "…mask differences in data representations or invocation mechanisms to enable interworking between objects."
- *Location:* "…masks the use of information about location in space when identifying and binding to interfaces".
- *Failure:* "…masks, from an object, the failure and possible recovery of other objects (or itself), to enable fault tolerance."

- *Migration:* "…masks, from an object, the ability of a system to change the location of that object. Migration is often used to achieve load balancing and reduce latency."
- *Relocation:* "… masks relocation of an interface from other interfaces bound to it."
- *Replication:* "… masks the use of a group of mutually behavioural compatible objects to support an interface. Replication is often used to enhance performance and availability."
- *Persistence:* "… mask, from an object, the deactivation and reactivation of other objects (or itself). Deactivation and reactivation are often used to maintain the persistence of an object when a system is unable to provide it with processing, storage and communication functions continuously."
- *Transaction:* "…masks coordination of activities amongst a configuration of objects to achieve consistency."

An example of a system that realizes a location transparency is the Corba naming service (OMG 2004). This service couples identifiers to the physical location (e.g. IP-address) of distributed objects. An application that uses the naming service can interact with objects by referring to them with this identifier instead of the physical address. Hence, the application becomes transparent for the physical location of the objects it interacts with.

Transparency and Abstraction

Abstraction is the act of ignoring certain development aspects to focus on others which are (at that point in time) more important. For example, when considering the systems and services concepts as introduced in Chapter 2, a developer can design a system by recursively zooming into the sub-systems that constitute the overall system. In the development of a particular sub-system, he can treat other sub-systems from an external perspective only considering the services they offer, thereby abstracting from how they are realized.

We argue that transparency is a specific form of abstraction. A developer uses services provided by a realized system (an infrastructure) that enable him to abstract from certain development problems. How these problems are solved by the infrastructure is 'hidden' for the developer.

Figure 4-1 presents the entities involved in a transparency: (i) **the transparency provider**, which is the system that realizes a transparency in terms of services, (ii) **the transparency user**, which is the application that can abstract from the development problem when using the services provided by the transparency provider.

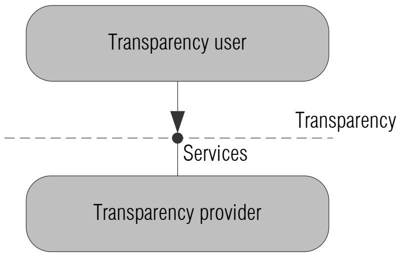


Figure 4-1 Entities involved in a Transparency.

Level of Transparency

Transparencies can be realized by different transparency providers in different ways (e.g. the cobra naming service and the cobra trading service offer both a form of location transparency (OMG 2004)). The services that the transparency provider offers, determines the level of transparency for the transparency user. The **level of transparency** denotes to what extend the development problem is hidden for the transparency user.

An example of transparency providers that provide an increasing level of transparency is visualized in *Figure 4-2*. This figure presents an application that uses different transparency providers that offer an increasing level of transparency. The more the transparency provider hides the development problem for the transparency user (visualized as bigger), the more transparent the application becomes for this development problem. Hence, the implementation of the solution to overcome this problem inside the application decreases (visualized as smaller).

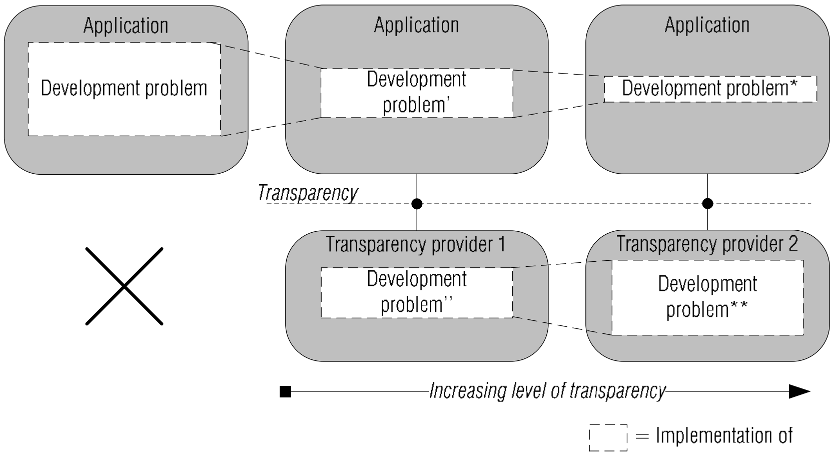


Figure 4-2 Example of systems with different levels of transparency.

A more concrete example, consider an information retrieval application that needs to interact with information sources. Without a transparency provider it has to directly connect to an information source using a fully qualified address (e.g. http://myservice.com:8080/myservice). Three underlying transparency providers can offer three levels of location transparency for the interaction between the application and information sources. The first one offers the lowest level of location transparency by offering a service that requires from the application a simplified address of the information service (e.g. myservice@myservice.com). The transparency provider determines the communication protocol and port number and attaches this to the provided simplified address to connect to the information with the fully qualified address. The second transparency provider offers a service that requires a friendly name (e.g. myInfoService) and transforms this into a fully qualified address by using a pre-determined mapping of friendly names to addresses. Finally, the third transparency provider has the highest level of transparency by offering a service that returns information sources filtered on service capabilities provided by the application (i.e. service discovery capability). The system returns a reference to the most suitable information source to the application.

An aspect that influences the level of transparency a developer of a transparency provider wants to offer is user-control. The developer of the transparency provider is confronted with a trade-off between the amount of hiding his system can perform and the possibility for control it still offers to the application developer. Assumingly on the one hand, the more the development problem is hidden for the application developer, the easier the development process for the application developer becomes. However, on the other hand, the more complex the transparency provider becomes, this may introduce performance overhead, security risks or other unwanted effects. Additionally, the application developer may still require a form of control to fulfil its application specific needs, such that complete hiding of the development problem is unwanted (Kon, Costa et al. 2002).

4.2 Context Binding Transparency

We define a **context binding transparency** as:

Definition 7 Context Binding Transparency

The property of masking the creation and maintenance of context bindings.

From the perspective of the application developer, a realization of a CBT enables him to abstract in his application from how a binding is created, with what context producer this binding is created, and how this binding is maintained in case of appearing and disappearing context producers and fluctuating QoC. *Figure 4-3* shows the realization of the context binding transparency from the developer perspective.

The developer of a context consuming application retrieves context information by expressing context requirements to an infrastructure-based ‘context binder’. The developer can consider this context binder a black-box that returns the required context information to his application. We develop a language to facilitate the application developer to specify their context requirements at an abstract level rather than directly programming these requirements. This language is discussed in detail in Section 4.4.

The binder is responsible for creating a context binding to suitable context producers by matching the context requirements of the consumer with the context offerings of the producers. If during the life-span of the context consumer the availability of the producer or the quality of the provided context information decreases, the binder is responsible for binding a new producer. If no suitable context producer is present this situation is notified to the consumer.

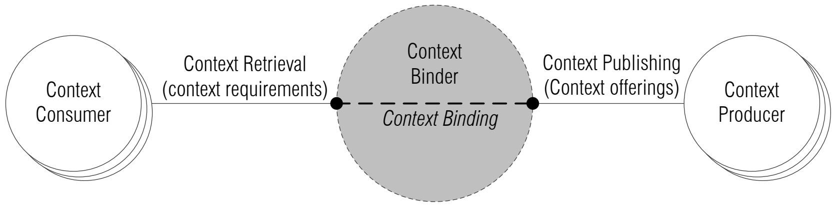
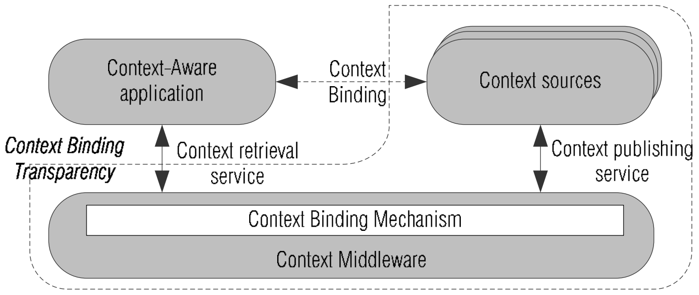

Context Consumer
Context Retrieval (context requirements)
Context Binder
Context Binding
Context Publishing (Context offerings)
Context Producer
Figure 4-3 The realization of a context binding transparency from the perspective of the application developer.

Figure 4-4 depicts from a system perspective the entities involved in a CBT:

- <u>*Context Middleware*</u>: an infrastructure that implements the ‘context binder‘. It acts as the transparency provider. Amongst others, the context middleware consists of a context binding mechanism that realizes a CBT by providing context retrieval and publishing services. These services are discussed in more detail in Section 4.3. The design and implementation of the context middleware and context binding mechanism is discussed in Chapter 5 and 6.
- <u>*Context-aware application and context sources:*</u> users of the context retrieval and publishing services. Transparency users that retrieve context information using the context retrieval service or offer context information by using the context publishing service.


Context-Aware application
Context Binding
Context sources
Context Binding Transparency
Context retrieval service
Context publishing service
Context Binding Mechanism
Context Middleware
Figure 4-4 Entities in a CBT.

In this way, the trend of offloading responsibilities of the context-aware application to the context middleware, as discussed in Section 1.2 is continued. *Figure 4-5* visually presents this trend. First generation context-aware applications contain all function needed to create (and maintain) context binding inside their context logic. This includes context retrieval, context source discovery and selection, and context binding establishment, monitoring and releasing. With second generation context-aware applications the responsibility of context source discovery, and sometimes selection, is offloaded to a context middleware. The application still needs to implement context logic for the context retrieval, context source selection, context binding establishment, monitoring and releasing. With mechanisms that offer a CBT, the responsibility for creating and maintaining context bindings is also offloaded to the context middleware. In this way, third generation context-aware applications are further relieved from development problems not directly related to the development of their application logic.

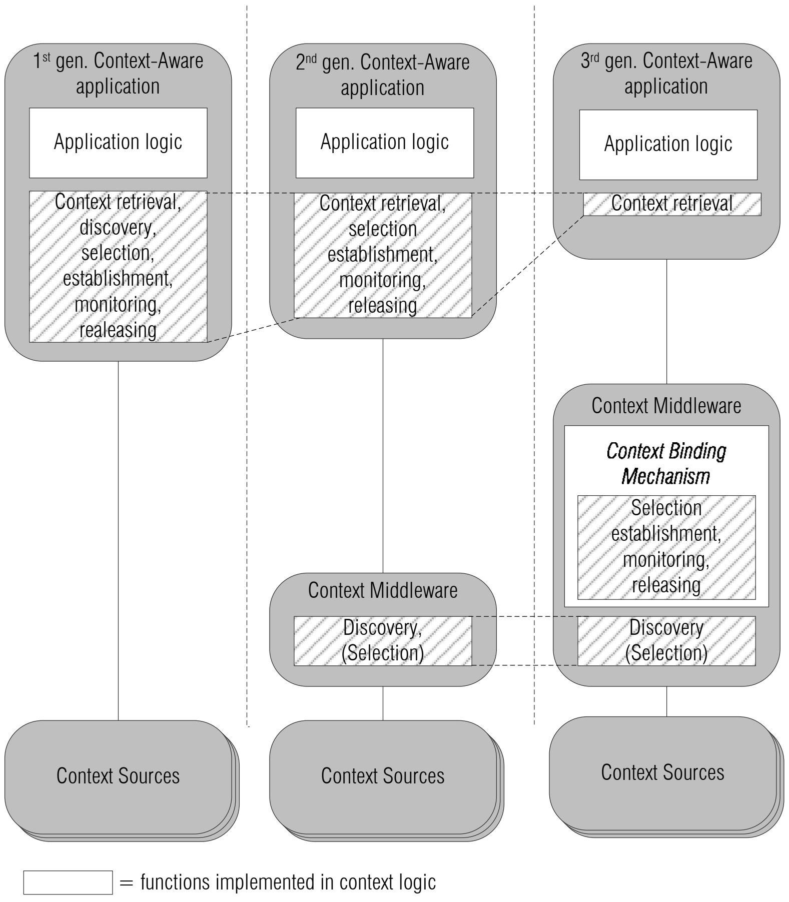

1st gen. Context-Aware application
Application logic
Context retrieval, discovery, selection, establishment, monitoring, realeasing
2nd gen. Context-Aware application
Application logic
Context retrieval, selection establishment, monitoring, releasing
3rd gen. Context-Aware application
Application logic
Context retrieval
Context Middleware
Context Binding Mechanism
Selection establishment, monitoring, releasing
Context Middleware
Discovery, (Selection)
Discovery (Selection)
Context Sources
Context Sources
Context Sources
= functions implemented in context logic

Figure 4-5 Increased offloading of context logic functions towards the context middleware.

Key features of a context binding mechanisms that offers a CBT are:

- *Initialization of a context binding:* based on context requirements specified by the context-aware application, the context binding mechanism resolves a context binding by discovering using available underlying discovery mechanisms, selecting and associating to one or more suitable context producers.
- *Maintenance of a context binding:* based on specified criteria (e.g. costs and QoC) the binding mechanism maintains bindings by:
    - Re-binding at run-time to other suitable context producers when already bound context producers disappear.
    - Re-binding at run-time to other suitable context producers when the QoC that is provided by the already bound context producer may fall below a specified level.
    - Re-binding to context sources with a 'higher' QoC when they become available.
- *Releasing of a context binding:* when the application no longer needs context information, the established bindings are released.

Although a CBT has as goal to hide as much of the creation and maintenance process of context bindings, there are two situations in which the application should be informed on the status of the binding in order to adapt its behaviour to this new situation:

- A context binding fails, because:
    - No suitable context match can be made at application initialization.
    - No suitable new context match can be made after an already bound context source disappears.
    - No suitable new context match can be made when the actual QoC of an already bound context source deteriorates below the required minimal QoC level.
- The actual QoC of an already bound context source fluctuates between application specified QoC levels.

Comparing a CBT with the ODP-RM Transparencies

The ODP-RM offers standards to deal with the communication between distributed objects via heterogeneous communication platforms. In order to interact there has to be a binding between the interfaces of the communicating objects. In comparison, a CBT provider deals with the communication between context consumers and producers. In order to exchange context information there has to be a context binding between these entities. *Table 4-1* compares ODP and CBT concepts.

Table 4-1 Comparing ODP-RM and CBT concepts.

ODP-RM	CBT
Client/Server Object	Context producer / consumer
Object binding	Context binding

ODP-RM offers eight distribution transparencies, as discussed in Section 4.1. When comparing the features of CBT with the transparencies proposed by the ODP-RM, a CBT can be considered as a compound transparency consisting of the features of a combination of basic ODP transparencies. *Table 4-2* relates the ODP-RM transparencies with the features of the CBT.

Table 4-2 Comparing ODP-RM transparencies and a CBT.

ODP-RM Transparency	Featured by CBT	Explanation
Access	√	CBT hides data representations and invocation mechanisms of different context producers, by offering a uniform context retrieval service that binds to suitable context producers, possibly offered by different discovery mechanisms from different administrative domains.
Location	√	CBT hides the physical location of context producers for context consumers by offering services that take over the discovery, selection and binding of suitable context producers, possibly offered by different discovery mechanisms from different administrative domains.
Failure	√	CBT hides disappearing and re-appearing context producers by offering services that perform monitoring of their availability.
Relocation	√	CBT hides the appearing of context producers with higher QoC by offering services that perform monitoring of appearing context producers and, selection and associating of higher QoC producers.
Migration	x	-
Replication	x	-
Persistancy	x	-
Transaction	x	-

Hence, we consider a CBT as a specific form of distribution transparency in which the features of multiple ODP-RM distribution transparencies are combined, for usage in the particular domain of context-aware applications.

4.3 Context Retrieval and Publishing Services

A CBT is exposed to application developers by the context middleware in the form of a context retrieval and publishing service. The application developer has to interact with the context middleware using these services to retrieve or publish context information. In this section, we discuss the abstract interactions offered by these services.

4.3.1 Context Retrieval Service

The context retrieval service facilitates the development and execution of context consuming applications. The context retrieval service has as goal to offer the 'best possible' context to the service user during the existence of a context binding. With the capability to offer the 'best possible' context we mean: (i) when possible, continuity of available context information and (ii) delivery of context information that has the optimal possible quality (costs /QoC).

Table 4-3 describes the abstract service primitives (SP) between the Service User (SU, in our case a context consuming application) and the Service Provider (SPr, in our case the context middleware). Additionally, it describes the type of interaction (i.e. [S]ynchronous and [A]synchronous), and the parameters and possible return parameters.

Table 4-3 Service primitives of the Context retrieval service.

Direction	S/A	SP identifier	Parameters	ReturnParameters
SU → SPr	[A]	createBinding	BindingID, Context_Requirements	-
SU → SPr	[S]	destroyBinding	BindingID	Acknowledgement
SU → SPr	[S]	getContext	BindingID	Context_Information _Sample, QoCLevel
SU → SPr	[A]	subscribetoContext	BindingID	Subscription_ID
SU → SPr	[S]	unsubscribetoContext	Subscription_ID	Acknowledgement
SPr → SU	[A]	notifyContextChange	Subscription_ID, Context_Information_Sample, QoCLevel	-
SPr → SU	[A]	notifyBindingEstablished	BindingID, QoCLevel	-
SPr → SU	[A]	notifyBindingStatus	BindingID, Status, QoCLevel	-

Figure 4-6 presents the relation of the service primitives of the context retrieval service in a time-sequence diagram. The context-aware application starts by expressing its context requirements to the context middleware (*createBinding*) such that the middleware can create a context binding.

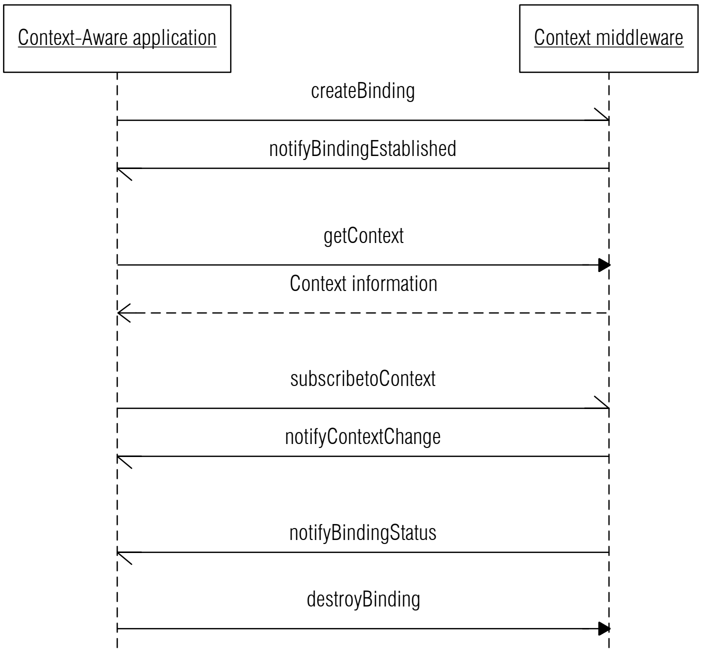


Figure 4-6 Time-sequence diagram of the context retrieval service.

When a suitable binding is established this is notified to the context-aware application (*notifyBindingEstablished*).The binding creation request and notification of established bindings are both asynchronous interactions. This is due to the independent execution of the application and the context middleware. In this way the application can continue its execution when the middleware is initializing a context binding. This corresponds with our notion of a context-aware application, which has a basic context-unaware behaviour that is augmented with context-aware behaviours in case sufficient quality context information is available. For a more elaborate discussion on this aspect of context-aware applications see Section 4.5.

From the moment of notification of an established binding, the context-aware application can retrieve context information, either in a request-response (*getContext*) or subscribe-notify manner (*subscribetoContext/notifyContextChange/unsubscribetoContext*). The middleware can notify the application on changes in the binding status (*notifyBindingStatus*). Types of binding status can be: (i) the binding can be used for retrieval of context information because a producer is bound, (ii) the binding becomes invalid because no producer is bound, (iii) the binding is temporarily invalid because the middleware is trying to rebind to a new producer and (iv) the quality of the offered context information shifts to another QoC level. Finally, a binding can be explicitly destroyed by the context-aware application (*destroyBinding*) or implicitly by the middleware in case the context-aware application undeploys.

In case a binding cannot be created this is notified to the application (*notifyBindingStatus*). In this case, a *notifyBindingEstablished* will not be notified to the application until a binding is established. Consequently, during this period the application is not able to retrieve context information (*getContext, subscribetoContext*).

4.3.2 Context Publishing Service

The context publishing service facilitates the development and execution of context producing applications. It has as goal to advertise the context offerings of the service user to available context discovery mechanisms during the lifespan of the service user.

Table 4-4 describes the abstract service primitives (SP) between the Service User (SU, in our case a context producing application) and the Service Provider (SPr, in our case the context middleware). Additionally, it describes the type of interaction (i.e. [S]ynchronous and [A]synchronous), and the parameters and possible return parameters.

Table 4-4 Service primitives of the Context publishing service.

Direction	S/A	SP identifier	Parameters	ReturnParameters
SU → SPr	[S]	registerContextOffering	Context_offering, Context producer reference	PublishingID
SU → SPr	[S]	deregisterContextOffering	PublishingID	Acknowledgement

The context-aware application can register and deregister its context offerings to the context middleware to enable the middleware to advertise the offerings to available context discovery mechanisms (*registerContextOffering/deregisterContextOffering*). Additionally, the application has to provide a reference to itself such that a context consumer can retrieve the context information offered by the context-aware application.

4.4 The Context Binding Description Language

As part of the CBT, the application developer has to specify its context requirements when using the context retrieval service to retrieve context information (i.e. *createBinding* service primitive, see Section 4.3.1). We propose a language, coined the *Context Binding Description Language* (CBDL), to enable application developers to specify their context requirements at a high level of abstraction rather than in programming code. In this way, the specification of context requirements and the implementation of these requirements in context logic is separated from the development of the actual application logic. For the prototype, as discussed in Chapter 5, we took a pragmatic approach to adapt CBDL to specify context offerings.

However, researching how to specify context offerings with CBDL is out the scope of this thesis.

4.4.1 CBDL Requirement Analysis

Context requirement specifications, expressed in CBDL documents, are used by a context binding mechanism to create and maintain context bindings. Thereby, the context binding mechanism has to create a match between the received context requirements and the context offerings available via underlying context discovery mechanisms. This is visualized in *Figure 4-7*.

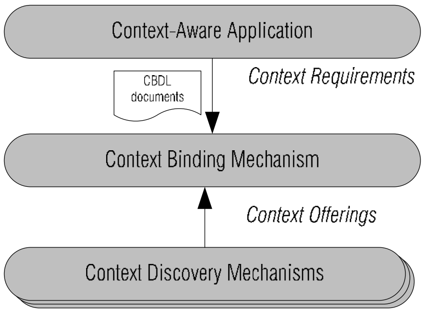

Context-Aware Application
CBDL documents
Context Requirements
Context Binding Mechanism
Context Offerings
Context Discovery Mechanisms

Figure 4-7 Matching the context requirements and context offerings.

To capture the functional requirements of CBDL, we take a two-step approach, addressing both the side of the discovery mechanisms and the application developer. First, we extend the state-of-the-art analysis as presented in Chapter 3, of current context middleware mechanisms to identify common capabilities currently offered. Secondly, we analyse possible use-cases.

In addition, we consider the following general requirement in the design of CBDL:

– Generality: specification of context requirements in CBDL should not be restricted to specific application domains and hence CBDL should be able to be applied to a broad range of context-aware applications.
– Usability: specification of context requirements in CBDL should be 'easy' and should not require a steep learning curve.

Analysis of Capabilities of Current Context Middleware Mechanisms

We consider current context discovery mechanisms because they implement solutions that fulfil context requirements that application developers may have and because our proposed context binding and discovery interoperability mechanism (see Chapter 5 and 6) builds on top of these solutions.

We review current context middleware mechanisms on the following aspects:

– *Interaction mechanism:* What interaction mechanism do the analyzed discovery mechanisms support?
– *Interaction data:* what type of information is expressed in the context discovery request and response?

The result of our analysis is presented in *Table 4-5*. The following common capabilities are provided by current context discovery mechanisms:

– All analyzed mechanisms support the request-response and subscribe-notify interaction mechanism to retrieve context information.
– All mechanisms require information about the type of context information and the entity to which the context relates, to be able to discover context sources.
– The majority of the mechanisms have the notion of quality of context in the request for context information. However, they may apply different QoC parameters.
– Some mechanisms require a form of security token, such as identity information on the entity that is requesting context information, to be able to discover context sources.

Table 4-5 Context requirement analysis result.

	Interaction mechansism		Interaction data				
Frameworks	Req-Resp	Sub-Not	Entity	Type	QoC	Sec. info	Format
Context Toolkit	●	●	●	●	x	●	XML
Solar	●	●	●	●	x	x	n/a
Pace	●	●	●	●	●	x	Context Modelling Language
JCAF	●	●	●	●	x	x	Java objects
CMS	●	●	●	●	●	x	RDF
CMF	●	●	●	●	●	●	RDF
CCS	●	●	●	●	●	●	SQL/PIDF
CDF	●	●	●	●	●	x	RDF/PIDF
Jexci	●	●	●	●	x	●	Negotiable (PIDF/java objects)
Legend: ●, x = 'support', 'not support'							

Analysis of Use-cases

We analysed multiple use-cases to identify additional unfound requirements relevant for future context-aware applications. In Appendix B, we present two which we believe are representative for a broad range of context-aware applications. From these use-cases, we derive the following characteristics of context information and context-aware applications:

– Context information is defined by its context type. For example, location, availability, bandwidth, meeting status.
– Context information is always related to a context entity. For example, patient, doctor, voluntary caregiver, meeting participant.
– Context information can be offered in different context formats. For example, lat/long, xyz, nmea, Boolean.

- Relevancy of context information for applications can depend on different QoC criteria. For example, precision, probability of correctness. See also (Buchholz, Kupper et al. 2003; Sheikh, Wegdam et al. 2007).
- Context information transfer might occur during the whole life-span of the application or during a limited period. For example, during a epileptic seizure.
- Delivery costs involved when using context information might pose criteria for the suitability of context bindings. For example, when retrieving context information the use of a certain communication mechanism or the commercial cost of the context information may differ between context producers.

Overall Conclusions and Identification of CBDL Requirements

Based on the analysis of current discovery mechanisms and use cases, we identify the following requirements for CBDL:

- *Basic context elements:* Context type, entity and format are basic elements needed to describe context requirements.
- *QoC criteria:* Applications have QoC requirements and may react differently when these QoC are not met. Therefore, CBDL should enable application developers to specify quality levels (i.e. minimal and intermediate levels) on the required context information.
- *Costs:* Additionally to QoC, context delivery costs pose criteria on the suitability of a context binding. Application developers should be able to specify in CBDL cost criteria related to QoC criteria. For example, high quality context information might only be relevant for a context-aware application when its costs are not too high. In that case, lower quality context information might be a better choice to use.
- *Binding characteristics:* Transfer of context information can be continuous during the life span of the application or can be limited to a certain period in the life span of the application. Context bindings are therefore not always required. An application developer should be able to specify in CBDL the characteristics of the required binding. This includes re-binding strategy (in case of losing a bound context source) and scope of the discovery. Furthermore, they should be able to specify if re-binding is necessary in case a QoC level cannot be maintained or better quality context sources may appear.
- *Notification:* Although our transparency strives for continuous availability of high quality context information, this might not always be possible. Application developers have to be able to specify in CBDL a notification strategy in case a lost binding cannot be recovered or QoC level cannot be maintained, such that the context-aware application can adapt its behaviour to these situations.

4.4.2 Design of the Context Binding Description Language

We distinguish three types of information in a CBDL document:

- *Context specification*: basic information on what context information the context-aware application requires.
- *Quality criteria*: information on the quality levels which are acceptable for the context-aware application.
- *Binding options:* configuration information required to control the discovery, selection, association, and maintenance process of a context binding.

Figure 4-8 represents the meta-model of the CBDL language, using a UML class diagram.

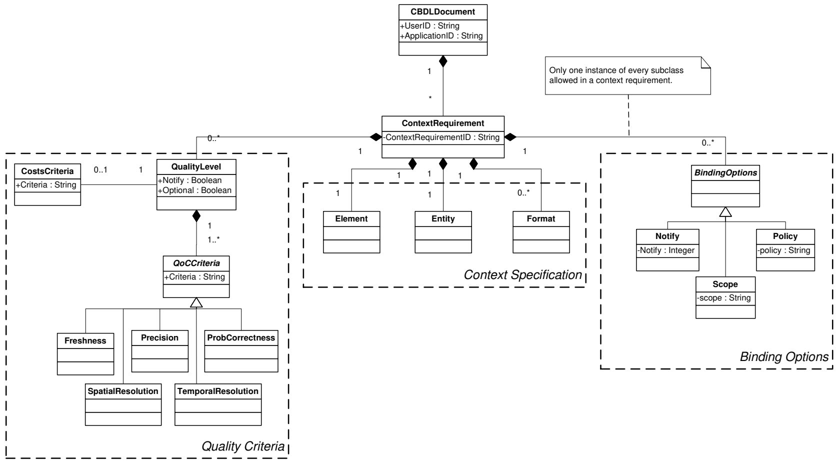


Figure 4-8 CBDL language meta-model.

The root of the CBDL language is the *CBDLDocument* element, which specifies which human user is requesting a context binding (*UserID*) and to which application this binding belongs (*ApplicationID*). This information can be used as security information, for example to retrieve a security token to be able to invoke underlying context discovery mechanisms. However, this is out of the scope of this thesis. Furthermore, a CBDL document (*CBDLDocument*) enables application developers to specify multiple context requirements (*ContextRequirement*). Context requirements have to be uniquely identified by an ID (*ContextRequirementID*). This ID can be used to retrieve a reference to the established binding. This reference can be used to enable the context-aware application to retrieve context information associated to the requirement.

Each context requirement (*ContextRequirement*) consists of mandatory context specification information. This information specifies: (i) a single type of context information that the application requires (*Element*), (ii) the entity to which the required context is related (*Entity*) and (iii) zero or more data formats the required context may have (*Format*).

Optionally, an application developer can specify multiple quality levels (*QualityLevel*). A quality level consists of one or more quality criteria coupled with an optional cost criterion. Multiple quality criteria encapsulated in a quality level are related with an "AND" relation. This means that all criteria have to be fulfilled for the binding to be in this quality level. We distinguish five possible types of QoC criteria based on (Buchholz, Kupper et al. 2003; Sheikh, Wegdam et al. 2007). These are: (i) Precision: "granularity with which context information describes a real world situation", (ii) Freshness: "the time that elapses between the determination of context information and its delivery to a requester", (iii) Temporal Resolution: "the period of time to which a single instance of context information is applicable", (iv) Spatial Resolution: "the precision with which the physical area, to which an instance of context information is applicable, is expressed" and (v) Probability of Correctness: "the probability that an instance of context accurately represents the corresponding real world situation, as assessed by the context source, at the time it was determined" (Sheikh, Wegdam et al. 2007).

Additionally, the application developer may specify if the application needs to be notified when the QoC/Costs of the delivered context information comes into the range of the specified level or falls out of the range (*Notify*, default= true). Furthermore, the application developer specifies if the re-binding mechanism needs to be triggered when the QoC of the delivered context information falls below the specified QoC level (*Optional*, default=false). This transition is notified to the application developer.

Furthermore, an application developer can optionally specify binding options (*BindingOptions*) to control the binding process of the context binding mechanisms. The following options can be specified:

- *Notify*: the application developer can specify the level of notification he wants to receive on the binding process. The following levels are identified:
  - 0: no notifications.
  - 1: notification when a binding is established.
  - 2: notification when a binding is established or broken.
  - 3: notification when a binding is being established, re-established or broken (default).
- *Policy*: the application developer can specify what binding policy should be taken:
  - Static: when a binding is broken, no re-binding is necessary.
  - Dynamic: when a binding is broken re-binding is necessary (default).

- *Scope*: the application developer can specify if context sources should be searched only inside the scope of the local infrastructure (i.e. producers deployed inside the local application container) or also outside the local infrastructure (e.g. in external context discovery mechanisms) (i.e. local/global, default = global).

4.4.3 Implementation of the CBDL Language

We implement the CBDL language using XML, as it is currently the de-facto standard for structured data. Tool support for creating and manipulating XML documents are widely available, which simplifies the creation process of CBDL documents for application developers. Furthermore, XML enables easy validation of the correctness of CBDL documents using XML Schema. The definition of this schema is presented in Appendix B.

Example 4-1 presents an example of a simple partial CBDL document for a healthcare centre application, which is used in the ESS use case explained in Appendix B. This document describes a context requirement with ID 'patient_location' for location information of Patient Tim. This context information should be formatted in lat/long format and should have a precision of 5m or less (i.e. '&lt;').

Example 4-1 XML-based CBDL document.

```
<?xml version="1.0" encoding="UTF-8"?>
<CBDLDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CBDL-schema.xsd" UserID="Healthcarecentre"
ApplicationID="ESS_Healthcarecentre">
    <ContextRequirement BindingID="patient_location">
        <Element>Location</Element>
        <Entity>Patient.Tim</Entity>
        <Format>lat/long</Format>
        <QualityLevel>
            <QoCCriteria>
                <Precision>&lt; 5m</Precision>
            </QoCCriteria>
        </QualityLevel>
    </ContextRequirement>
</CBDLDocument>
```

4.4.4 Related Work

To the best of our knowledge, no other initiatives exist to develop a language for the purpose of specifying context requirements. Although

Hong (Hong 2002) recognizes the need for such a language, coined the context specification language (CSL), this language has not been detailed.

On the other hand, several types of languages have been proposed to facilitate the development of context-aware applications in other ways. For example, Chan et al. (Chan, Wong et al. 2004) define a mathematical rule-based context request language. This language, implemented in XML, enables developers to specify context reasoning rules, using predicate calculus, interpreted by an infrastructure inference engine to retrieve required context information. Yua et al. (Yua, Wang et al. 2002) define a Situation-Aware object interface definition language (SA-IDL), which can be used to generate base classes for a situation-aware objects. Etter et al. (Etter, Dockhorn Costa et al. 2006) describe a rule-based approach to specify context-aware behaviour in the ECA-DL language and to delegate the execution of this behaviour to the infrastructure, using Event-Condition-Action rules. Robinson et al. (Robinson and Henricksen 2007) describe the Context Modelling Language (CML) which can be used to capture context information requirements to be used in the design of context-aware applications. Chen (Chen, Finin et al. 2005) discusses a context ontology (SOUPA) that can be used to exchange context among entities in a uniform manner.

4.4.5 Limitations and Future Work

We acknowledge some limitations of the CBDL language, which we consider future work:

- *Semantics:* In this chapter we specified the syntax of the CBDL language, which application developers can use to specify context requirements. However, for a context binding mechanism to fully be able to match context requirements also the semantics of the document expressed in the CBDL language should be known.
- *Context offerings:* The CBDL language is currently focussed on specifying context requirements of context consuming applications. However, context producing applications could possibly be supported when context offerings could be specified in CBDL. Although in this research we provide a pragmatic solution for specifying context offerings in CBDL (see Chapter 5), more research is needed.
- *Privacy:* Besides being able to specify context requirements, the CBDL language could be extended with elements to specify privacy properties. Especially, this provides opportunities for context producing applications that want to restrict access to their context information.
- *Conflicting QoC levels:* An application developer could specify conflicting QoC levels for a context requirement. More research is needed how to

detect conflicting QoC levels and notify the application developer of these conflicts.
- *QoC level construction:* The current version of the CBDL language assumes that the QoC criteria encapsulated in a quality level are related with an 'AND' relation. However, also other types or relations could be of interest.

4.5 Discussion

In this section, we discuss the development process of context-aware applications with a CBT. We start with a general reflection on the development process and the influence of availability and QoC on this process. Additionally, we present guidelines for developing a context-aware application with our implementation of the CBT.

4.5.1 Developing Context-Aware Applications with a CBT

As discussed in Chapter 2, a context-aware application consists of (i) application logic and (ii) context logic. Application logic is behaviour which adapts based on available context information. The required context information is acquired by the context logic. A context middleware that offers a CBT can facilitate the development of a context-aware application by reducing the complexity of the context logic. Such a middleware takes over the responsibility of the creation and maintenance of context bindings. Hence, such a middleware enables application developers to spend less effort in creating context logic and focus on the development of the application logic. Furthermore, when bindings become unavailable the middleware tries to re-establish the required context binding.

Although the context binding middleware makes an effort to maintain context bindings, it is inevitable that context bindings sometimes fail. For example, due to unavailability of context sources or unavailability of context sources that can provide context with the required QoC. Hence, we claim that also availability of context bindings and QoC should be parameters that influence the development of the application logic. Application developers should incorporate availability of context bindings and QoC levels as an integral part in their development process, to:
- Be aware in the design process of context-aware applications of different situation that can occur in a realistic operational environment;
- Create more robust context-aware applications.

We distinguish the following characteristics of context-aware applications:

- A context-aware application should be able to function without context information. Hence it has default behaviour, which is context-independent and fulfils the basic user need.
- Additional to the default behaviour, a context-aware application has context-aware behaviour that enhances the quality of the application.
- A context-aware application should be able to react on availability of context information and fluctuating quality of its information.

When we assume that the availability of context bindings and QoC levels corresponds to distinct context-aware behaviours, the application logic can be seen as a composition of a default behaviour augmented with multiple context-aware behaviours that execute based on the availability of context bindings and QoC levels.

Figure 4-9 presents an abstract example of a context-aware application showing these aspects. The context logic contains behaviour to retrieve context information from the context middleware. The application logic consists of default, context-aware (CA) and coordinating behaviour. Coordinating behaviour is responsible for:

- Receiving binding status notifications of the context middleware;
- Composing the applications default behaviour with the context-aware behaviours based on the binding status. This includes selecting and enabling/disabling context-aware behaviours.

Consider for example an application that can display the location of the user and the route to available buddies. This application requires context information on the location of the user, location of the buddies and availability of the buddies. The default behaviour of the context-aware application is to show a map with the location of the user. When only context information on the availability of the buddies is available (so no location information is available), the application shows the buddy but draws them outside the map. When also the location of buddies is available the application can draw the buddies on the map with an availability icon and draw routes from the user to that buddy. When the location of buddies is available but the availability context information is not available, the application can point the buddies on the map with an ‘availability unknown’ icon and does not draw routes. For a more elaborated example, we refer to the case study in Chapter 8.

A possible option to facilitate the development of coordinator behaviour is to deploy a Event-Condition-Action rule engine like proposed in (Dockhorn Costa 2007). This would enable an application developer to specify the composition of behaviours in a rule language (i.e. in terms of conditions on binding status events). The ECA-enabled coordinator parses this document and based on incoming status events from the context middleware, can trigger composition of the default behaviour with specified context-aware behaviours. Another option to implement coordinator behaviour is by applying the state design pattern (Gamma, Helm et al. 1995). In this approach, the developer identifies the states in which his application can be and the transitions between these states. For example in case of the buddy location application, there is a state 'available location and availability information' which leads to a state 'available location information' when availability information becomes unavailable. Being in a state corresponds with a certain combination of default and CA behaviour.

Further research on how to develop context-aware applications is needed. The realization of coordinator functionality is out of the scope of this thesis. We consider this a promising extension of the work presented in this thesis.

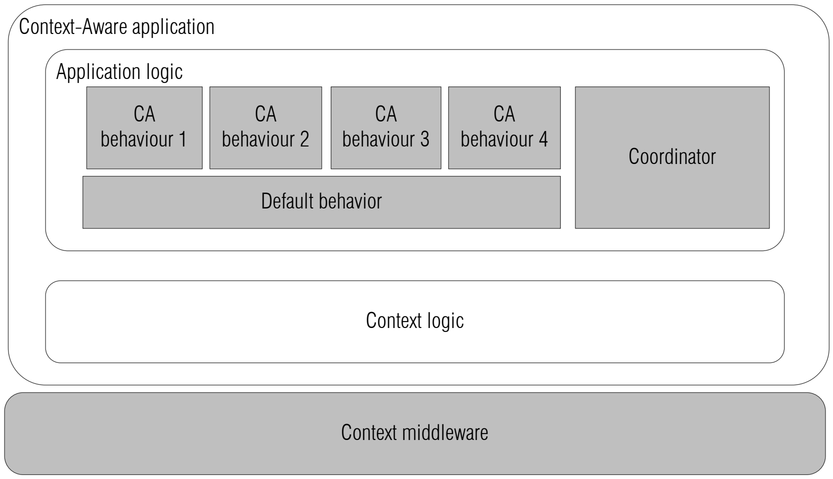


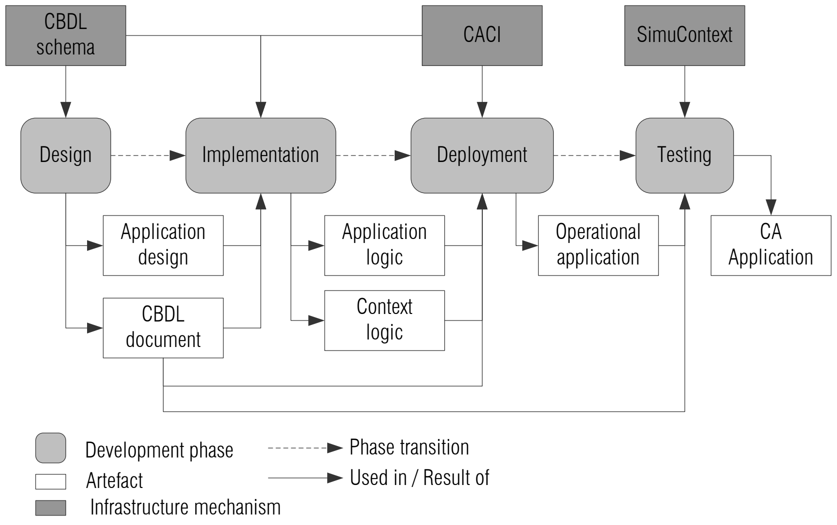
Figure 4-9 Internal perspective of application and context logic

4.5.2 Guidelines for the Development of Context-Aware Applications based on a CBT

In the remainder of this thesis, we discuss our realization of a context middleware that offers a CBT. In this section, we present guidelines for the development of a context-aware application with our context middleware.

Figure 4-10 presents the development trajectory of a context-aware application using our proposed context middleware mechanisms, based on the common waterfall or linear development process model (Pressman 2000). In Chapter 8, we present a case study following this development trajectory.

Figure 4-10
Development trajectory of a context-aware application using our realization of a CBT.



Successively, we present a checklist consisting of steps executed in the phases of the proposed development trajectory. Thereby indicating, (i) which infrastructure mechanisms (dark grey rectangles) are used in the development phases (light grey rounded rectangle) and, (ii) which CBDL language elements result from every step (i.e. between '[]') and, (ii) which artefact results from a development phase.

Design:

The design step results in an application design and a CBDL document specifying context requirements. This step includes:

1. Based on application requirements, design the context-aware application to be implemented:
    a. Determine which types of context information is required for this behaviour to execute → [*ContextRequirement*], *[ContextRequirementID],* [*element]*
    b. Design the application behaviour in case the context information is available but also the behaviour when the context information becomes unavailable.
    c. Determine the required binding behaviour → [*binding options*].
2. For every context type determine from which entities the required context information should be retrieved → [*entity*].
3. For every context type determine the formats the context information should have → [*format*].
4. For every context type determine the quality of context levels which influence the application behaviour:

   a. Optionally determine the minimal required QoC → [*QoCLevel*].
   b. Determine intermediate QoC levels on which the behaviour of the context-aware application changes → [*QoCLevel*].
   c. Determine the application behaviour in case that QoC shifts from one QoC level to another but also when the QoC falls below the minimal required QoC.
   d. Determine how the application should react on unavailable QoC.
5. Based on the CBDL schema, transform the designed context requirements into a CBDL document.

Implementation

The implementation step results in one or more application components consisting of the context and application logic. This step includes:

6. Implement the context logic:
   a. Use OSGi interfaces and the ContextRequirementID's specified in the CBDL document to retrieve handles on context bindings to retrieve context information.
   b. Implement a notification callback where the context middleware can provide information on the availability and current QoC of the context information available provided by context bindings.
7. Implement the application logic.
   a. Connect the application logic with the context logic.
   b. Implement a strategy for situations in which context becomes unavailable (see 1b)
   c. Implement a strategy for shifts in QoC levels (see 4c)

Deployment:

The implementation step results in a packaged and operational context-aware application. This step includes:

8. Package the complete application together with the CBDL document.
9. Install and run the CACI container
10. Deploy and run the packaged application inside the CACI container.

Testing:

11. Use SimuContext (see Chapter 6) and the created CBDL document to generate simulated context sources.
12. Test the deployed CA application by deploying the simulated context sources and tweaking its properties.

Chapter 5

Context Binding in CACI

This chapter presents the Context-Aware Component Infrastructure (CACI). This infrastructure consists of the context binding and context discovery interoperability mechanism. In this chapter, we describe the overall design of CACI and give a detailed description of the design and prototype implementation of the context binding mechanism. The next chapter discusses the context discovery interoperability mechanism. Parts of this chapter are published in (Broens, Halteren et al. 2006; Broens, Sinderen et al. 2007).

This chapter is structured as follows: Section 5.1 presents the overall design of CACI. Section 5.2 presents the design of the context binding mechanism. Section 5.3 presents the prototype implementation of the context binding mechanism. Section 5.4 discusses related work. Finally, Section 5.5 discusses limitations of the context binding mechanism and indicates future work.

5.1 Overall Design of CACI

This section positions CACI and gives an overview of the mechanisms of which CACI consists, by decomposing the models introduced in Chapter 2. Additionally, it presents some development alternatives and motivates the choices we made for the development of CACI.

Modelling and Positioning CACI

Figure 5-1, presents a high-level model of a context-aware application and a context middleware. Context-aware applications and context sources use context retrieval and publishing services to retrieve and publish context information. These services are provided by a context middleware. We distinguish a context management entity that realizes these services. Context middleware might also provide other services like, services to support context reasoning and privacy enforcement. These other services are, for clarity, omitted in the remainder of this section.

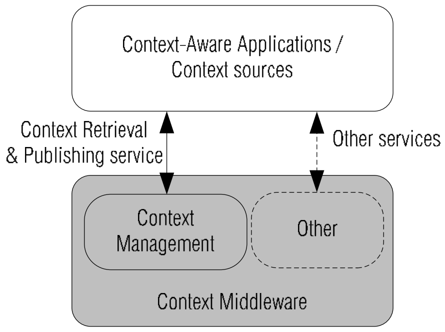

Context-Aware Applications / Context sources
Context Retrieval & Publishing service
Other services
Context Management
Other
Context Middleware

Figure 5-1 High-level overview of a context-aware application and context middleware.

Figure 5-2 presents a decomposition of the context management entity. The context management entity consists of CACI and multiple context discovery mechanisms. CACI realizes the context retrieval and publishing service. CACI is responsible for creating and maintaining context bindings on behalf of context-aware applications. Amongst others, this includes discovering suitable context sources. For this purpose, CACI uses discovery services offered by available context discovery mechanisms. Context discovery mechanisms are used to register offerings of context sources and discover registered context sources. Hence, CACI assumes the availability of one or more context discovery mechanisms as an underlying support mechanism.

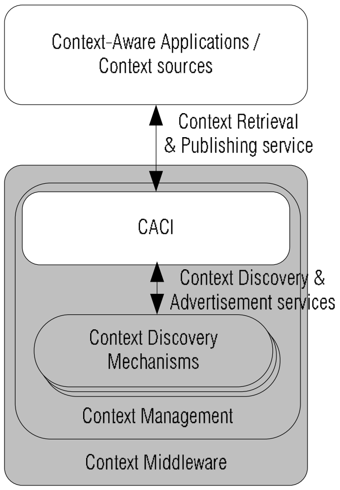

Context-Aware Applications / Context sources
Context Retrieval & Publishing service
CACI
Context Discovery & Advertisement services
Context Discovery Mechanisms
Context Management
Context Middleware

Figure 5-2 Decomposition of the context management entity.

Figure 5-3 presents a decomposition of CACI. CACI consists of two mechanisms: (i) the context binding mechanism and (ii) the context discovery interoperability mechanism. The context binding mechanism realizes the context retrieval and publishing service and is responsible for creating and maintaining context bindings based on the context requirements and offerings of context-aware applications and context sources. In the remainder of this chapter, we discuss the design and implementation of the context binding mechanism in detail.

The context binding mechanism requires discovery services to find suitable context sources. However, the environment of an application may consist of multiple heterogeneous context discovery mechanisms. Furthermore, a mobile application can travel through multiple environments and can hence encounter multiple dynamically (dis)appearing context discovery mechanisms during its lifespan.

The context discovery interoperability mechanism is responsible for supporting the context binding mechanism in finding suitable context sources using these dynamically available and heterogeneous context discovery mechanisms. The context discovery interoperability mechanism offers a uniform context discovery and advertisement service that takes care of finding and registering context sources from and to encountered context discovery mechanisms. In Chapter 6, we discuss the design and implementation of the context discovery interoperability mechanism in detail.

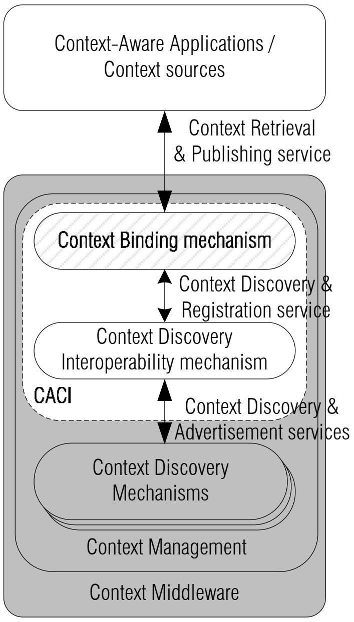

Context-Aware Applications / Context sources
Context Retrieval & Publishing service
Context Binding mechanism
Context Discovery & Registration service
Context Discovery Interoperability mechanism
CACI
Context Discovery & Advertisement services
Context Discovery Mechanisms
Context Management
Context Middleware

Figure 5-3 Decomposition of the CACI block.

Development Alternatives

The goal of CACI is to offer mechanisms that support the development of mobile context-aware applications. Support mechanisms can be classified along several dimensions. We distinguish the following dimensions:

– *Local versus remote support mechanisms:* Local support mechanisms execute on the host that is also executing the application that the mechanism is supporting. Instead, remote support mechanisms execute on a remote host reachable from the application host. Here, we see a trade-off between available processing capacity and introduced communication overhead. In case of a remote support mechanism, a remote host performs (part of) the processing required to support the application. This offloads the local host of (part of) this processing responsibility. However, the request for processing and the results of the processing needs to be transferred via a communication platform from and to the application and the remote support mechanism. This introduces communication overhead. This also implies that the application has to be interoperable with the remote support mechanism by conforming to the used communication platform (i.e. communication protocol, interfaces). In case of a local support mechanism, the processing is performed locally and no remote communication is required. However, the capacity of the local host, for example processing power and battery power, should be sufficient to perform this processing.

– *Private versus shared support mechanisms:* The services of a private support mechanism can only be used by a single application. Instead, the services of a shared support mechanism can be used by multiple applications. Hence, a private support mechanism is tightly coupled to a specific application and does not expose its services to other applications. Sharing functionality can reduce redundancy and costs. However, it may also introduce security risks.

– *Specific versus generic support mechanisms*: Specific support mechanisms are developed for a specific type of application. Instead, generic support mechanisms are developed for a range of different applications. Generic support mechanisms provide solutions for recurring problems faced by a range of applications. Middleware-based mechanisms, as discussed in Chapter 2, are typically generic support mechanisms.

We observe that some combinations of these dimensions are more commonly applied in support mechanisms than others. For example, remote, shared and generic support mechanisms are common for current context middleware that support context discovery. For example, Pace, CCS, CDF, Context toolkit, JCAF, CMS and CDF (see Chapter 3) offer generic, shared and remote support mechanisms. However, other combinations are also feasible.

Motivating the Support Mechanism offered by CACI

CACI offers a generic, shared and local support mechanism. A major difference between current context middleware and CACI is that the functions offered by CACI's support mechanisms are executed locally. Both the context binding mechanism and the context discovery interoperability mechanism are co-located with the application and execute on the application host. These mechanisms interact with remote counter parts, such as context discovery mechanisms for context producer discovery and remote context sources for actual context information transfer. *Figure 5-4* shows the execution location of the involved entities in a CACI-enabled system.

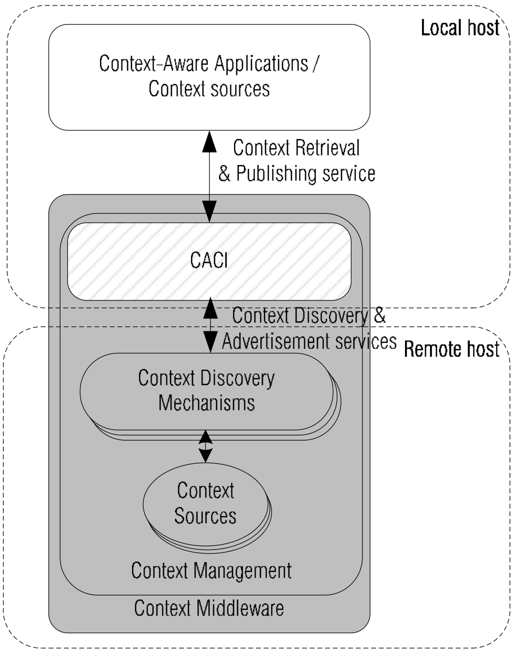

Local host
Context-Aware Applications / Context sources
Context Retrieval & Publishing service
CACI
Context Discovery & Advertisement services
Remote host
Context Discovery Mechanisms
Context Sources
Context Management
Context Middleware

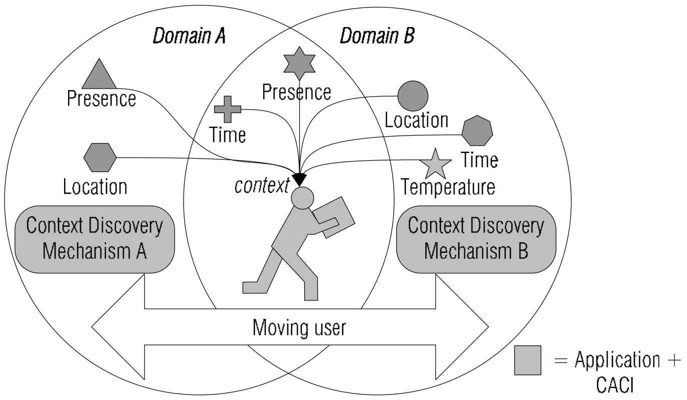
Figure 5-4 Support mechanisms offered by CACI.

Offering local support mechanisms has several advantages:

– Application specific support: CACI is tightly coupled with the application it has to support. Consequently, CACI can more easily access local application knowledge, like (un)deployment of application parts which offers the opportunity for enhanced context binding functionality such as automated creation and destruction of context bindings. Furthermore, CACI 'travels' with the mobile application from domain to domain (see *Figure 5-5*). Therefore, it has (or can get) knowledge of the domain it is in, like availability of domain specific context discovery mechanisms.

- *Configurability:* Remote support mechanisms are commonly shared and generic for a multitude of applications. Individual configuration of these mechanisms for specific applications is therefore complex and mostly not feasible. A feature of CACI is that it can be configured locally by application/infrastructure developers. For example, developers could create and configure application specific adapters for context producer selection and rebinding algorithms.
- *Robustness:* As CACI 'travels' with the application, it can cope with loss of infrastructure. It could for instance use ad-hoc context discovery mechanisms to discover context sources or when entering another domain it could transparently switch to the available context discovery mechanism offered by that infrastructure.
- *Performance*: Context-aware applications are very promising to deploy on mobile devices, as they operate in constant changing environments. Therefore, we target these mobile devices as the main underlying resource platform for CACI. Adding support functions to the application host increases the resource requirements on the client's hardware resources. For example, CPU, memory and network capacity. Furthermore, mobile devices are resource limited and therefore the feasibility, in terms of performance, of locally executing CACI has to be addressed (see Section 5.3). Nevertheless, we envision that the number of required context bindings for a context-aware application is limited. Additionally, with the increasing capabilities of mobile devices, we expect that creating and maintaining a limited amount of bindings is feasible.

Figure 5-5 CACI 'travels' with the mobile device of the user.



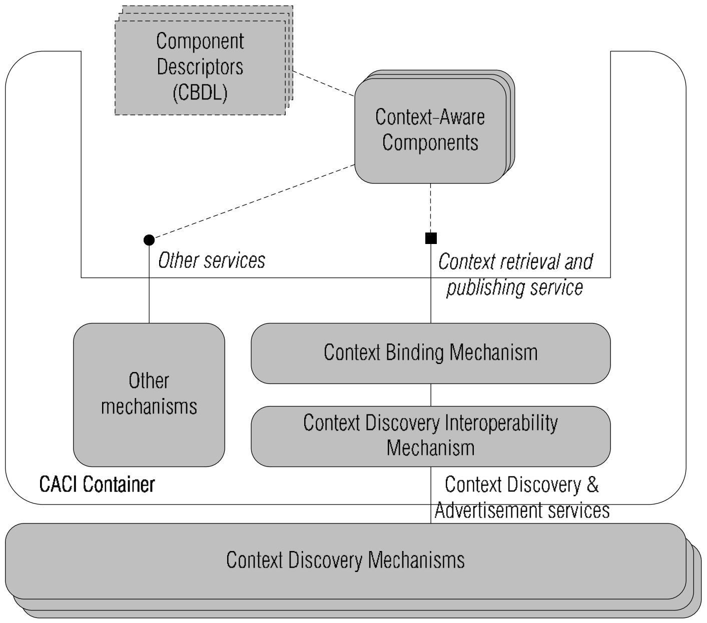
Overall Design

CACI adopts a component-based development approach for context-aware applications. We refer to Chapter 3 for more background information on component-based middleware. Hence, CACI considers a context-aware application as a composition of *context-aware components* (see *Figure* 5-6). These components are deployed in a component environment called a *container*. Generally, a container offers support mechanisms to components such as component life-cycle management and inter-component communication. Specifically, the *CACI container* contains the *context binding mechanism* and the *context discovery interoperability mechanism*. The context binding mechanism offers the *context retrieval and publishing service* to context-aware components. These services have been specified in Section 4.3.

The capabilities of components are described in *component descriptors.* Amongst others, a component descriptor describes the required and offered interfaces of a component. The descriptions are used at deploy-time of the component to configure the container. For example when considering a component A, the configuration of the container may include finding and connecting to a component B that can offer the required interface specified in the component descriptor of component A.

In our solution, we also use the component descriptor to describe the context requirements of a component. These descriptions are expressed in the CBDL language (see Section 4.4). The context binding mechanism handles on deploy-time the context requirements specified in the component descriptor. Actions that the context binding mechanism takes are the initialization and maintenance of context bindings. These configuration actions are discussed in detail in the next section. After deployment of the component, the context-aware component can exchange context information via the context retrieval and publishing service.

Figure 5-6 Overview of the CACI design.


Component Descriptors (CBDL)
Context-Aware Components
Other services
Context retrieval and publishing service
Other mechanisms
Context Binding Mechanism
Context Discovery Interoperability Mechanism
CACI Container
Context Discovery & Advertisement services
Context Discovery Mechanisms

5.2 Design of the Context Binding Mechanism

In this section, we present the design of the context binding mechanism by describing the design of the context retrieval and publishing services. As part of the design of the context retrieval service, we discuss a possible rebinding algorithm. Finally, we combine the two designs in an integrated design of the context binding mechanism.

5.2.1 Context retrieval service

This section starts with a high-level overview of the part of the context binding mechanism that realizes the context retrieval service. Consecutively, we present the functional decomposition and behaviour of the context binding mechanism that realizes this service.

High-level Overview

Figure 5-7 presents a further decomposition of a context consuming application and the context binding mechanism. A context-aware application with its specific application logic, formulates context requirements (including a unique requirement identifier). These are sent to the binding mechanisms in one or more binding creation requests to the context retrieval service by the context logic. Such requests can consist of three parts: (i) context information specification, (ii) optional QoC criteria and (iii) optional binding options.

When a binding creation request is invoked by the context-aware application, the context binding mechanism creates a context producer proxy (CP') which acts as the single point of access to context information used by the application.

The context binding mechanism discovers suitable context sources (CS) by using the discovery services of available context discovery mechanisms. These mechanisms are used to store the offerings of context sources. Additionally, a reference to the context source is stored, which can be used to access its context information (CP*). The context binding mechanism selects a suitable context source from the discovery results based on the offerings of the source and the requirements of application. The CP' is bound to the selected CP*. From this moment the proxy (CP') can deliver context to the context-aware application. The context-aware application can retrieve the proxy from the context retrieval service using its specified requirement ID.

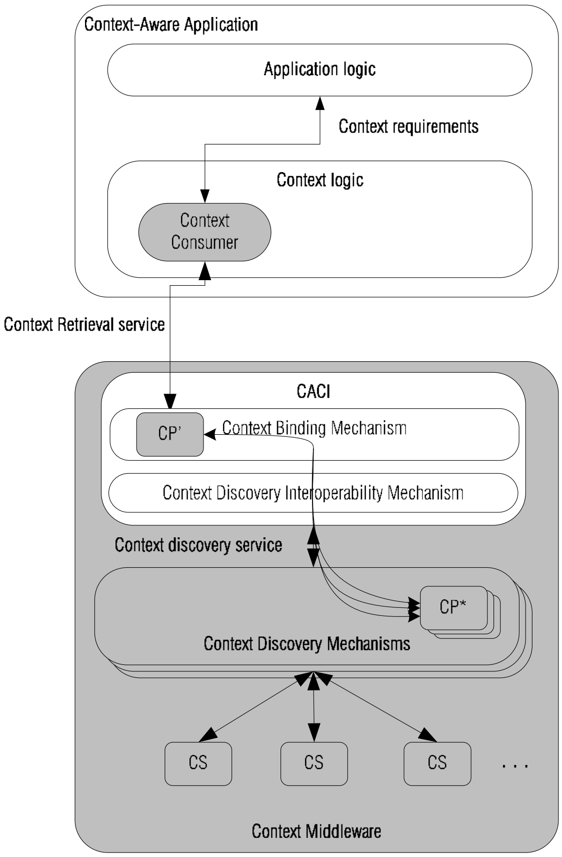

Context-Aware Application
Application logic
Context requirements
Context logic
Context Consumer
Context Retrieval service
CACI
CP'
Context Binding Mechanism
Context Discovery Interoperability Mechanism
Context discovery service
CP*
Context Discovery Mechanisms
CS
CS
CS
. . .
Context Middleware

Figure 5-7
Decomposition of the context binding mechanism.

Decisions to rebind in case of (dis)appearing context sources and degrading QoC are dealt with by a rebinding algorithm. The rebinding algorithm can react on two types of situations: (i) explicit registration or deregistration of context source received from the context discovery mechanism and (ii) context retrieval errors intercepted by the proxy. A possible rebinding algorithm is discussed in the next section.

The context binding mechanism applies the proxy pattern (Buschmann, Meunier et al. 1996) for the internal representation of bound context sources (see *Figure 5-8*). This proxy shields the context consuming application from the dynamic aspects of a bound context source. The use of a context producer proxy has the following advantages:

– Transparency: by shielding the application for direct interaction with context sources, the application can be made unaware of the dynamic availability and offered QoC of the bound context source. The proxy acts as a middleman that can instruct the context binding mechanism to initiate a re-binding process in case of unavailability of the bound context source or degrading QoC.
– *Optimization:* the proxy enables one to have different context acquisition strategies between the context consuming application and the proxy, and the proxy and the context source. This acquisition strategy can be optimized dynamically based on the context producer or underlying network technology. For example, by caching or buffering context information samples. This direction is not further explored in this thesis, we consider this future work.

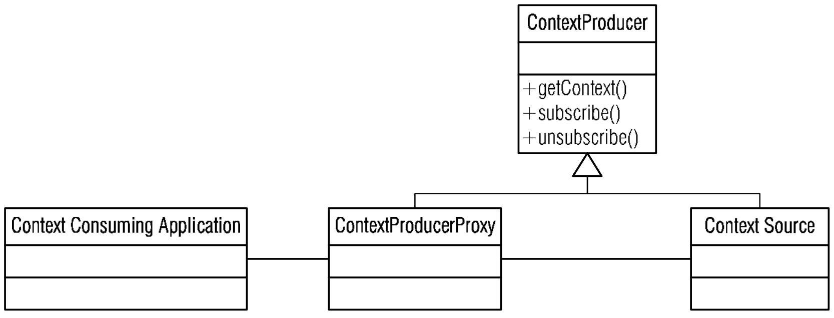


Figure 5-8 Appliance of the proxy pattern for representing a bound context source.

Functional Decomposition

Figure 5-9 presents the functional decomposition of the context binding mechanism. In the figure, we define the functions required to realize the context retrieval service. These functions are:

– Deployer & Parser: The deployer is responsible for intercepting deploying components. The deployer determines if the context-aware component is a CACI-enabled component and instructs the parser to parse the

CBDL description. A CBDL enabled component is a component that has a CBDL component. The parser distils the binding requests from the CBDL document.

- *Binder:* The binder has a coordinating role in the context binding mechanism. It coordinates the process of creating and maintaining context bindings for a deploying context consuming component. The created context producer proxy might notify the binder of context retrieval errors. This triggers the rebinding algorithm.
- *Discovery manager:* The discovery manager is responsible for issuing discovery requests to the available context discovery mechanisms via the context discovery interoperability mechanism. Furthermore, it collects all the discovery results.
- *Selector:* The selector is responsible for selecting a suitable context producer for the context binding. This selection is based on the offerings of a set of producers, which are the result of the discovery phase, and the distilled context requirements of the deployed component.
- *Monitor:* The monitor is responsible for monitoring the availability of context producers. It receives notifications of changes in the availability of context producers from the discovery mechanisms. Two types of notifications can be received (i) producer de-register events and (ii) new-producer register events. This triggers the rebinding algorithm.
- *Decider:* The decider is responsible for executing the rebinding algorithm.
- *Proxy manager:* The proxy manager is responsible for the encapsulation of the selected context producer into a context producer proxy and making this proxy available to the deployed context consuming component. The proxy manager exposes the context retrieval service to the context-aware component.
- *CACI_database:* The CACI database stores the administration of the context binding mechanism. This includes, amongst others, which CACI-enabled components are installed and which context requirements they have.
- *GUI:* The GUI is a graphical representation of the CACI database that is used for testing and debugging purposes.

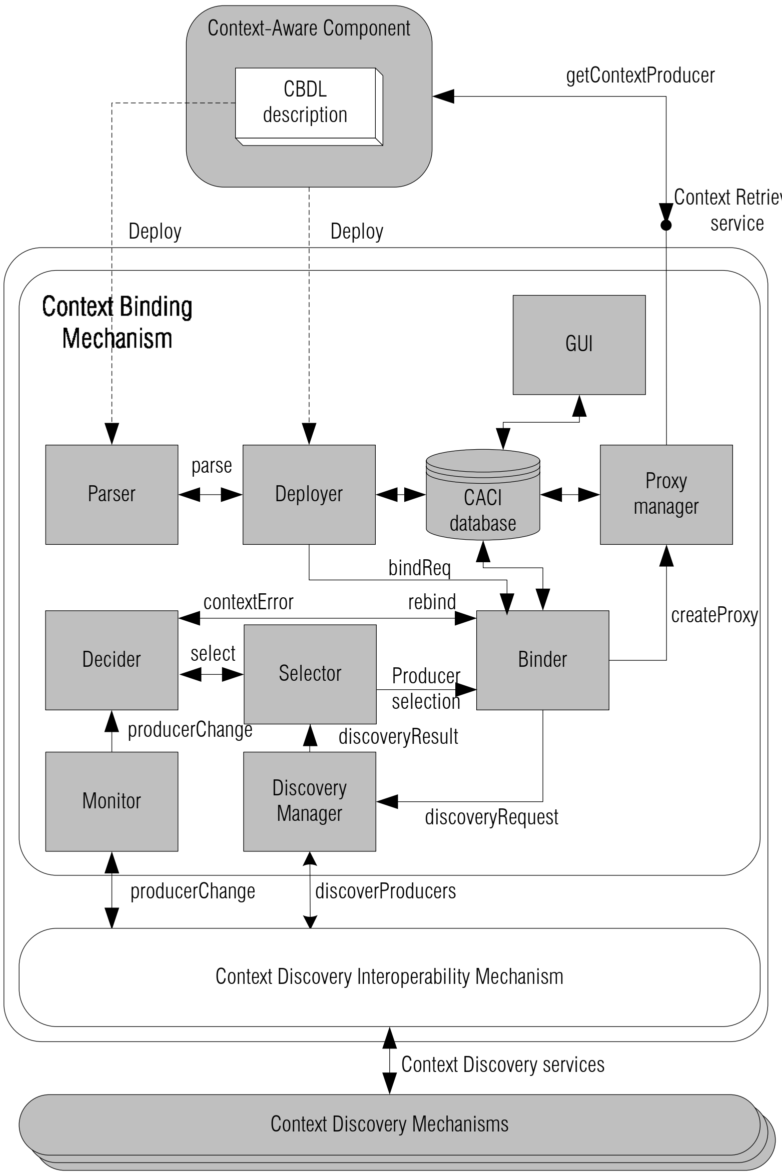


Figure 5-9 Funtional decomposition of the context binding mechanism.

Figure 5-10 shows a high-level diagram of the states in which a context binding can be. When a component is deployed, the binding initiates in the unbound state. A binding is established by creating a proxy and connecting this proxy to a physical context source. When this process fails, the component is either not deployed, or it is deployed but the unbound state is notified to the component. When there is successful binding the binding makes a transition to the bound state. When the binding fails, in case of disappearing of currently bound context sources, appearing context sources or degrading QoC, a re-binding process is initiated. The binding makes a transition to the re-binding state. In case of a disappearing bound context source, the rebinding process can fail. If no suitable context sources are available. The binding returns to the unbound state and a notification is send to the application.

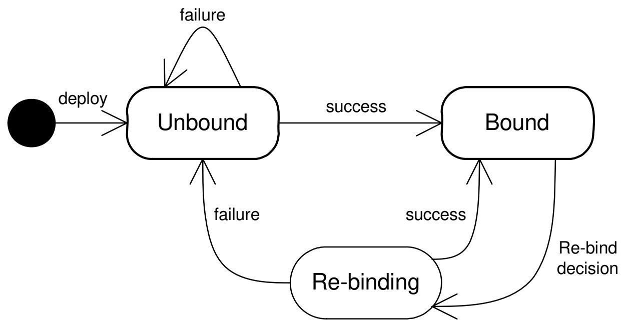


Figure 5-10 Binding states.

Internal Behaviour

Figure 5-11 presents an activity diagram that represents the internal behaviour of the context binding mechanism. After initializing, the context binding mechanism waits for changes in the container. These are notified to the mechanism by 'deployevents' which can be of two types: (i) deployment of a new component and (ii) un-deployment of an already deployed component. In both cases, the mechanism checks if a CBDL document is available, which means that the component is CACI-enabled. If this is the case CACI continues to handle the component, else the event is ignored and CACI stops handling the component. In case of un-deployment of a CACI-enabled component, the context binding mechanism cleans-up the administration and releases the established bindings. In case of a deploying CACI enabled component, the process of initializing the required context bindings is started.

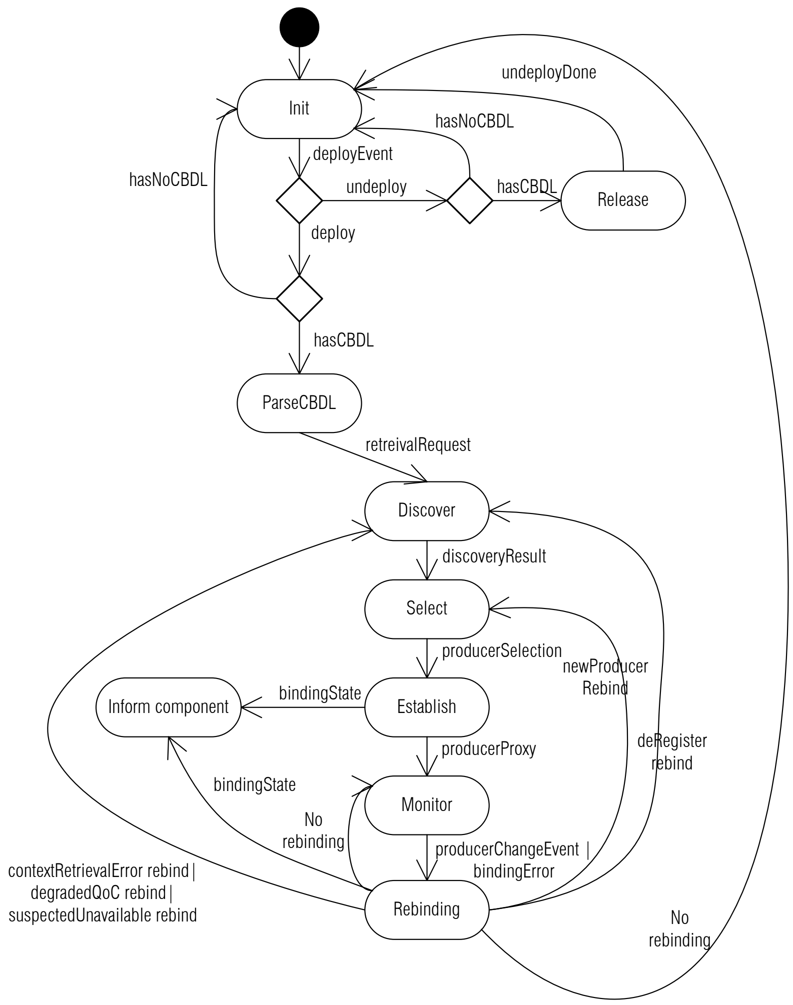
This process starts by parsing the CBDL and extracting context binding requests. For every context retrieval request, a discovery session is initiated in which discovery requests are issued (using the information encapsulated in the context retrieval request) to the available discovery mechanisms. From the results, a suitable context producer is selected and a binding is created by connecting a proxy to this producer. The deploying component is informed of the status of the binding.

Subsequently, the binding is monitored for the availability of the bound context source and the quality of the context information it offers. This consists of monitoring the binding for:

- *Producer Change Events:* These events are explicitly received from the underlying context discovery mechanisms. There are two types of possible events: (i) de-registration of the bound context source and (ii) a new context source becomes available.
- *Binding Errors:* These errors are determined by the context producer proxy. There are three types of possible errors: (i) a request for context information results in an error, (ii) the QoC of the bound producer degrades below the minimal required QoC level and (iii) in case of a subscription, no context information is received for a certain period of time, which raises the suspicion that the bound context source is unavailable.

These situations may lead to a decision to start a rebinding process, visualized by the rebinding activity. We discuss a possible rebinding algorithm in the next section.

Figure 5-11 Activity diagram of the behaviour of the context binding mechanism.



5.2.2 Rebinding Algorithm

In this section, we zoom into the rebinding activity. The proposed rebinding algorithm is merely an example of how to deal with binding errors and change event. More research is needed to define optimal rebinding algorithms. *Figure 5-12* shows an overview of the rebinding activity.

The first step in the rebinding algorithm is to check if the developer has specified in the CBDL component descriptor the option for dynamic rebinding. This is specified in the 'Policy' field of the CBDL document. When the component requires dynamic rebinding (i.e. 'Policy =

Dynamic') a rebinding decision has to be made. We zoom into the rebinding decision activity for the different producer change events and binding errors in the following sections.

When the component does not require dynamic rebinding (i.e. 'Policy = Static') the rebinding process is aborted and the component is informed of a failed binding. Additionally, in the case the rebinding process is triggered by a 'newProducer event' the binding mechanism continues monitoring, as the current binding is still valid. Additionally, when the rebinding process is triggered by the "suspectedUnavailability event', first the suspected unavailability of the bound producer is checked. We discuss how this can be done in more detail when discussing the 'suspected producer unavailability algorithm' in the next sections.

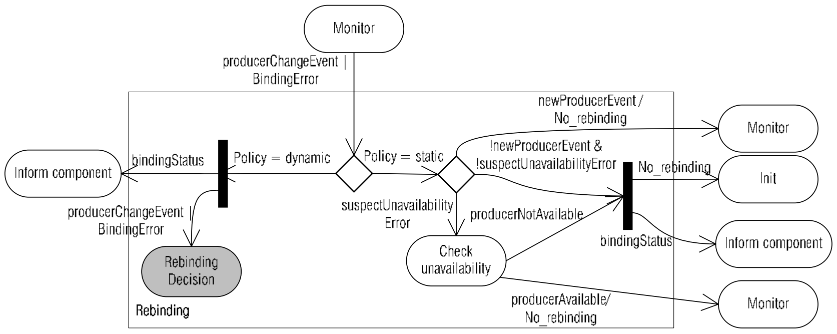

Monitor
producerChangeEvent | BindingError
newProducerEvent/ No_rebinding
Monitor
Inform component
bindingStatus
Policy = dynamic
Policy = static
!newProducerEvent & !suspectUnavailabilityError
No_rebinding
Init
producerChangeEvent | BindingError
suspectUnavailability Error
producerNotAvailable
bindingStatus
Inform component
Rebinding Decision
Check unavailability
Rebinding
producerAvailable/ No_rebinding
Monitor

Figure 5-12 Overview of the Rebinding activity

In the following sections, we successively zoom into the rebinding decision activity for the various producer change events and binding errors.

New Context Producer Algorithm

Figure 5-13 shows the rebinding decision activity when considering an incoming 'newProducerEvent'.

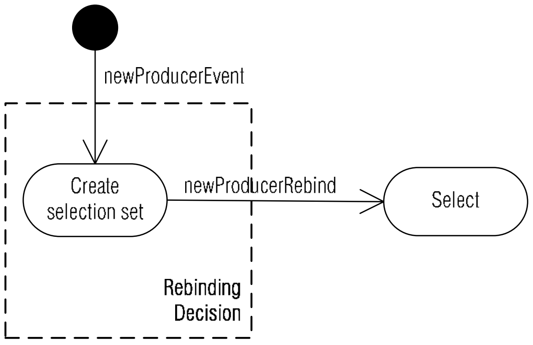


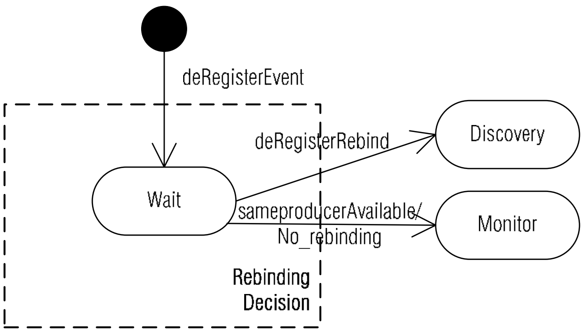
Figure 5-13 Rebinding decision in case of a new context producer. event.

When a new producer event is received from an underlying context discovery mechanism, a new selection set is made. This set consists of the context offerings of the already bound context producer and the context offerings of the new context producer. This selection set is input to the select activity that selects the most suitable context producer from this set. If the selected context producer is different from the already bound context producer a new context binding is created.

Deregistering Context Source Algorithm

Figure 5-14 shows the rebinding decision activity when considering an incoming ‘deRegisterEvent.

Figure 5-14 Rebinding decision in case of a deregistration event.



In case a de-registration of the bound context producer is received from a context discovery mechanism, the decision is made to rebind after a certain wait period (possibly specified in the CBDL). Hence, new discovery, selection, establishment and monitoring phases are started. Otherwise, when the same producer re-registers, the binding is still valid and the binding mechanism returns to the monitoring activity.

Context Retrieval Error Algorithm

Figure 5-15 shows the rebinding decision activity when considering an incoming ‘contextRetrievalError’.

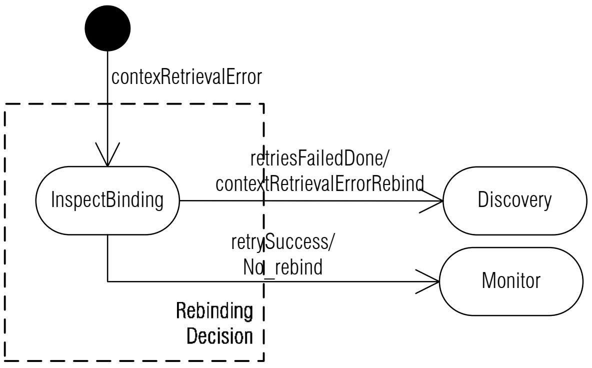


Figure 5-15 Rebinding decision in case of a contextRetrievalError event.

When a context retrieval error is received from the context producer proxy, the binding mechanism inspects the binding by doing multiple attempts to retrieve context information. The number of retries can be specified by the application developer. If all retries fail, the binding is considered unusable and a new discovery, selection, establishment and monitoring phase is started. If one of the retries succeeds (i.e. does not result in an error) the retry counter is reset. If all retries succeed the binding is found to be still useful and the binding is returning to the monitoring activity. Otherwise, the retry counter is reset again until it reaches a number of allowed successes to fail transitions. In *Example 5-1*, we describe, in pseudo Java code, the previously discussed algorithm of the 'InspectBinding' activity. Again, this is merely an example on how to implement a rebinding algorithm.

Example 5-1 Inspect binding algorithm in pseudo Java code.

```
successcntr = 0;      // the number of successive successful context retrievals
failurecntr = 0;      // the number of successive failed context retrievals
fails = 0;            // the number of times context retrieval failed

for(int i=0; (i<#retries | fails < #fails) ; i++){
    Try{
      binding.getContext();    // try to get context information
      successcntr++;           // context retrieval success
      // Transition from failure to success
      if(failurecntr > 0){ i = 0; failurecntr = 0;}
    }catch (Exception e){// context retrieval failed
      failurecntr++;
      //Transition from success to failure
      if(successcntr > 0){ i = 0; successcntr = 0;   fails++;}
    }
}
If(successcntr >0){Monitor.continueMonitor()}       // Success, no rebind
else{Binder.rebindDiscovery()}                      // Failure, rebind
```

Degrading QoC Algorithm

Figure 5-16 shows the rebinding decision activity when considering an incoming 'degradedQoCError'.

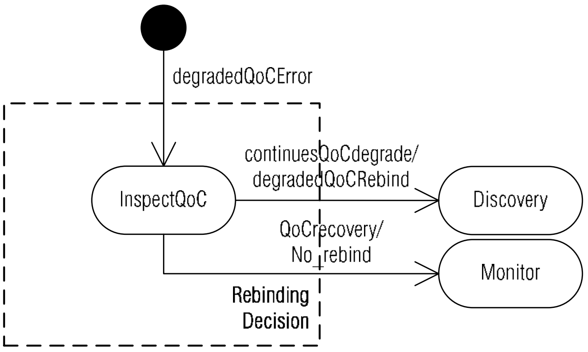


Figure 5-16 Rebinding decision in case of a degradedQoCerror event.

When a notification of the degraded QoC of a bound context producer is received from the context producer proxy, the binding is inspected on the QoC it can offer. In a number of context information retrieval retries, the binding mechanism determines if the QoC remains below the minimal required level. If this is the case, a new discovery, selection, association phase is started. Else, the decision to not rebind is made and the binding mechanism returns to the monitor activity. This can be done in a similar way as the 'inspectBinding' algorithm.

Suspected Producer Unavailable Algorithm

Figure 5-17 shows the rebinding decision activity when considering an incoming 'suspectedUnavailabilityError' event. In this case the context binding mechanisms inspects the binding to determine if it is broken. The 'InspectBinding' actity is explained previously.

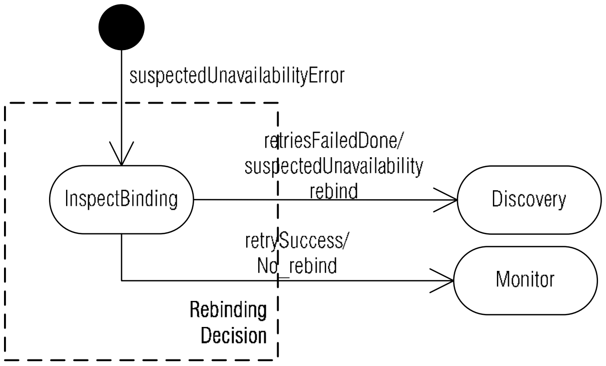


Figure 5-17 Rebinding decision in case of a suspectedUnavailability event.

Concurrent occurrence of context producer events and binding errors

The previous sections discuss the rebinding behaviour for individual context producer events and binding errors. However, these errors/events can occur concurrently. For example, when dealing with a new producer event, another new producer event can be received or the QoC of the current bound producer can degrade.

When considering all possible combinations of producer change events and binding errors, this results in an unmanageable set of possible situations to be dealt with in the context binding algorithm. For example, when considering all combinations of concurrent occurrence of the five possible situations, already 120 (5!) possible situations can be distinguished. Consequently, we assign priorities to events/errors that determine to pre-empt the handling of lower priority events or errors. This means that when the binding mechanism is handling a certain event or error and it receives an event or error with a higher priority, the handling of the current event is pre-empted and the higher priority event or error is dealt with. *Table 5-1* shows the distinguished priorities.

One type of possible events and errors deals with the status of the bound context source. These are deregisterEvent, contextRetrievalError, degradedQoC and suspectedUnavailabilityEvent. When the discovery mechanism explicitly notifies CACI of the deregistration of the bound producer this has the highest handling priority. Occurrence of the other situations in this category are then caused by the disappearing of the bound producer. Similarly for contextRetrievalErrors and suspectedUnavailability, when the bound producer disappears without notification by the discovery mechanism, degrading QoC is caused by the disappeared producer. Hence, the degradedQoCError has the lowest priority.

Table 5-1 Priority of events/errors that deal with the status of the bound producer.

Event / Error	Priority (highest = 1… lowest =4)
deregisterEvent	1
contextRetrievalError	2
suspectedUnavailabilityError	3
degradedQoCError	4

A second type of events deals with the new sources becoming available. This type of event can occur concurrently with the events/errors from the first category. Hence, the space of possible combination consists of the combinations of the events/errors from the first type with the event from the second type, and the combination of all the events/errors with themselves. *Table 5-2*, identifies the possible concurrent combinations and corresponding rebinding actions. As can be seen, several actions are equal or very similar.

Table 5-2 Combinations of concurrent events/errors.

Triggering event/error	Concurrent event/error	Action
deregisterEvent	deregisterEvent	Faulty behaviour, continue the started rebinding decision behaviour.
	newProducerEvent	If the earlier deregistered producer reregisters, keep the binding and return to the monitor state. Else, create a selection set that only contains the new producer offerings and start a new selection phase.
newProducerEvent	newProducerEvent	Add the new producer to the selection set and continue the started rebinding decision behaviour.
	deregisterEvent	Remove the old producer from the selection set and continue the started rebinding decision behaviour.
	contextRetrievalError	Create a new selection set and select the most suitable producer. When the result of this selection is the old producer, inspect the binding. Else rebind to the new producer.
	degradedQoCError	Create a new selection set and select the most suitable producer. When the result of this selection is the old producer, inspect the QoC that can be offered by the bound producer (see the algorithm for handling a degradedQoCError). Else rebind to the new producer.
	suspectedUnavailability Error	Create a new selection set and select the most suitable producer. When the result of this selection is the old producer, inspect the binding (see the algorithm for handling a contextRetrievalError). Else rebind to the new producer.
contextRetrievalError	contextRetrievalError	Continue the started rebinding decision behaviour.
	newProducerEvent	Create a new selection set and select the most suitable producer. When the result of this selection is the old producer, inspect the binding (see the algorithm for handling a contextRetrievalError). Else rebind to the new producer.

degradedQoCError	degradedQoCError	Continue the started rebinding decision behaviour.
	newProducerEvent	Create a new selection set and select the most suitable producer. When the result of this selection is the old producer, inspect the binding (see the algorithm for handling a contextRetrievalError). Else rebind to the new producer.
suspectedUnavailability Error	suspectedUnavailability Error	Continue the started rebinding decision behaviour.
	newProducerEvent	Create a new selection set and select the most suitable producer. When the result of this selection is the old producer, inspect the binding (see the algorithm for handling a contextRetrievalError). Else rebind to the new producer.

5.2.3 Context Publishing Service

This section starts with a high-level overview of the part of the context binding mechanism that realizes the context publishing service. Subsequently, we present the functional decomposition and behaviour of the context binding mechanism that realizes this service.

High-level Overview

Figure 5-18 presents a decomposition of a context producing application and the context binding mechanism. A context producing application (i.e. context-aware application or context source) has application logic that produces context information. The context producing capabilities are specified in a context offering. This offering is sent to the context middleware using the context producer logic. The context offering is advertised via the context binding mechanism to one or more available context discovery mechanisms using their context advertising services. Additionally, a reference to the context logic (CP') is advertised to the available context discovery mechanisms. This reference is internally stored in the discovery mechanism (CP*). Other context-aware applications can retrieve this reference to obtain its context information.

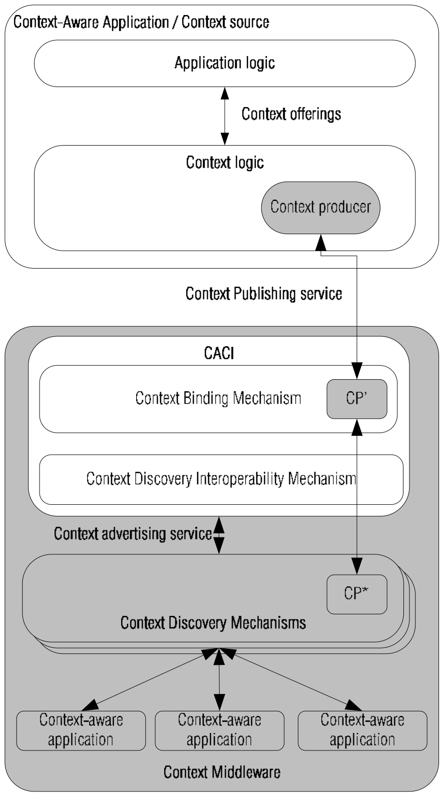

Context-Aware Application / Context source
Application logic
Context offerings
Context logic
Context producer
Context Publishing service
CACI
Context Binding Mechanism
CP'
Context Discovery Interoperability Mechanism
Context advertising service
CP*
Context Discovery Mechanisms
Context-aware application
Context-aware application
Context-aware application
Context Middleware

Figure 5-18
Decomposition of the context binding mechanism.

Functional decomposition

Figure 5-19 presents the functional decomposition of the context binding mechanism. In the figure, we define the functions required to realize the context publishing service. These functions are:

– *Deployer & Parser:* The deployer is responsible for intercepting deploying components. The deployer determines if the context-aware component is a CACI-enabled component and instructs the parser to parse the CBDL description. The parser distils publishing requests from the CBDL document.
– *Binder:* The binder has a coordinating role in the context binding mechanism. It starts the process for the advertisement of the offering of a deploying context producing component to available context discovery mechanisms.
– *Proxy manager:* The proxy manager is responsible for creating a proxy that encapsulates the reference to the context producing application.
– *Publisher:* The publisher is responsible for advertising the offering of a deploying context producing component to available context discovery

mechanisms. This includes registering the proxy to available discovery mechanisms.

- *CACI_database:* The CACI database stores the administration of the context binding mechanism. Additionally, it stores the offerings of a context producing component and the reference to the context producing application. The CACI database exposes the context publishing service to context producing components.
- *GUI:* The GUI is a graphical representation of the CACI database that is used for testing and debugging purposes.

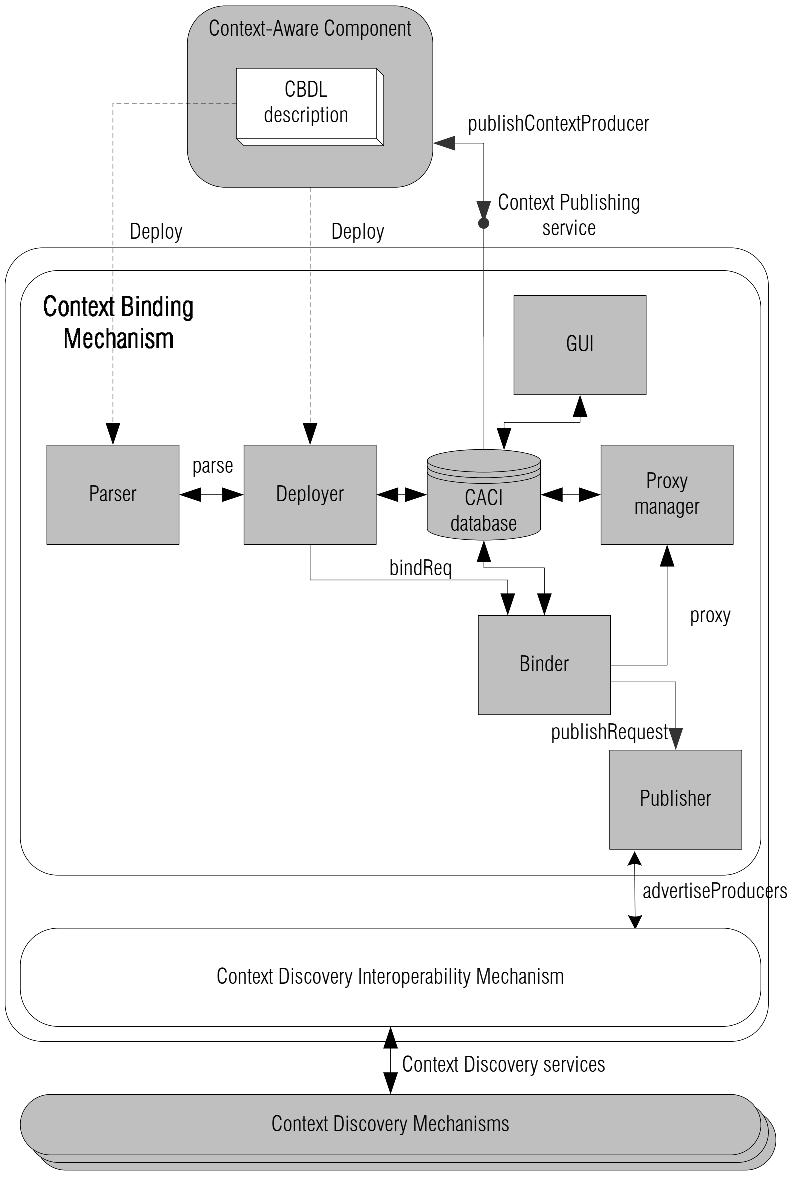
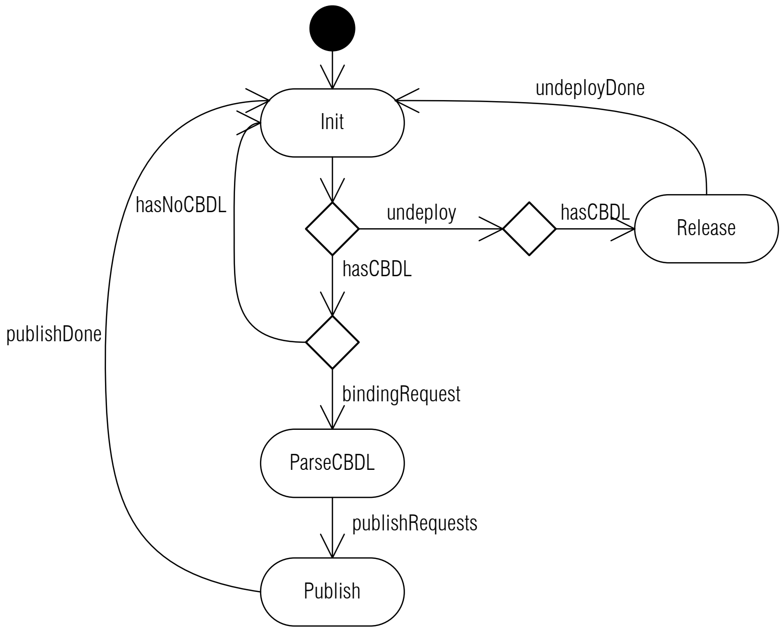


Figure 5-19 Functional decomposition of the context binding mechanism.

Internal Behaviour

Figure 5-20 presents an activity diagram that represents the internal behaviour of the context binding mechanism. The process starts by determining if a deploying component is a CACI-enabled component. If this is the case, the CBDL document is parsed. From the document publishing requests are distilled. The context offerings specified in these requests and the reference to the context publishing application are published to the available context discovery mechanisms. We do not decompose the publishing activity further, as this is not the core of this research.

Figure 5-20 Activity diagram of the behaviour of the context binding mechanism.



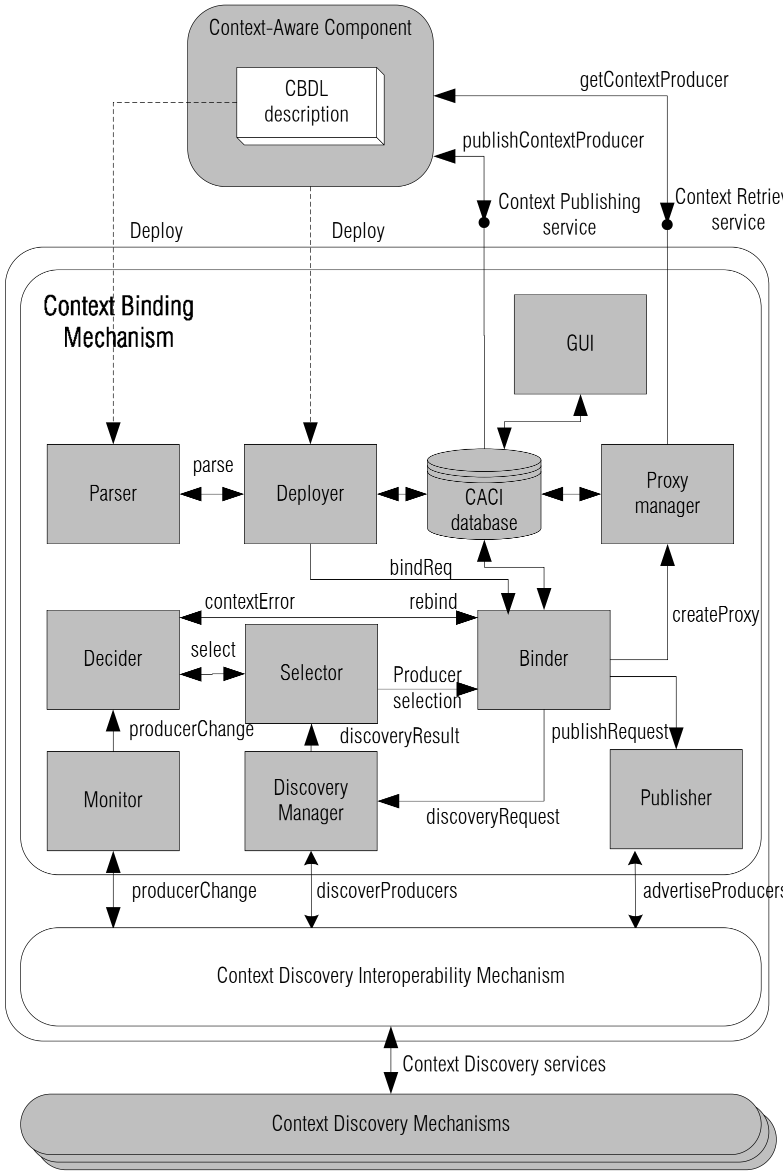
5.2.4 Integrated Design

Partially the previously discussed functions of the design of the context retrieval and publishing service, overlap. For example, detecting deploying components, parsing of the CBDL document and creating context producer proxies is part of the behaviour of both services. In this section, we combine the two designs and present an integrated design, including a functional decomposition and internal behaviour description of the context binding mechanism. This integrated design is implemented in the prototype discussed in the next section.

For the integration, the parser, deployed, CACI database, GUI, proxy manager and binder are combined to offer both functions to support the context publishing and retrieval service. The decider, selector, monitor and discovery manager are unique for the realization of the context retrieval service. The publisher is unique for the realization of the context publishing service.

Figure 5-21 presents an integrated functional decomposition of the complete context binding mechanism. *Figure 5-22* presents an activity diagram of the integrated internal behaviour of the overall context binding mechanism. For a discussion on the individual functions and the internal behaviour, we refer to the previous sections.

Figure 5-21 Integrated functional decomposition of the context binding mechanism.



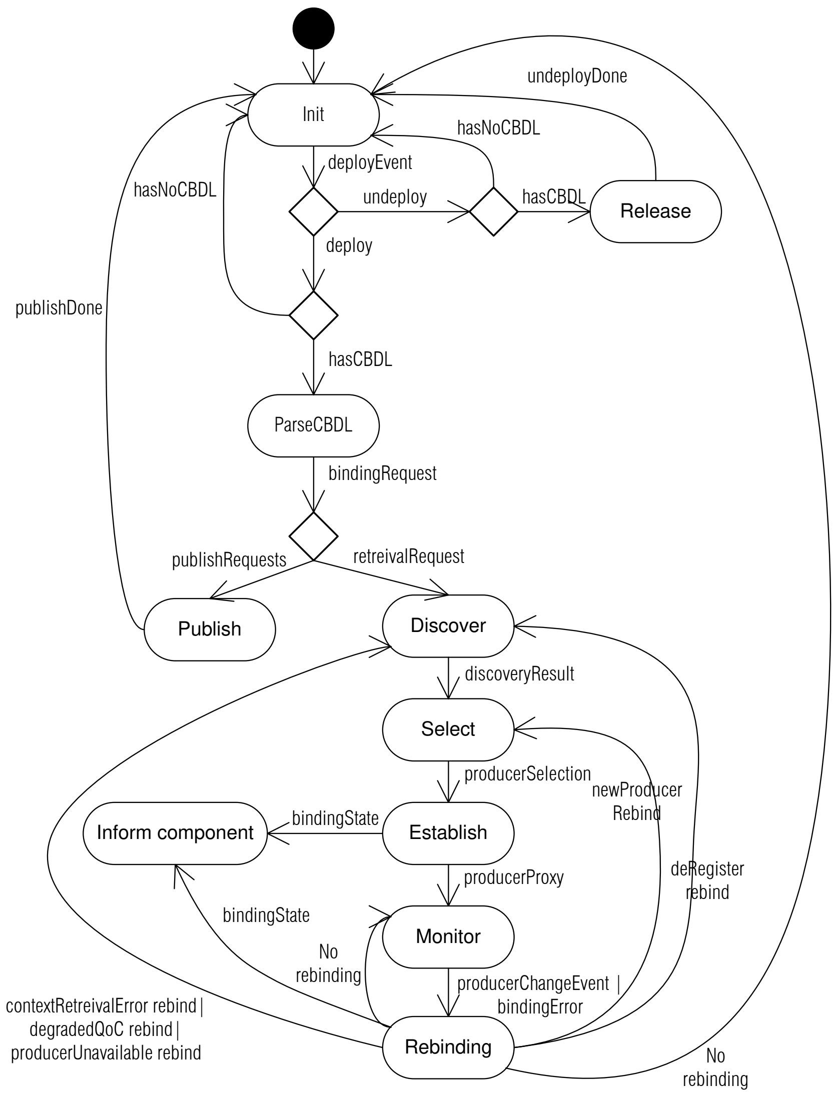


Figure 5-22 Integrated activity diagram of the behaviour of the context binding mechanism.

5.3 Implementation of the Context Binding Mechanism

In this section, we discuss the prototype implementation of CACI. As a foundation of the prototype, we use a light-weight component framework based on the Open Services Gateway Initiative (OSGi) specification. We leverage from the life-cycle and service capabilities of an existing OSGi container and deploy a virtual CACI container that intercepts CACI-enabled context-aware components. Additionally, we use some existing technology such as kXML to parse CBDL documents and Log4J for logging. Additionally, we use SimuContext for simulating context sources for testing purposes. This mechanism is created by us and discussed in detail in Chapter 6. This section starts with an overview of the used technology followed by a discussion on the implementation of the CACI prototype. This section ends with a high-level analysis on the performance, scalability and stability of the prototype.

5.3.1 Used Technology

For the implementation of CACI, we used the following existing technologies: (i) Open Services Gateway Initiative (OSGi), (ii) kXML and (iii) Log4J.

Open Services Gateway Initiative (OSGi)

As the foundation for CACI, we chose a component framework based on the OSGi specification (OSGi Alliance 2004; OSGi Alliance 2005). OSGi defines a specification of a light-weight, extendible and easy to use component framework. Currently, the specification of OSGi is at release 4 (OSGi Alliance 2005).

Figure 5-23 presents the abstract architecture of an OSGi-based component framework. An OSGi framework facilitates the deployment and execution of Java-based components into an OSGi container. A component is called a bundle in terms of OSGi. The container itself can be deployed on top of a Java-based execution environment. The modules layer handles class loading policies of bundles. A feature of OSGi is that it supports class loading on multiple class loaders. Amongst others, this enables class sharing by loading bundles that require sharing of functionality on the same class loader. The lifecycle layer enables dynamic installing, starting, stopping, updating and uninstalling of bundles. The service registry layer enables bundles to register services which can be used by other components. Hence, OSGi supports two ways of inter-component communication:

- *Class sharing:* Bundles can indicate in their component descriptor if they want to export packages or require packages. The class sharing mechanism matches import statements and export statements and places these bundles on the same class loader. Consequently, standard Java invocations can be used to execute functionality from one bundle by another.
- *Service invocations:* A bundle can use the service registry to discover services that are registered by other bundles. It can retrieve a reference to a discovered service and use this reference to invoke service requests.

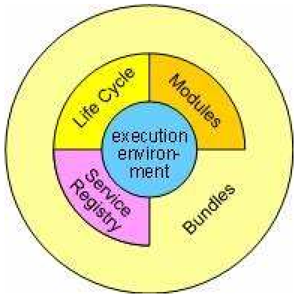


Figure 5-23 OSGi architecture.

A typical usage scenario of OSGi consists of a developer who creates application bundles and installs/starts these bundles using the life-cycle manager. This life-cycle manager uses the module layer for class loading and possible resolving of required shared classes. Furthermore, the developed bundles can discover and use services from other bundles using the service registry.

A bundle is packaged as a standard JAR file with an extended manifest that contains deployment information. This manifest acts as the component descriptor. *Example 5-2* gives an example of a manifest. Information that can be specified in the manifest is for instance; the bundle name, a description, the vendor, and the update location. Furthermore, it can contain information on the packages that should be shared with other bundles (export package) and the packages that the deploying bundle requires to function (import package). The activator property indicates the class that should be started when the component, which incorporates this activator class and possible other classes, is deployed.

Example 5-2 Bundle manifest of the CACI bundle.

```
Bundle-Name = CACI
Bundle-Description = Context-Aware Component Infrastructure.
Bundle-Vendor = Tom Broens
Bundle-Version = 2.5.2
Bundle-UpdateLocation = http://ewi554.ewi.utwente.nl/obr/caci.jar
Bundle-Activator =nl.utwente.CACI.Bundle.CACIActivator
Import-Package = org.ungoverned.osgi.service.shell, org.apache.log4j, org.kxml,
org.kxml.io, org.kxml.parser, nl.utwente.SimuContext, nl.utwente.SimuContext.Repository,
nl.utwente.SimuContext.Configurator
Export-Package = nl.utwente.CACI.Common, nl.utwente.CACI.Common.Interfaces,
nl.utwente.CACI.PerformanceMonitor,nl.utwente.CACI.DiscoveryManager.DiscoveryAdapter,
nl.utwente.CACI.Monitor, nl.utwente.CACI.DiscoveryManager
```

Registering services to the OSGi container and using registered services is done by simply invoking OSGi API's that are offered by the service registry. *Example 5-3* gives a code segment in which 'MyService' is registered and retrieved from the service registry.

Example 5-3 Registering of a OSGi service.

```
// Point of access to the OSGi framework
BundleContext bc;

// Registering a Service
Hashtable props = new Hashtable();
props.put("description", "Service description.");
IMyService myservice = new MyService();
bc.registerService(IMyService.class.getName(), myservice, props);

// Retrieving of a Service
ServiceReference ref = bc_.getServiceReference(IMyService.class.getName());
IMyService my_retrieved_service =
(IMyService) bc.getService(ref);
```

OSGi only provides the specification of a component framework. Specific implementations of this specification exist. Amongst others, the following initiatives offer open-source OSGi implementations:

– *Oscar* (Oscar.org 2005): Research initiative, currently offering a light-weight implementation of the OSGi release 3 specification. Oscar is tested and fully functional on a pocket pc running the IBM J9 virtual machine. Transition of Oscar to release 4 of the OSGi specification is done in the Felix project (Apache Felix Project 2006).
– *Knopflerfish* (Knoplerfish.org 2005): OSGi project maintained by Gatespace Telematics, currently offering an implementation of the OSGi release 4 specification.
– *Equinox* (Equinox 2006): OSGi project originating from the eclipse project. They offer an implementation of the OSGi release 4 specification.
– *Osxa* (Osxa 2006): Research project offering currently a fairly limited implementation of the OSGi release 4 specification.

For more insights on open source OSGi implementations we refer to Campanelli (Campanelli 2007). Due to the standard specification of an OSGi implementation, CACI should be able to function on all the aforementioned OSGi implementations. We tested the CACI prototype on top of the Oscar and Knopflerfish frameworks.

kXML

For representing CBDL descriptions, we use XML as the de-facto standard for representing structured data. For parsing the CBDL descriptions, we use the kXML 1.21 pull parser (kXML project 2006). This is a lightweight XML parser developed to operate on mobile devices. *Example 5-4* gives a code segment with an example on how to use kXML to parse an XML document.

Example 5-4 Parsing of a XML document using kXML.

```
XmlParser xmlparser = new XmlParser(reader);
ParseEvent pe = xmlparser.read();
while (pe.getType() != Xml.END_DOCUMENT) {
  if(pe.getType() == Xml.START_TAG){
    if (pe.getName().equals("tag_name")){
      // Handle tag 'tag_name'
    }
  // Handle other tags
  }
// Handle other tag types like close tags etc.
pe = xmlparser.read(); // Read the following event.
}
```

First the parser is created using a pointer to the XML file. The XML file is read sequentially and XML events are collected. These events can for instance be 'start document', 'end document', 'start tag', 'end tag' etc. Until the document has ended, events have to be handled. Depending on the tags and the actions to be taken, application code has to be added to handle the event.

Log4J

For logging purposes, we use the Apache Log4j libraries (Apache Log4J project 2006). Log4j enables developers to specify their logging requirements in a logging configuration file. Hence, changing logging behaviour does not affect the application code. Furthermore, the output pattern, which defines the information you want to log and how it is formatted, and the log location (e.g. file, console, remote server) can be changed at run-time. *Example 5-5* gives an example indicating how to log with Log4j using a configuration file and log statements.

Example 5-5
Configuration and usage of Log4J

```
// Configure Log4J (done once)
PropertyConfigurator.configure(System.getProperty("l.utwente.CACI.logfilecfg","log4j.cfg"));

// Create a logger (done for every class)
private Logger logger = Logger.getLogger(MyClass.class);

// Log statements
logger.info("This is a information log statement.");
logger.debug("This is a debug log statement.");
logger.error("This is a error log statement.");

// ***************
// Log4J.cfg configuration file
log4j.rootLogger=info, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d][%p] %m - %p %C{1}:%L%n
```

First the logging framework has to be configured for the specific needs of the application using a configuration file. For every class that needs logging, a logger has to be retrieved. Log statements can be made using this logger. These statement can have different levels of severity:
- *FATAL:* The FATAL level designates very severe error events that presumably lead the application to abort.
- *ERROR:* The ERROR level designates error events that might still allow the application to continue running.
- *WARN:* The WARN level designates potentially harmful situations.
- *DEBUG:* The DEBUG Level designates fine-grained informational events that are most useful to debug an application.
- *INFO:* The INFO level designates informational messages that highlight the progress of the application at coarse-grained level.

The configuration file describes what to log and where to log it. A rootLogger is defined which specifies the level of logging. For example, INFO level shows all log statements, while ERROR level only shows ERROR and FATAL log statements. Additionally, the configuration file specifies the output mechanism(s). In the case of the example, console output is generated following a certain pattern as specified by the 'ConversionPattern'.

5.3.2 Prototype Implementation

In this section, we discuss the implementation of the CACI prototype. It starts with the overall implementation architecture of the prototype.

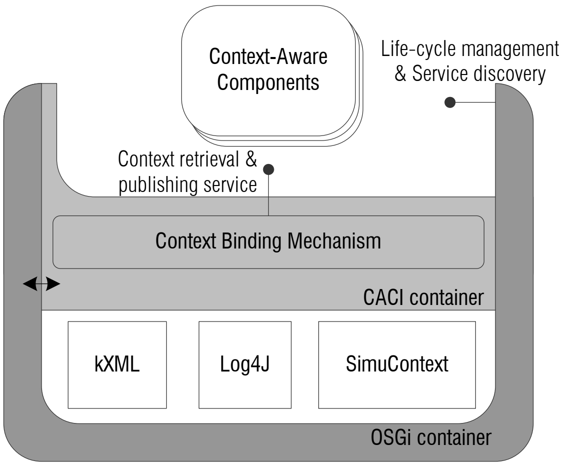
Subsequently, it discusses the extensibility of the prototype using application specific adapters. Finally, it describes a test scenario showing the usage of CACI.

Overall Implementation

The foundation of CACI is the CACI container. The CACI container, which contains the context binding mechanism, is implemented in Java as a standard OSGi bundle. The package structure of the CACI implementation corresponds with the identified functional blocks from the design as presented in *Figure 5-21*. The CACI bundle contains approximately 3500 lines of code and has a size of approximately 70kb. We tested CACI on a laptop PC and a windows mobile PDA. A total installation of CACI, including the OSGi environment and the J9 virtual machine, requires 3,5MB.

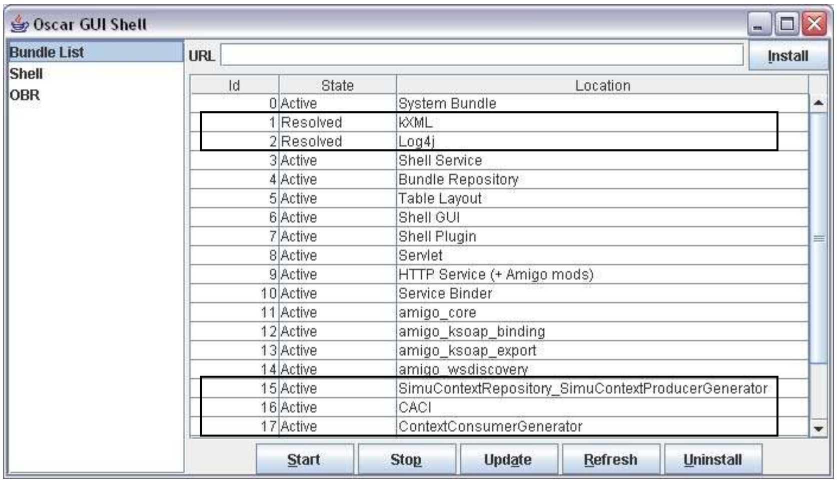
Figure 5-24 presents the implementation architecture of the prototype. The CACI container implements a virtual container inside the OSGi container. The CACI container intercepts deploying CACI-enabled components such that the context binding mechanism can create the required context bindings, which are specified in the component descriptor. Additionally, the context binding mechanism offers the context retrieval and publishing services to the context-aware application to retrieve or publish context information.

Figure 5-24
Implementation architecture.


Context-Aware Components
Life-cycle management & Service discovery
Context retrieval & publishing service
Context Binding Mechanism
CACI container
kXML
Log4J
SimuContext
OSGi container

Besides these services, a context-aware component can use the life-cycle and service discovery services offered by the underlying OSGi container. Internally, the context binding mechanism uses kXML, Log4J and SimuContext functionality. These functions are packaged in separate bundles and deployed in the OSGi container. The context binding mechanism can use their capabilities via the services capabilities of the OSGi container. Consequently, context-aware components can also use these bundles independently from CACI.

Besides the bundle containing the CACI container, we have created a context consumer generator for testing and debugging purposes. This generator contains a graphical user interface to automatically generate context consuming components. We discuss this generator more in the next section. *Figure 5-25* shows the installed components in an Oscar based OSGi container. Here you can see the kXML, Log4J, SimuContextRepository, CACI and ContextConsumerGenerator bundles.



Id	State	Location
0	Active	System Bundle
1	Resolved	kXML
2	Resolved	Log4j
3	Active	Shell Service
4	Active	Bundle Repository
5	Active	Table Layout
6	Active	Shell GUI
7	Active	Shell Plugin
8	Active	Servlet
9	Active	HTTP Service (+ Amigo mods)
10	Active	Service Binder
11	Active	amigo_core
12	Active	amigo_ksoap_binding
13	Active	amigo_ksoap_export
14	Active	amigo_wsdiscovery
15	Active	SimuContextRepository_SimuContextProducerGenerator
16	Active	CACI
17	Active	ContextConsumerGenerator

Figure 5-25 Graphical representation of the installed components in a Oscar OSGi container.

Application Specific Adapters

We implemented the prototype in a modular fashion such that it can be extended by application developers to suit their application specific needs. Application developers can develop and configure the following application specific adapters:

- <u>*Parser adapters:*</u> to support different types of context requirement specification languages.
- <u>*Deployer adapters:*</u> to support different types of underlying component middleware frameworks.
- <u>*Selector adapters:*</u> to support different context source selection algorithms.
- <u>*Decider adapters:*</u> to support different decision algorithms to determine, in case of a 'failing' context binding, if a rebinding process should be started.

CACI can be configured by specifying the classpath of the application specific adapters using system properties (successively, 'nl.utwente.CACI.ParserAdapter', 'nl.utwente.CACI.DeployerAdapter', 'nl.utwente.CACI.SelectorAdapter' and 'nl.utwente.CACI. DeciderAdapter'). Based on the specified classpath of the adapters, a specific adapter is instantiated at run-time using Java reflection. *Example 5-6* shows the code for instantiating a parser adapter, which is similar for the other types of adapters.

Example 5-6 Using Java reflection to instantiate a parser adapter.

```
IParserAdapter parser;

public Parser() {
    // Retrieve the system property specified by the application developer.
    String parsername = System.getProperty("nl.utwente.CACI.ParserAdapter",
                        "nl.utwente.CACI.Parser.ParserAdapter.CBDLParser");
    try {
        // Instantiate the adapter and set the callback object
        parser = (IParserAdapter) Class.forName(parsername).newInstance();
        parser.setCallback(deployer);
    } catch (InstantiationException e) {
        logger.error("Cannot instantiate parser adapter.");
    } catch (IllegalAccessException e) {
        logger.error("No permission to instantiate parser adapter.");
    } catch (ClassNotFoundException e) {
        logger.error("Cannot find class to instantiate parser adapter.");
    }
}
```

For all the adapter types, we have created interfaces (see *Figure 5-26* to *5-29*) that have to be implemented by the specific plug-in adapters. These interfaces define one method that CACI uses to set a callback object ('setCallback'). On initialization of the adapter this 'setCallback' is invoked by CACI to set a callback object. This callback object should be used by the adapter to interact with CACI to pass its results. The only requirement on the classes that implement the adapter is that they should be available on the classpath of CACI. This can be done by using the class sharing capabilities of the OSGi framework.

Figure 5-26 depicts the interfaces relevant for a parser adapter. A parser adapter has to implement the 'IParserAdapter' interface. This includes implementing a 'setCallback' and 'parse' method. The 'parse' method is called by CACI on deployment of a new CACI enabled component. This method should parse the component descriptor of this component using the passed input stream. Additionally, it should notify identified retrieval and publishing request to CACI via the 'IParserAdapterCallback' callback object. For the prototype, we created a CBDL parser adapter to parse CBDL-based component descriptors.

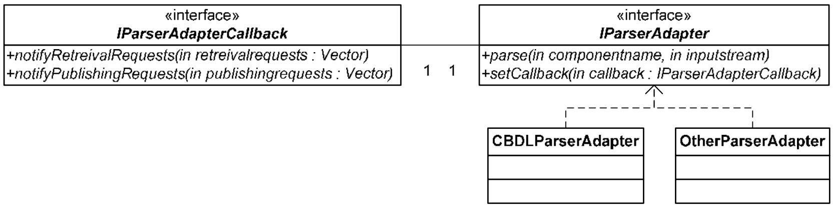

«interface»
IParserAdapterCallback
+notifyRetreivalRequests(in retreivalrequests : Vector)
+notifyPublishingRequests(in publishingrequests : Vector)
1
1
«interface»
IParserAdapter
+parse(in componentname, in inputstream)
+setCallback(in callback : IParserAdapterCallback)
CBDLParserAdapter
OtherParserAdapter

Figure 5-26 Interfaces to develop parser adapters.

Figure 5-27 depicts the interfaces relevant for a deployer adapter. A deployer adapter has to implement the 'IDeployerAdapter' interface. This includes implementing a 'start', 'stop' and 'setCallback' method. The 'start/stop' methods are called by CACI to start or stop the deployer adapter from detecting (un)deploying components in the underlying component container. When the adapter is started it should indicate deploying or undeploying components to CACI via the 'notifyComponentInstall' and 'notifyComponentUnistall' methods via the callback object. For the prototype, we have implemented an OSGi deployer adapter to detect deploying OSGi bundles.

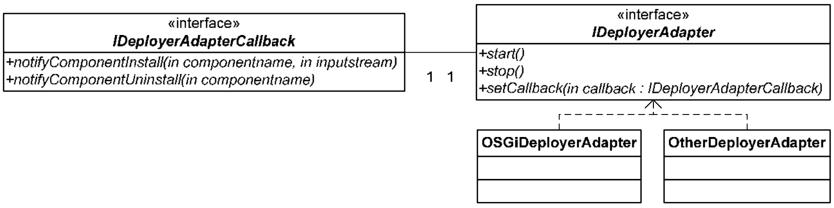

«interface»
IDeployerAdapterCallback
+notifyComponentInstall(in componentname, in inputstream)
+notifyComponentUninstall(in componentname)
1
1
«interface»
IDeployerAdapter
+start()
+stop()
+setCallback(in callback : IDeployerAdapterCallback)
OSGiDeployerAdapter
OtherDeployerAdapter

Figure 5-27 Interfaces to develop deployer adapters

Figure 5-28 depicts the interfaces relevant for a selector adapter. A selector adapter has to implement the 'ISelectorAdapter' interface. This includes implementing a 'select' and 'setCallback' method. The 'select' method is called by CACI in two cases: (i) after CACI receives discovery results and (ii) after CACI receives a notification of a new context source becoming available. In the second case, CACI creates a producer set consisting of the new producer and the old producer to be able to compare both. The implementation of the select methods ranks the vector of passed 'ContextProducers' and selects the most suitable context producer. This can be done based on the element, entity, format and QoC offerings of the producer. QoC offerings consisting of minimal and maximal QoC limits this producer can offer (see Chapter 2 for possible QoC parameters). This selected producer is passed to CACI via the "notifySelection" method of the callback object. For the prototype we have implemented a simple selector that selects the first producer from the list of discovered context producers.

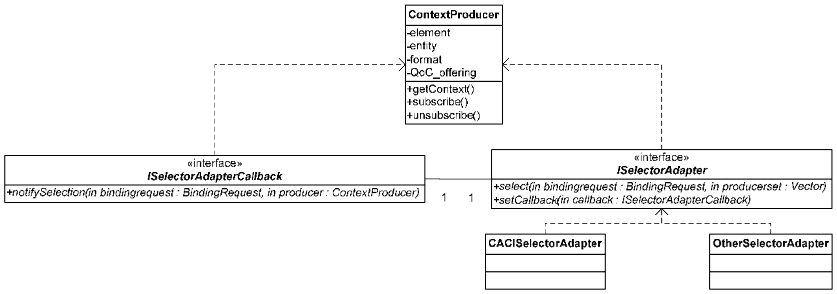

ContextProducer
-element
-entity
-format
-QoC_offering
+getContext()
+subscribe()
+unsubscribe()
«interface»
ISelectorAdapterCallback
+notifySelection(in bindingrequest : BindingRequest, in producer : ContextProducer)
1
1
«interface»
ISelectorAdapter
+select(in bindingrequest : BindingRequest, in producerset : Vector)
+setCallback(in callback : ISelectorAdapterCallback)
CACISelectorAdapter
OtherSelectorAdapter

Figure 5-28 Interfaces to develop selector adapters.

Figure 5-29 depicts the interfaces relevant for a decider adapter. A decider adapter has to implement the 'IDeciderAdapter' interface. This includes implementing a 'notifyDeregisterProducer', 'notifyNewProducer', 'notifyDegradedQoC', 'notifyContextRetrievalError', 'notifyBindingInspection' and 'setCallback' method. The decider has to decide to commence rebinding when the notify methods are called by CACI in the following cases:

- *notifyDeregisterProducer:* The bound producer deregisters (i.e. dissapears) from its discovery mechanism.
- *notifyNewProducer:* A new and possibly better (i.e. indicated by the selector) context producer becomes available.
- *notifyDegradedQoC:* The QoC of retrieved context samples from the bound context producer is below the minimal required QoC level.
- *notifyContextRetrievalError:* After an explicit request for context information by the application, an exception occurs (i.e. catched by the context producer proxy).
- *notifyBindingInspection:* In case of a subscription by the application, no new context information is received for a certain period.

When the decider decides to start the rebinding process, it invokes the notify methods of the callback object. In case of a rebinding due to new context producers becoming available, an establishment phase is started (notifyDirectRebind). In the other cases, a new discovery, selection and establishment phase is started (notifyDiscoveryRebind). For the prototype we implemented a decider based on the rebinding algorithm discussed in Section 5.2.2.

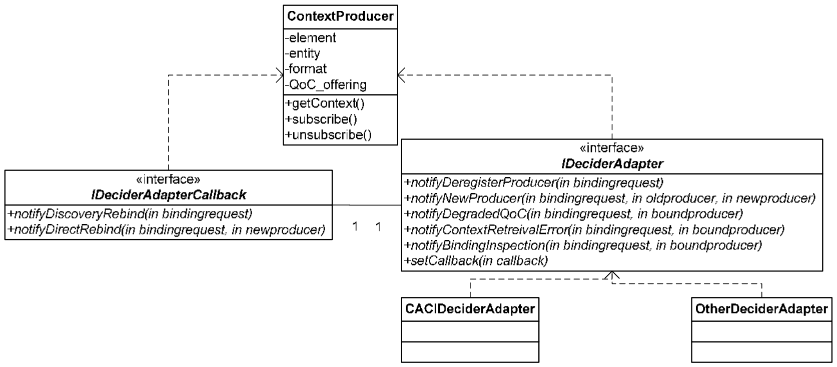

ContextProducer
-element
-entity
-format
-QoC_offering
+getContext()
+subscribe()
+unsubscribe()
«interface»
IDeciderAdapterCallback
+notifyDiscoveryRebind(in bindingrequest)
+notifyDirectRebind(in bindingrequest, in newproducer)
1
1
«interface»
IDeciderAdapter
+notifyDeregisterProducer(in bindingrequest)
+notifyNewProducer(in bindingrequest, in oldproducer, in newproducer)
+notifyDegradedQoC(in bindingrequest, in boundproducer)
+notifyContextRetreivalError(in bindingrequest, in boundproducer)
+notifyBindingInspection(in bindingrequest, in boundproducer)
+setCallback(in callback)
CACIDeciderAdapter
OtherDeciderAdapter

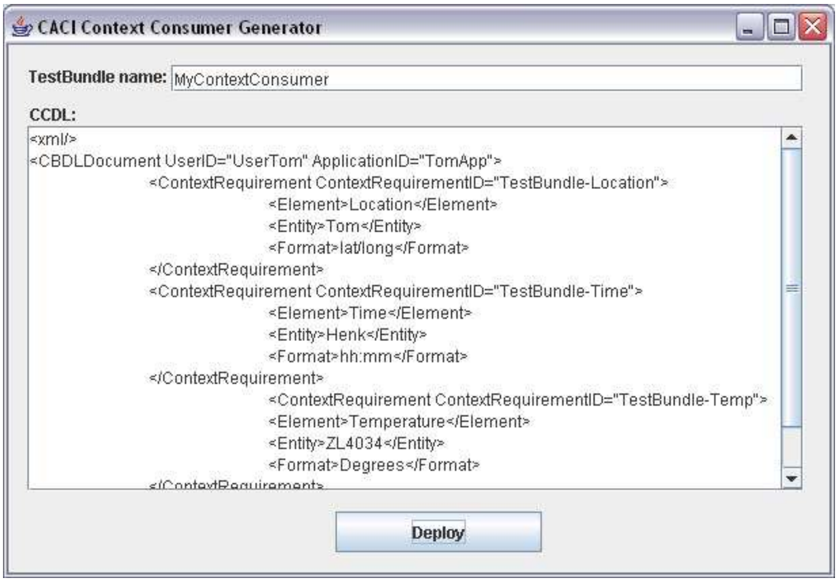
Figure 5-29 Interfaces to develop decider adapters.

Testing & Debugging

For testing and debugging purposes on a Desktop PC, we created graphical user interfaces to (i) automatically generate and deploy context consuming components, (ii) monitor the state of the context binding mechanism and (iii) monitor incoming context information at the generated context consuming component. We used these testing instruments to perform feasibility test in which we run common use case scenarios that should be supported by CACI.

The context consumer generator can be used to generate a context consuming component by specifying the component name and its context requirements in the CBDL language. *Figure 5-30* presents the GUI of the context consumer generator. This GUI can be used to specify that the component to be generated should have the name 'MyContextConsumer' and has three context requirements: (i) Location of Tom in lat/long format, (ii) Time of Henk in hh:mm format, and (iii) Temperature of room ZL4034 in degrees Celsius.

By pressing the deploy button a consumer component is generated. This is done by specializing a pre-defined template component according to the specified name and CBDL document. This transformation is done using an ANT script. The generated code is compiled and packaged in a JAR file together with the CBDL document. The packaged component is automatically deployed in the OSGi/CACI container.



TestBundle name: MyContextConsumer

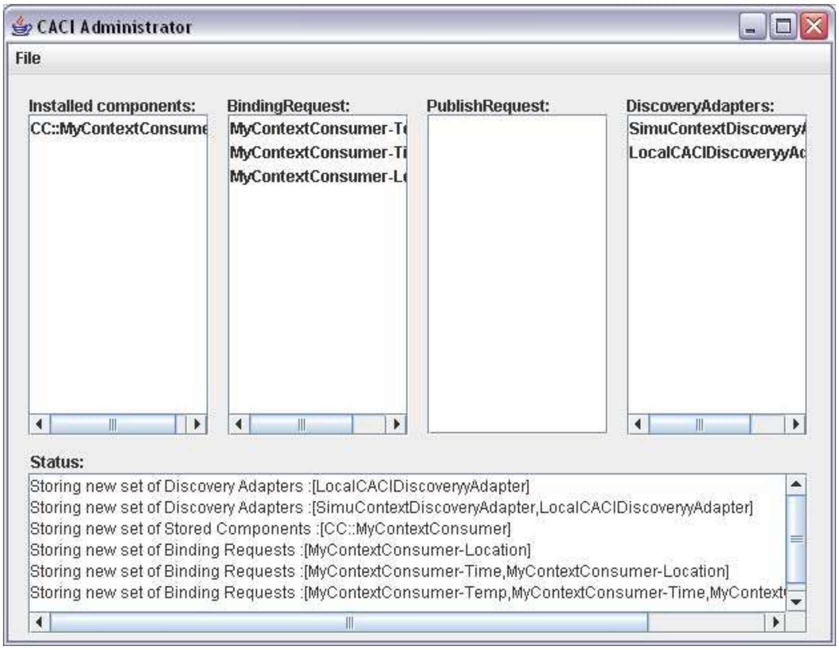
CCDL:

```
<xml/>
<CBDLDocument UserID="UserTom" ApplicationID="TomApp">
            <ContextRequirement ContextRequirementID="TestBundle-Location">
                        <Element>Location</Element>
                        <Entity>Tom</Entity>
                        <Format>lat/long</Format>
            </ContextRequirement>
            <ContextRequirement ContextRequirementID="TestBundle-Time">
                        <Element>Time</Element>
                        <Entity>Henk</Entity>
                        <Format>hh:mm</Format>
            </ContextRequirement>
                        <ContextRequirement ContextRequirementID="TestBundle-Temp">
                        <Element>Temperature</Element>
                        <Entity>ZL4034</Entity>
                        <Format>Degrees</Format>
            </ContextRequirement>
```

Figure 5-30 GUI of the context consumer generator.

The result of the deployment of the 'MyContextConsumer' component can be seen in the CACI Administrator GUI, which is presented in *Figure 5-31*. On the left, the 'MyContextConsumer' component is added to the list of installed components. The CBDL document of the component is parsed and the ID's of the context requirements are added to the list of binding requests that are known to CACI.

In general, the CACI Administrator GUI gives an overview of the state of the context binding mechanism. When a (generated) component is deployed, the component and its binding requests are registered. Additionally, the GUI shows possible context publishing requests from context producing components. Furthermore, it shows the registered context discovery adapters, which are discussed in more detail in Chapter 6.



CC::MyContextConsume

MyContextConsumer-T
MyContextConsumer-Ti
MyContextConsumer-L

SimuContextDiscovery
LocalCACIDiscoveryyAc

```
Storing new set of Discovery Adapters :[LocalCACIDiscoveryyAdapter]
Storing new set of Discovery Adapters :[SimuContextDiscoveryAdapter,LocalCACIDiscoveryyAdapter]
Storing new set of Stored Components :[CC::MyContextConsumer]
Storing new set of Binding Requests :[MyContextConsumer-Location]
Storing new set of Binding Requests :[MyContextConsumer-Time,MyContextConsumer-Location]
Storing new set of Binding Requests :[MyContextConsumer-Temp,MyContextConsumer-Time,MyContext
```

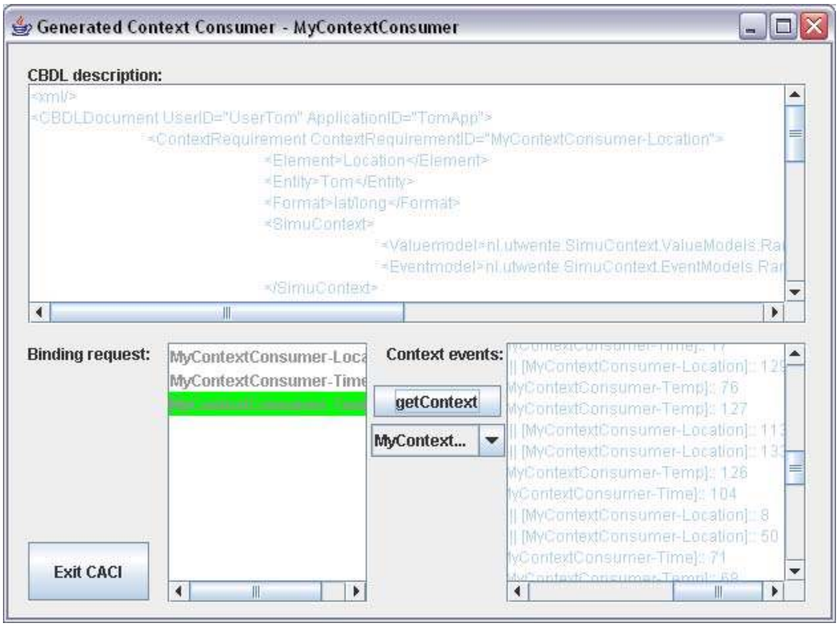
Figure 5-31 CACI Administrator GUI that can be used to monitor the state of the context binding mechanism.

Figure 5-32 shows the generated GUI of the 'MyContextConsumer' component. This GUI shows the CBDL specification and context requirements, called binding requests, of the component. Additionally, it shows incoming context events such as incoming context samples. Context samples are received as the result of an explicit request to the bound context source or via a notification from the bound context source in case of a subscription.

Binding status notifications received from CACI are visualized by colouring the context binding request label. A green label indicates the 'bound' state of a context binding, an orange label indicates the '(re)binding state' of this binding while a white label indicates an 'unbound' state of this binding. Hence, in the figure, the context requirement 'Temperature of room ZL4034' can be fulfilled and context information is delivered to the application. No context information is available anymore to fulfil the other context requirements.

As the counter part of context consuming components, context producing components can also be automatically generated and deployed in CACI using the SimuContext Configurator. This is discussed in more detail in Chapter 6.

Figure 5-33 shows a running instance of CACI on a windows mobile PDA. It shows the logging messages inside the console of the IBM J9 virtual machine, which is used as the underlying execution platform. For testing and debugging on this platform we used command line statement.



Generated Context Consumer - MyContextConsumer

CBDL description:

```
<xml/>
<CBDLDocument UserID="UserTom" ApplicationID="TomApp">
                <ContextRequirement ContextRequirementID="MyContextConsumer-Location">
                                <Element>Location</Element>
                                <Entity>Tom</Entity>
                                <Format>lat/long</Format>
                                <SimuContext>
                                                <Valuemodel>nl.utwente.SimuContext.ValueModels.Ra
                                                <Eventmodel>nl.utwente.SimuContext.EventModels.Ra
                                </SimuContext>
```

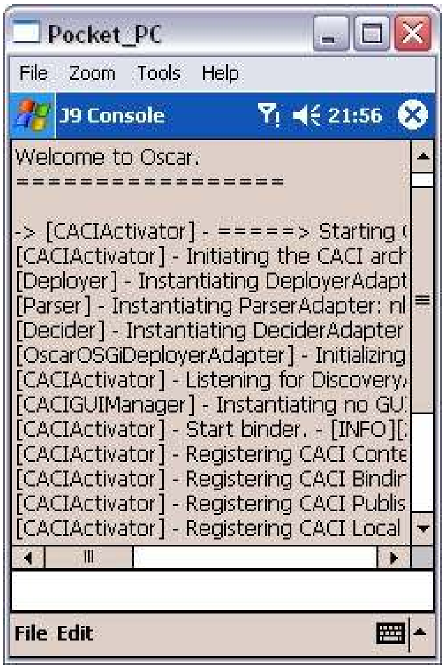
Binding request:

```
MyContextConsumer-Loca
MyContextConsumer-Time
```

Context events:

```
|| [MyContextConsumer-Location]:: 129
MyContextConsumer-Temp]:: 76
MyContextConsumer-Temp]:: 127
|| [MyContextConsumer-Location]:: 113
|| [MyContextConsumer-Location]:: 133
MyContextConsumer-Temp]:: 126
lyContextConsumer-Time]:: 104
|| [MyContextConsumer-Location]:: 8
|| [MyContextConsumer-Location]:: 50
lyContextConsumer-Time]:: 71
```

Figure 5-32 GUI of the generated context consumer component.



```
Welcome to Oscar.
=================

-> [CACIActivator] - =====> Starting (
[CACIActivator] - Initiating the CACI arch
[Deployer] - Instantiating DeployerAdapt
[Parser] - Instantiating ParserAdapter: nl
[Decider] - Instantiating DeciderAdapter
[OscarOSGiDeployerAdapter] - Initializing
[CACIActivator] - Listening for Discovery
[CACIGUIManager] - Instantiating no GU
[CACIActivator] - Start binder. - [INFO][
[CACIActivator] - Registering CACI Conte
[CACIActivator] - Registering CACI Bindir
[CACIActivator] - Registering CACI Publis
[CACIActivator] - Registering CACI Local
```

Figure 5-33 Running instance of CACI on a windows mobile PDA.

5.3.3 Performance, Scalability and Stability

In this section, we discuss performance, scalability and stability aspects of the CACI prototype.

Performance

We executed some performance tests to get an impression on the efficiency of CACI and the context binding mechanism. *Figure 5-34* depicts the used test set-up. We generated a context consumer using the context consumer generator. This consumer is deployed in the CACI container. As part of the binding process, CACI discovers simulated context producers from a local SimuContext producer repository. In this repository, we register simulated context sources that fulfil the context requirements of the context consumer.

Using this set-up, we performed four types of tests that measure the time spend to: (i) start-up CACI, (ii) establish a new binding to a context source registered in the SimuContext repository, (iii) rebinding to a new SimuContext source when the already bound context source disappears and (iv) rebinding to an appearing SimuContext source.

To get an overall impression on the time spend by CACI to create and maintain a context binding on behalf of a context-aware application, all the measurements, except the start-up time of CACI, are performed at the generated context consumer. The generated context consumer measures the time to create a binding and rebind based on context binding status notifications it receives from CACI. CACI's start-up time is measured inside CACI.

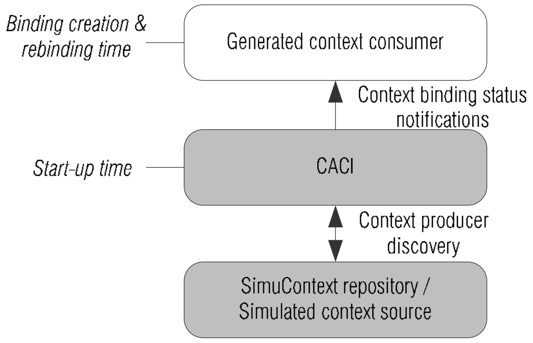

Binding creation & rebinding time
Generated context consumer
Context binding status notifications
Start-up time
CACI
Context producer discovery
SimuContext repository / Simulated context source

Figure 5-34 Test set-up

The results of these tests are depicted in *Table 5-3*. The tests are performed with the test setup running on a laptop pc and a windows mobile PDA. The tests are repeated 250 times. The results show the lowest, average and highest time spend for the corresponding test.

Table 5-3 Performance test results.

Test	Time spend on a laptop pc	Time spend on a PDA
i) Start-up of the CACI infrastructure	Low: 0ms Avg: 15ms High: 47ms	Low: 23ms Avg: 26ms High: 636ms
ii) Establish a new binding	Low: 0ms Avg: 7ms High: 94ms	Low: 13ms Avg: 120ms High: 967ms
iii) Re-bind to new context source due to disappearing of bound context source	Low: 0ms Avg: 5ms High: 16ms	Low: 10ms Avg: 88ms High: 263ms
iv) Re-binding to an appearing context source (e.g. higher QoC)	Low: 0ms Avg: 3ms High 16ms	Low: 10ms Avg: 73ms High: 255ms

When compared to the time-frame of a typical context-aware application, we consider the overhead introduced by CACI, which is in the range of milliseconds, marginal. Some initial overhead is witnessed when starting the CACI infrastructure. However, the CACI infrastructure and applications can be started independently. Hence, CACI can already be pre-loaded such that it does not influence the start-up of the application. Establishing a new context binding takes, as expected, the most time. This includes parsing the CBDL document, discovery and selection of context sources, and establishment of the context binding by creating a new proxy. In case of the rebinding tests (i.e. tests iii and iv), parsing of the CBDL document and creation of the proxy are not necessary and hence these operations are less costly. Re-binding to a new context source in case of a failing context binding is a little bit more costly compared to re-binding in case of an appearing context source. This is as expected because the first also includes a discovery process to find a replacement for the context source while in the second case this replacement context source is already provided.

In the test, we purposely used a local discovery mechanism (i.e. the SimuContext repository) to minimize the time spend on discovery. In this way the results of the test give a representative impression on the efficiency of CACI. However, in a real-world deployment of CACI the used discovery mechanisms are often remote. Consequently, we estimate that the overhead introduced by CACI is less or can even be neglected compared to the delay for discovery of context producers at a remote context discovery mechanism. For example due to introduced communication delay to a remote context discovery mechanism. Furthermore, without CACI, an application still has to perform (remote) discovery. Consequently remote discovery does not impose extra overhead when using CACI.

Scalability

Scalability of CACI and the context binding mechanism is determined by the number of application components and the number of binding requests it can handle concurrently. As CACI is co-located with the applications on the application hosts, we assume the number of components, and hence binding requests, are fairly limited.

We tested the scalability of the CACI prototype by deploying 10 CACI enabled components with in total 100 binding requests. The time spend to create a new binding or to rebind to new context sources showed similar overhead as in the performance tests presented in the previous section. This is as expected because every binding request is handled sequentially.

Sequential handling of binding requests also has a disadvantage: some binding requests are handled later than other binding requests depending on the time they are sent to the context binding mechanism. For example, on average this means that the time between the deployment of a component with multiple binding requests and the time a proxy is available, is a multitude of 7,3ms (i.e. average time spend to establish a new context binding). Similar reasoning applies to the time spent for rebinding in case of disappearing or newly available context sources. Nevertheless, for a small number of context bindings the overhead (i.e. which is in the order of milliseconds) introduced by CACI is relatively small.

Concurrent handling of context binding requests may reduce the time spend on creating and rebinding of context bindings. For example, this could be useful to use the waiting time introduced by a remote discovery request more efficiently. We consider concurrent handling of binding requests as future work.

Stability

The rebinding behaviour of the context binding mechanisms may become instable such that the availability of context information becomes interrupted. For example, a situation could occur in which the context binding mechanism, on behalf of one or more context bindings, binds between appearing and disappearing context sources. During the switching, no context information is available to the application. Also such a situation could appear if the QoC of a bound context source fluctuates heavily in and out of the acceptable quality range specified by the application.

Selection and rebinding algorithms should be developed to overcome those situations. For example, a waiting time before switching to a new context source could be applied such that it is more certain that this new context source stays available. Or, in case of fluctuating QoC, an algorithm that takes into account QoC averages over a period of time instead of considering the QoC values of every individual context sample. However, we consider extending the context binding mechanism with such algorithms as future work.

Another danger of providing rebinding capabilities are ‘denial of service’ (DoS) attacks. A malicious context source could, by specifying a fake context offering with a very high QoC, let the context binding mechanism create bindings only to this source. Additionally, by rapidly appearing and disappearing, it could cripple the context binding mechanism, which in that case is constantly trying to rebind from and to this source. This could for example be countered by introducing certificates that ensure the trustworthiness of the context source. We refer to (Anderson, Roscoe et al. 2004) for a general discussion on certificates and DoS in internet-based applications. However, this aspect needs further research.

5.4 Related Work

In general, current middleware binding mechanisms have two related goals: (i) shift parts of the binding process to the infrastructure to make the binding more implicit for the developer (explicit vs. implicit bindings (Blair and Stefani 1998)) and (ii) only create the binding when needed (at run-time), incorporating the situation at hand (early vs. late binding). Examples of middleware binding mechanisms are the CORBA Naming Service (OMG 2004), the CORBA Trader (OMG 2004), CORBA Dynamic Invocation Interface (DII) (OMG 2004), Web service UDDI and the Web Service Invocation Framework (WSIF) (Apache WSIF project 2006). However, dynamic behaviour of the binding, like appearing and disappearing binding ends are not incorporated in these mechanisms. Coping with this issue is still the responsibility of the application developer.

Several mechanisms, such as Context-sensitive bindings (Sen and Roman 2003), Service-oriented network sockets (Saif and Palusak 2003) and OSGi (Extended) Service Binder (Cervantas and Hall 2004; Bottaro and Gerodolle 2006), recognize the need for extending binding mechanisms in which the dynamic availability of services is incorporated. Compared to CACI, they have a similar goal but are not tailored to the novel type of context-aware applications. Although, context producers and consumers could be considered generic distributed entities, they have distinct characteristics (e.g. limited scope and interface, QoC) that have to be incorporated in the binding mechanism to be able to fully support the application developer. For example, context binding mechanisms should be based on a model of context and the notion of quality of context (QoC) should be incorporated in the mechanisms.

When zooming into binding mechanisms specific for context-aware systems, several context-aware middleware infrastructures (Kummerfeld, Quigley et al. 2003; Bardram 2005; Henricksen, Indulska et al. 2005; Kranenburg and Eertink 2005) shift parts of the context specific binding mechanism to the infrastructure. Generally, this functionality enables context consumers to discover and bind to context producers using programming statements. However, often dynamic monitoring capabilities are not available and when context producers become (un)available the decision to re-bind and the choice to which context source to bind has to be taken by the application rather than the infrastructure. CACI tries to leverage from the capabilities of current context-aware middleware infrastructures and extend them with binding maintenance that copes with the dynamic availability and quality of context producers. Additionally, we take a different approach, compared to most context management systems, by offering developers ways to specify their context requirements in high level descriptions. These descriptions are interpreted and dealt with by CACI instead of developers having to program technology specific binding statements. In this way, we further facilitate developers in rapidly creating context-aware applications.

5.5 Limitations and Future work

We acknowledge that the current design and prototype has certain limitations and can be extended:

- *Prototype optimization:* the current implementation of the prototype assumes certain time-out values (e.g. discovery session time) which could be tuned to optimize performance. Additionally, concurrent handling of binding requests could be researched to improve performance and scalability.
- *Context producer support:* the implementation of the support function for context producers is limited to advertisements in the local repository. Full support for context producers also includes advertisement of the context offerings to available context discovery mechanisms and deregistering in case of un-deployment of the component. Additionally, research has to be done on how-to de-register the context offerings in a certain context discovery mechanism in case of disappearance of the context producer.
- *Dynamic QoC re-binding:* the current implementation of the context binding mechanism only considers re-binding on static QoC parameters. It matches the offered QoC of an appearing context source with the required QoC specified by the application developer. However, the actual QoC of the incoming context samples is not dealt with.

Further research is needed to extend the context binding mechanism with this aspect.

- *Stability:* more research has to be done to decision algorithms to overcome possible instable situations of context bindings. Both oscillating behaviour due to rapidly fluctuating QoC and availability of context sources have to be taken into account.
- *Reasoning:* in case certain context requirements cannot be matched with context offerings of context sources a context reasoning mechanism could be deployed to infer this required type of context. Both horizontal and vertical reasoning techniques could be used. The first to maintain the required QoC, the second to derive required context information from available context sources.
- *Privacy:* especially for context producing components the CACI container could function as a privacy enforcement point. Future research is necessary to extend the CBDL language with privacy parameters. These parameters can be used by an extended context binding mechanism to enforce the privacy of the owner of the context information provided by the context producing component.

Chapter 6

Context Discovery and Simulation in CACI

This chapter presents the design and prototype implementation of: (i) a mechanism that enables context-aware applications to interoperate with different dynamically available context discovery mechanisms and (ii) a mechanism to simulate context sources (coined SimuContext). Parts of this chapter are published in (Aarts 2006; Broens and Halteren 2006; Broens, Poortinga et al. 2007; Hesselman, Benz et al. 2008).

This chapter is structured as follows: Section 6.1 discusses the design and prototype implementation of the context discovery interoperability mechanism. Section 6.2 presents the design and prototype implementation of the SimuContext framework.

6.1 Context Discovery Interoperability Mechanism

This section discusses the context discovery interoperability mechanism. This mechanism supports the context binding mechanism (discussed in Chapter 5), to discover context sources. This section starts with a problem analysis, followed by a description of the design and prototype implementation. Subsequently, it discusses the integration of this mechanism in CACI. Finally, it presents related work and, limitations and future work.

6.1.1 Problem Analysis

A major enabler for the development of second and third generation context-aware applications are context discovery mechanisms. These mechanisms can be used to find context sources that can deliver context information suitable for the application. As discussed in Chapter 3, currently, a vast amount of context discovery mechanisms exist, which have different capabilities and scope (Dey and Abowd 2000; Bardram 2005; Henricksen, Indulska et al. 2005; Benz, Hesselman et al. Freeband AWARENESS Dn2.1, 2006).

An important characteristic of current context discovery mechanisms is that they are often developed for specific application environments. For example, some discovery mechanisms are specifically geared towards home environments (Lehmann, Bauer et al. 2004), whereas others are dedicated to large-scale mobile environments (Lehmann, Bauer et al. 2004; Roussaki, Strimpakou et al. 2006), or to small ad-hoc networks (Perich, Avancha et al. 2002). For context-aware applications, it is complex to interoperate with these different types of discovery mechanisms. For instance, because these mechanisms have different operational interfaces, use different discovery protocols, different naming schemes for their users, or different context information models. This means that context-aware applications are generally limited to their 'native' context discovery mechanism.

We believe it is unlikely that there will be one commonly adopted context discovery mechanism in the future. As implied by the diversity of currently available context discovery mechanisms, different application environments may require different mechanisms to exchange context information. Consequently, context-aware applications have the possibility to use a range of context discovery mechanism to find context sources.

Additionally, the range of a context discovery mechanism is often limited to a certain domain. For example, a certain discovery mechanism is only reachable if the application is in the same network domain. These domains are also often physically limited. For example, a discovery mechanism is only reachable via the wireless network, deployed in an office building. Hence, this discovery mechanism becomes unavailable when an application moves out of the range of this network. A mobile context-aware application is likely to travel between domains. Hence, during its lifespan, it may encounter multiple heterogeneous context discovery mechanisms.

Figure 6-1 illustrates a mobile user with a context-aware application. This user travels from domain to domain, encountering multiple heterogeneous context discovery mechanisms.

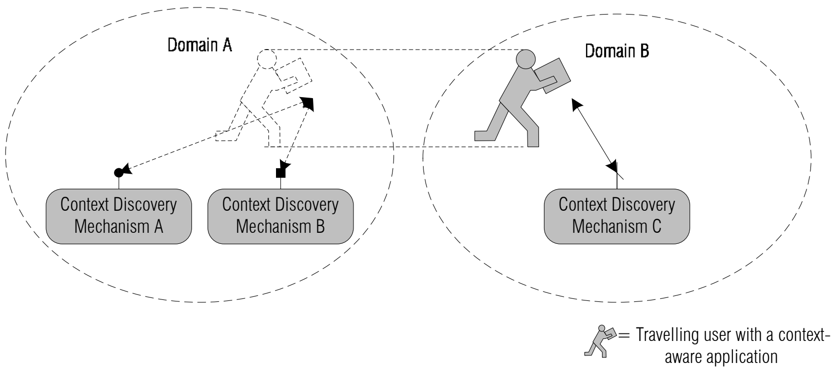

Domain A
Domain B
Context Discovery Mechanism A
Context Discovery Mechanism B
Context Discovery Mechanism C
= Travelling user with a context-aware application

Figure 6-1 Travelling user encountering multiple heterogeneous context discovery mechanisms.

Without supporting mechanisms to cope with the previously sketched situation, developers of a context-aware application have to consider diverse discovery mechanisms in their application. Additionally, they may have to develop code to detect and monitor the availability of these context discovery mechanisms. Besides the required, substantial, programming effort, this also distracts from the primary task of developing context-aware applications. Hence, we propose to shift the responsibility of interoperating context-aware applications with heterogeneous and dynamically available context discovery mechanisms to an infrastructure-based context discovery interoperability mechanism. For example, such a mechanism enables the following scenario of a buddy navigation application (see *Example 6-1*).

Example 6-1 Scenario showing the use of different context discovery mechanisms during the lifespan of a "buddy navigation" application.

Dennis is a young adult, always wanting to be in contact with his friends. He has a mobile device running the 'buddy navigation application'. This application is able to navigate to available buddies by using location and availability context information of him and his friends. Dennis notices that Monica is in the mall and available for a cup of coffee. He decides to visit her. He instructs the 'buddy navigation application' to help him find her.

Inside Dennis' home, an RFID based location context source, found by his home context discovery mechanism, provides an accurate location of Dennis. From Monica, no precise location source is available in Denis's home, it is only known that she is somewhere in the mall. The 'buddy navigation application' instructs Dennis to take the car to the mall. When Dennis leaves his home, to go on his way to Monica, his home discovery mechanism becomes unavailable. The application switches to a cell based location context source found by the context discovery mechanism of his telecommunication provider.

On entering the mall in which Monica is in, accurate context information on Monica's location becomes available, offered by a Bluetooth beacon context source found by the context discovery mechanisms of the mall. The buddy navigation application pops up a map of the mall, to instruct Dennis how to walk to the book store where Monica is currently shopping.

In our view, there are three approaches to enable context-aware applications to interoperate with context discovery mechanisms:

– Standardization: every environment provides one or more standardized context discovery mechanisms. These mechanisms can be found and accessed in a standardized manner when an application enters the domain. However, as already indicated, due to the heterogeneity and different requirements of the application environments, we do not believe this approach is feasible or likely.
– Bridging: In a bridging approach, every environment provides different types of discovery mechanisms which are internally bridged to other discovery mechanisms by bridging code. The application interacts with one or a limited set of context discovery mechanisms. Context sources, registered in other bridged context discovery mechanisms, are discovered via these context discovery mechanisms. *Figure 6-2* illustrates this approach. For more information on an implementation of the bridging approach to interoperate context-aware applications with context discovery mechanisms see (Hesselman, Benz et al. 2008).

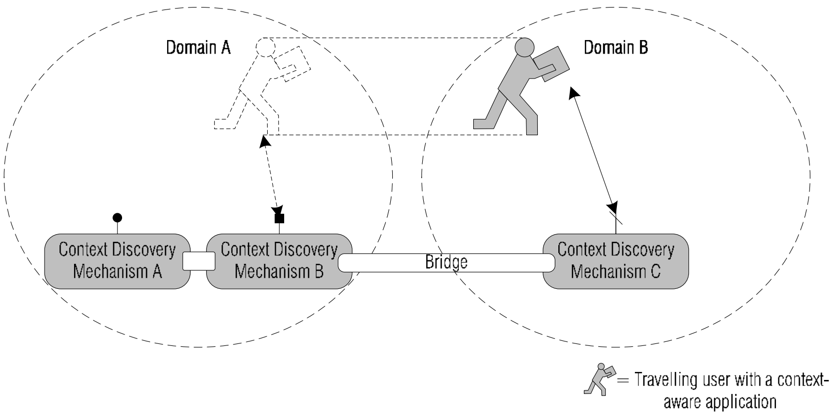

Domain A
Domain B
Context Discovery Mechanism A
Context Discovery Mechanism B
Bridge
Context Discovery Mechanism C
= Travelling user with a context-aware application

Figure 6-2 The use of bridges to interoperate context discovery mechanisms.

– Homogenizing: In a homogenizing approach different context discovery mechanisms are homogenized by a generic homogenizing mechanism. The application interacts with this mechanism to discover context sources from available context discovery mechanisms. The homogenizing mechanism is responsible for detecting available discovery mechanisms and executing the discovery on behalf of the application. This homogenizing mechanism can travel with the application. Dynamically downloaded adapters enable the homogenizing mechanism to interoperate with the specific context discovery mechanisms. *Figure 6-3* illustrates this approach.

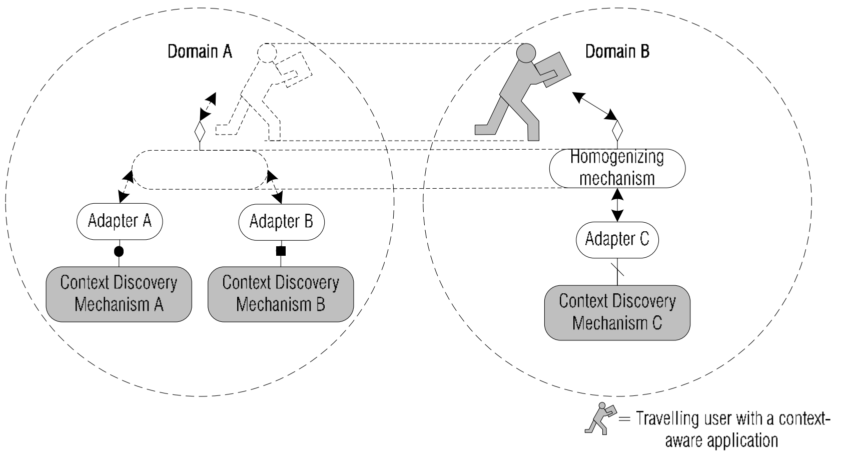

Domain A
Domain B
Homogenizing mechanism
Adapter A
Adapter B
Adapter C
Context Discovery Mechanism A
Context Discovery Mechanism B
Context Discovery Mechanism C
= Travelling user with a context-aware application

Figure 6-3 The use of adapters to homogenise the access to dynamically available context discovery mechanisms .

In *Table 6-1,* we compare some (dis)advantages of the bridging and homogenizing approach.

Table 6-1 Comparing the bridging and homogenizing approach.

Bridging	Homogenizing
- Every combination of context discovery mechanisms requires separate bridges.	+ Every discovery mechanism requires only one adapter.
- Developers of a bridge require extensive knowledge on the (two) discovery mechanisms they are bridging.	+ Developers of an adapter require only knowledge on the discovery mechanism for which they are providing an adapter.
- The application needs to have knowledge of at least one context discovery mechanism.	+ The application only requires to have knowledge on the homogenizing mechanism to potentially access multiple context discovery mechanisms.
+ Suitable to interoperate context discovery mechanisms that reside in different domains.	- Not suitable to interoperate context discovery mechanisms residing in different domains.
+ Bridges can consider specific capabilities of the context discovery mechanisms it bridges and can offer these capabilities to the application.	- Adapters can offer capabilities that are supported by the homogenizing mechanism to the application.

We acknowledge that a bridging and homogenizing approach have their particular uses, and could even be combined. In this section, we explore the homogenizing approach. This approach corresponds with the local type of support mechanisms CACI offers. Additionally, a homogenizing approach offers the opportunity to dynamically use available context discovery mechanisms to create and maintain context bindings. Hence, we develop a homogenizing *context discovery interoperability mechanism* that enables the context binding mechanism to transparently interoperate with dynamically available context discovery mechanisms.

6.1.2 Design

Figure 6-4 presents the position of the context discovery interoperability mechanism in CACI. The context binding mechanism (see Chapter 5) transforms the context requirement of a context-aware application into a discovery request. It invokes this discovery request by using the context discovery service of the context discovery interoperability mechanism. The context discovery interoperability mechanism is responsible for detecting available context discovery mechanisms, issuing discovery requests to the available discovery mechanisms on behalf of the application, and collect the results. The results are forwarded to the context binding mechanism that uses them to create and maintain context bindings.

In the remainder of this section, we start with a high level overview of the context discovery interoperability mechanism. Subsequently, we present a functional decomposition.

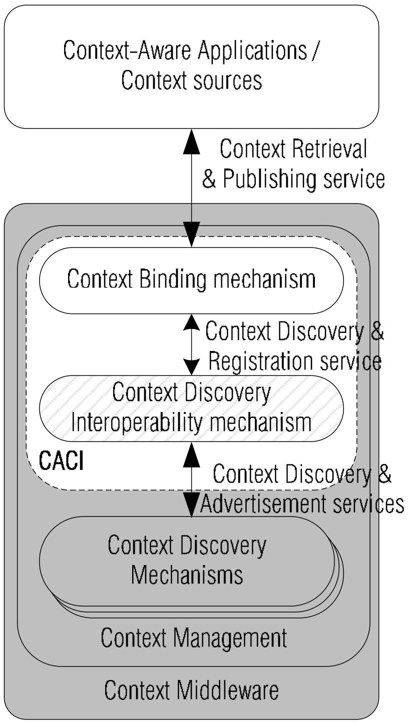

Context-Aware Applications / Context sources
Context Retrieval & Publishing service
Context Binding mechanism
Context Discovery & Registration service
Context Discovery Interoperability mechanism
CACI
Context Discovery & Advertisement services
Context Discovery Mechanisms
Context Management
Context Middleware

Figure 6-4 Position of the context discovery interoperability mechanism in CACI..

High-level Overview

The problems the interoperability mechanism has to solve are the following:

- (Un)availability of context discovery mechanisms during the life-span of the application. We propose to detect the availability of context discovery mechanisms and continuously monitor their availability.
- Heterogeneous interaction behaviour and communication mechanisms of context discovery mechanisms. We propose to introduce an adapter as a middleman between a discovery mechanism and the context binding mechanism to overcome this heterogeneity. These adapters are dynamically downloaded when the application enters a network domain and a context discovery mechanism is detected.
- Heterogeneous syntax of the applied information models of context discovery mechanisms. We propose to use the adapter as a middleman between a discovery mechanism and the context binding mechanism to overcome this heterogeneity.

In the remainder of this section, we discuss how the context discovery interoperability mechanism solves these problems in more detail. We acknowledge that it is important to tackle, besides its syntax, also the semantics of the information models of context discovery mechanisms. However, we consider this out of the scope of this work.

Figure 6-5 presents a further decomposition of the context discovery interoperability mechanism in relation with a context discovery mechanism. Involved entities are:

- *Context discovery interoperability mechanism:* offers a standard and generic discovery service to the context binding mechanisms. The interoperability mechanism performs a discovery to detect available context discovery mechanisms. For a detected context discovery mechanism an adapter is available.
- *Adapter supplier:* offers an adapter supplier service to the context discovery interoperability mechanism. This service can be used to detect context discovery mechanisms and retrieve corresponding adapters. An adapter can be used to perform a discovery on a specific context discovery mechanism.
- *Context discovery mechanism:* offers a specific context discovery service that can be used to find context sources.

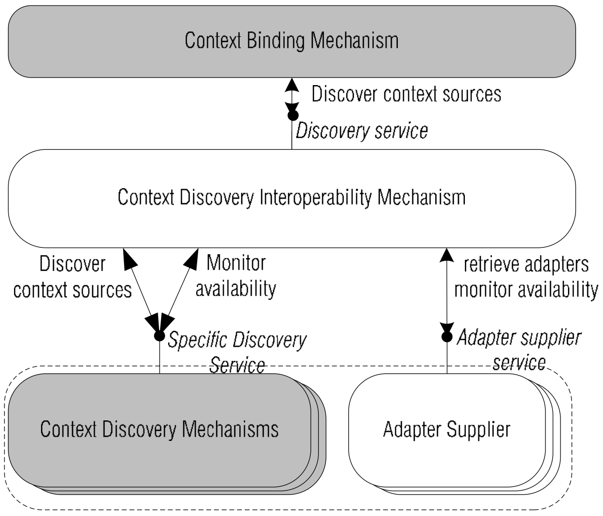

Context Binding Mechanism
Discover context sources
Discovery service
Context Discovery Interoperability Mechanism
Discover context sources
Monitor availability
retrieve adapters monitor availability
Specific Discovery Service
Adapter supplier service
Context Discovery Mechanisms
Adapter Supplier

Figure 6-5 Refinement of the context discovery interoperability mechanism and context discovery mechanisms.

Hence, besides the context discovery mechanism itself, an adapter has to be provided to enable CACI to discover context sources. Additionally, when CACI wants to dynamically detect the context discovery mechanism once it enters its domain, an adapter supplier has to be running in this domain. An adapter supplier has the responsibility of providing adapters for the specific context discovery mechanisms within its domain. Multiple adapter suppliers can co-exist (e.g. multiple suppliers servicing multiple application environments). Their location is not prescribed (i.e. a supplier is not restricted to be co-located on the same host running the context discovery mechanism).

Table 6-2 presents the abstract service primitives of the discovery service offered by the context discovery interoperability mechanism. It describes the service primitives (SP) between the Service User (SU, in this case the context binding mechanism) and the Service Provider (SPr, in this case the context discovery interoperability mechanism). Additionally, it describes the type of interaction (i.e. synchronous and asynchronous), and the input parameters and possible return parameters.

This service consists of a way to start a discovery for context sources based on a discovery request (discoveryProducers). A discovery request consists of a context specification and possible QoC criteria. The service user provides a callback which is called to transfer the corresponding discovery results.

Table 6-2 Discovery service.

Direction	**S/A**	**SP identifier**	**Parameters**	**ReturnParameters**
SU-SPr	A	discoverProducers	discoveryRequest, callback	-
SPr-SU	A	notifyDiscoveryResult	-	discoveryResult

Table 6-3 discusses the abstract service primitives of the adapter supplier service. This service can be used to retrieve a list of available adapters, get a download URL of an adapter, and send a heartbeat signal to check the availability of the adapter supplier.

Table 6-3 Adapter supplier service.

Direction	S/A	SP identifier	Parameters	ReturnParameters
SU-SPr	S	listAdapters	-	AdapterIDs
SU-SPr	S	getAdapterURL	AdapterID	AdapterURL
SU-SPr	S	heartbeat	-	Acknowledgement

Functional Decomposition

Figure 6-6 presents a functional decomposition of the context discovery interoperability mechanism. The mechanism consists of the following functions:

– *Adapter:* an adapter 'translates' the generic context discovery request provided by the context binding mechanism to a specific context discovery request to specific context discovery mechanisms (and visa versa for the discovery result). This includes translating between the possibly different information models and using the right communication technologies to invoke the discovery request. For integration with the discovery coordinator, it offers the same discovery service as the overall context discovery interoperability mechanism.
– *Monitor:* a monitor, continuously checks the availability of a specific context discovery mechanism. In case of detected unavailability of a context discovery mechanism, it notifies the discovery coordinator.
– *Discovery Coordinator:* the coordinator downloads corresponding adapter/monitor combinations for the detected discovery mechanisms and load/unloads them when the mechanisms are available or unavailable, respectively. Additionally, it implements the discovery service offered to the context binding mechanism. Hence, it is responsible for dispatching the received discovery requests, via the loaded adapters, to the context discovery mechanisms.

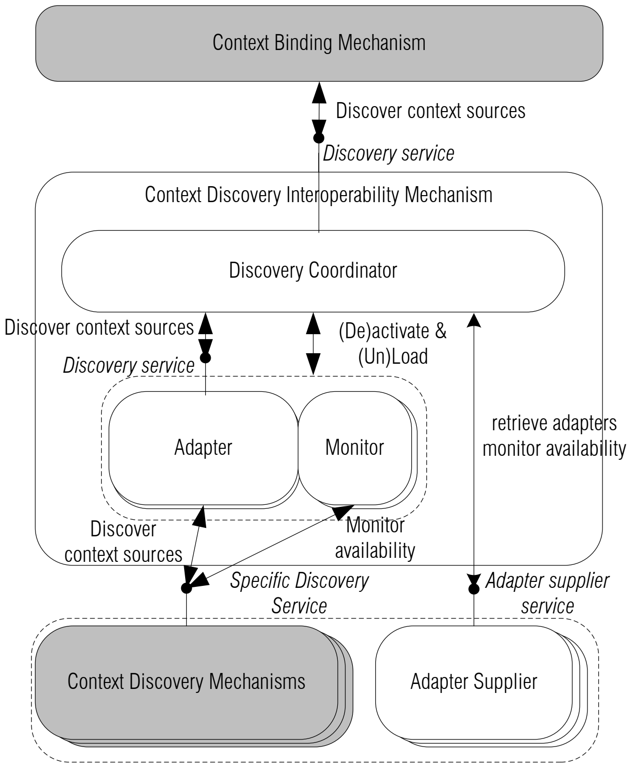


Figure 6-6 Architecture of the Context Discovery Interoperability mechanism.

A typical behaviour of the discovery interoperability mechanism is represented in a time-sequence diagram in *Figure 6-7.* On start-up of the application, the Discovery Coordinator initiates the discovery of available adapter suppliers (1); this is done continuously during the life-span of the discovery coordinator. When a supplier is found its registered adapter/monitor combinations are downloaded (2). The monitor is started (3) to check the availability of the discovery mechanism (4). If it is indeed available, the corresponding adapter is registered to the Discovery Coordinator. When the application then performs a discovery request, the coordinator will use the downloaded adapters to discover context sources (5 & 6). The monitor is continuously keeping track of the availability of the discovery mechanism it supports (7). If discovery mechanisms become unavailable, their adapters are made inactive by the coordinator (8). Also the adapter supplier is monitored for its availability (9). In case a supplier disappears, its inactive adapters/monitors are unloaded from the system.

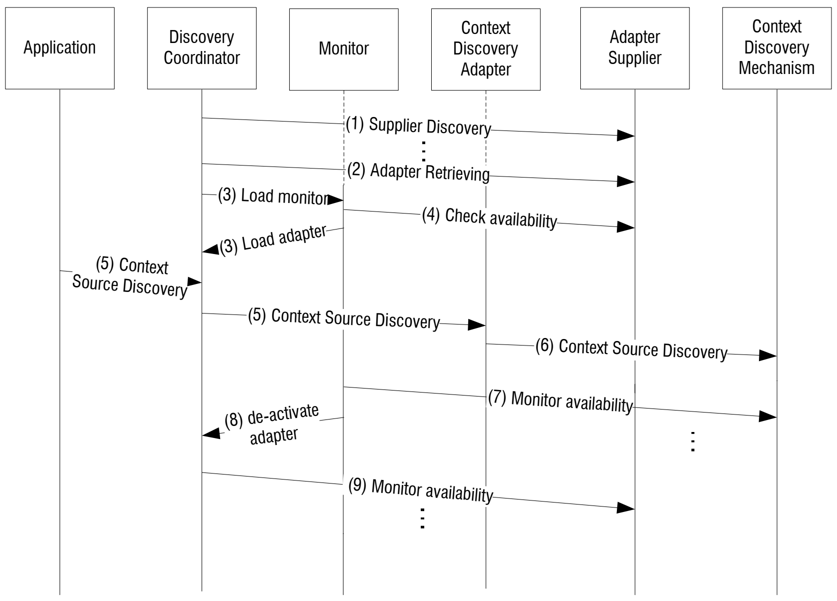


Figure 6-7 Time-sequence diagram of the behaviour of the context discovery interoperability mechanism.

The figures represent only one monitor and adapter, multiple monitors and adapters can co-exist at the same time and can become active or inactive during the lifespan of the application.

6.1.3 Implementation

This section discusses the prototype implementation of the context discovery interoperability mechanism. It discusses the used technology, usage scenario, performance tests and a reference implementation of an adapter/monitor.

Used Technology

We created a Java based prototype using the OSGi component framework. The functional components depicted in the design are implemented as separate OSGi bundles (e.g. coordinator, adapter bundles). The prototype (excluding discovery adapters and monitors) consists of approximately 1000 lines of code and the OSGi bundles have a size of around 30kB. We tested the prototype both on a laptop PC and a windows mobile PDA. Context discovery adapter and monitor components are implemented for the CCS, CMS, (Benz, Hesselman et al. Freeband AWARENESS Dn2.1, 2006), and SimuContext (Broens and van Halteren 2006).

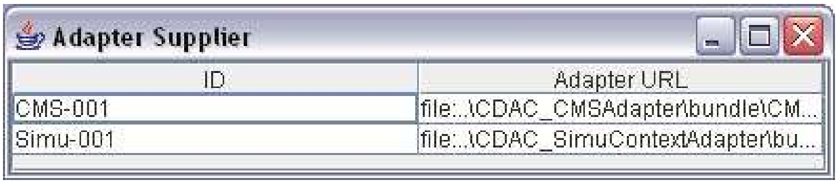
For the discovery of adapter suppliers, we used the middleware developed in the IST Amigo project (http://www.amigo-project.org). This middleware offers, amongst others, functions for easy Web Service communication and Web Services Dynamic Discovery (WS-Discovery).

WS-Discovery uses multicast to discover web services in a network. Consequently, we used WS-Discovery as the 'standard' discovery mechanism for finding adapter suppliers.

Usage Scenario

In order to be discoverable by a discovery coordinator, an adapter supplier registers its adapter supplier Web Service with a scope of 'urn:CDIM' and a service type of 'IAdapterSupplier' (i.e. interface specifying the Adapter Supplier Service, see *Table 6-3*). The adapter supplier is configured with the information on which adapters/monitors it can offer and the adapter URLs.

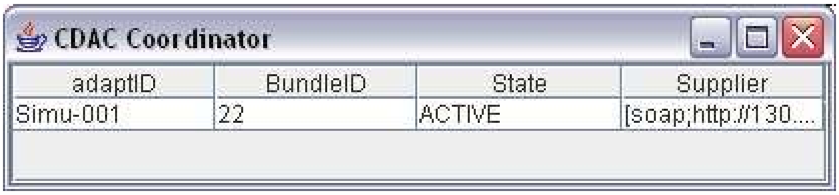
After an adapter supplier is discovered, the Discovery Coordinator retrieves the list of components provided by the adapter supplier, consisting of one or more combinations of registered context discovery adapters and monitors. *Figure 6-8* presents the GUI of the adapter supplier, showing the registered CMS and SimuContext adapter/monitor combinations and download URL's.



ID	Adapter URL
CMS-001	file:..\CDAC_CMSAdapter\bundle\CM...
Simu-001	file:..\CDAC_SimuContextAdapter\bu...

Figure 6-8 GUI of an adapter supplier offering two context discovery adapters/monitors.

The discovery coordinator downloads (using OSGi's component downloading capabilities via http or the file system) the adapters/monitors registered at the Adapter Supplier. It starts the monitor which checks the availability of the discovery mechanism. If the monitor successfully detects the context discovery mechanism, the coordinator loads and starts the adapter component and indicates the availability of the context discovery mechanism to the Discovery Coordinator. *Figure 6-9* presents the GUI of the discovery coordinator, which shows the detected and active SimuContext discovery adapter.



adaptID	BundleID	State	Supplier
Simu-001	22	ACTIVE	[soap;http://130....

Figure 6-9 GUI of the discovery coordinator.

The monitor keeps checking the availability of its context discovery mechanism. If they become unavailable, the monitor will inform the discovery coordinator which stops and unloads the relevant adapters (i.e.

using OSGi lifecycle capabilities). Next to the monitor, the discovery coordinator will continuously check for the availability of the adapter Supplier via a straightforward heartbeat mechanism that sends a periodic heartbeat signal and waits for an acknowledgment. If the supplier becomes unavailable (i.e. no acknowledgement is recieved), the coordinator will respond by stopping the inactive adapters/monitor belonging to the supplier that disappeared.

Performance Tests

Table 6-5 presents the results of some performance tests. These tests are done to get an impression of the time spent for the (i) discovery of adapter suppliers, (ii) retrieval of the list of available discovery adapters and, downloading and registration of a discovery adapter, and (iii) the overall process of a new adapter becoming available. The test includes a sequence of 250 iterations in a situation where (i) an adapter supplier, offering the SimuContext adapter, is locally available on the host that runs the interoperability mechanism and (ii) an adapter supplier, offering the SimuContext adapter, is remotely available somewhere in the network of the host that runs the interoperability mechanism.

Table 6-4 Performance measures.

Description	Local supplier	Remote supplier
i) Discovery of an adapter supplier	Low: 1,0s Avg: 3,1s High: 5,1s	Low: 1,0s Avg: 3,2s High, 5,0s
ii) Retrieving a list of available adapters and , downloading and registering of an adapter	Low: 0,5s Avg: 0,5s High: 0,8s	Low: 0,6s Avg: 0,7s High: 2,1s
iii) Overall process	Low:1,6s Avg: 3,7s High: 5,7s	Low: 1,7s Avg: 3,9s High: 5,9s

The average time spend to discover an available local/remote discovery adapter supplier is respectively 3,1s and 3,2s. The size of this measure consists of multiple parts. First, the discovery coordinator is configured to check for new adapter suppliers every 4 seconds. Hence, on average, the expected waiting time for detection of a newly available adapter supplier is 2 seconds. Secondly, the discovery of new suppliers is done using the WS discovery mechanism, which is configured to execute a discovery session for exactly 1 second. Hence, the minimum for discovery of an adapter supplier is 1sec while the average lies around 3 sec. The time spent to retrieve a list of adapters and downloading/registering adapters is approximately between 0,5 and 2 seconds.

The average time for both measurements is about 3,7 seconds for a local supplier, and 3,9 seconds for a remote supplier Hence, on average, an application using the discovery interoperability mechanism should be able to use a newly available context discovery mechanism within 4 seconds upon accessing the network of an adapter supplier.

Especially, for the discovery of adapter suppliers, some configuration values have been estimated, such as the supplier discovery repetition rate (i.e. discovery every 4 seconds) and the discovery time (i.e. 1 seconds). These values have not been optimized and future research could be done to possibly decrease the overhead of the discovery of adapter suppliers.

6.1.4 Implementing a SimuContext Discovery Adapter

To enable context discovery mechanisms to be used by the context discovery interoperability mechanism, a monitor and adapter have to be developed. Additionally, this adapter and monitor have to be registered at a deployed adapter supplier.

Since a large part of the functionality of the monitor is equal for every type of context discovery mechanism, a new one can be implemented by deriving from a reference monitor component and adapting some parts for the specific needs of the targeted context discovery mechanism. The specific context discovery adapters are less generic than the monitor, and should at least implement the IDiscoveryAdapter interface. The discovery coordinator and adapter supplier are generic and do not have to be (re-) implemented for new context discovery mechanisms. However, the adapter supplier has to be configured with the appropriate information for the context discovery mechanism it supplies adapters (i.e. Unique ID and URLs of the monitor and adapter).

In this section, we discuss how-to create a discovery adapter and monitor. We do this by explaining the development of the SimuContext discovery adapter (i.e. consists of adapter and monitor capabilities) that can interact with the SimuContext framework (see Section 6.2 for more details on the SimuContext framework). We believe this implementation can act as a reference implementation for other adapters/monitors.

Implementing a discovery adapter/monitor requires the creation of a separate component (i.e. bundle in OSGi terms) that implements bundling, adapter and monitoring functionality. This is schematically shown in *Figure 6-10.* This component has to support the IDiscoveryAdapter interface by exposing it as an OSGi service. The ID and location (i.e. URL) of the packaged component can be registered to an adapter supplier, which in turn can advertise it to requesting context discovery interoperability mechanisms.

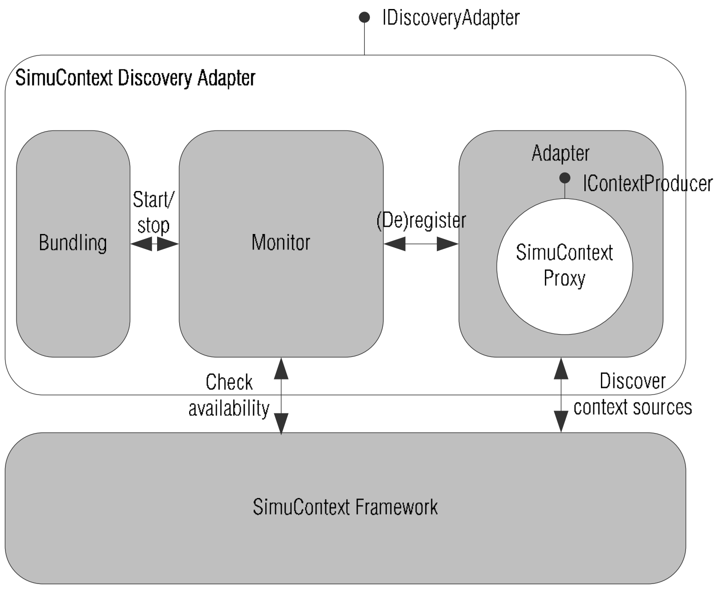


Figure 6-10 Structure of the SimuContext Discovery Adapter

The bundling functionality of a discovery adapter contains behaviour that is triggered by the OSGi container on start-up of the adapter component. On start-up, the bundling behaviour starts the monitor, which continuously checks the availability of the supported discovery mechanism. In case of SimuContext, this is done by checking if there is a component registered that exposes the SimuContextSource Retrieval service (see Section 6.2). When the discovery mechanism is available, the adapter functionality is started and the discovery adapter service, this adapter offers, is registered to the OSGi container.

The adapter functionality is responsible for transforming a generic discovery request to a discovery request suitable for the specific context discovery mechanism. This transformed request is send to the discovery mechanism on behalf of the application. The returned discovery result (i.e. references to context sources) is packaged in a proxy object (i.e. implementing the IContextProducer interface) which an application can use to retrieve context information. This proxy object translates the proposed generic context information data model to the specific context information data model of the underlying context discovery mechanism *Example 6-2* gives an example of such a transformation for a SimuContext adapter.

Example 6-2 implementation of the getContext() method in the SimuContext Proxy.

```
SimuContextSource cs;

/** @see IContextProducer
public ContextInfo getContext() {
    // Contextinfo is the generic datamodel of context information
    ContextInfo ci = new ContextInfo();

    // Translate the datamodels
    ci.setElement(cs.getName());
    ci.setEntity(cs.getEntity());
    ci.setFormat(cs.getFormat());
    ci.setContextSample(cs.getContext().getValue());

    return ci;
}
```

We implemented the SimuContext Discovery adapter as a Java-based OSGi component which contains about 270 lines of code. The size of the adapter component is approximately 11kB.

6.1.5 Integration of the Context Discovery Interoperability Mechanism in CACI

The context discovery interoperability mechanism is an integral part of the CACI infrastructure. It is used by the binding mechanism (see Chapter 5) to discover context sources from which a suitable one is selected to create a context binding. Hence, part of the discovery coordinator functionality is integrated with the context binding discovery manager which is part of the context binding mechanism. The discovery manager implements a listener for incoming discovery services offered by the adapter. *Example 6-3* gives an example of such a listener.

On registration of a discovery adapter service to the OSGi environment (i.e. OSGi triggers the serviceChanged() method in the listener) the reference to this service is added to the list of usable discovery adapters. The discovery interoperability mechanism is still responsible for detecting discovery adapter suppliers, downloading discovery adapters and registering their discovery adapter service to the OSGi environment.

Example 6-3 listener for incoming discovery adapter services.

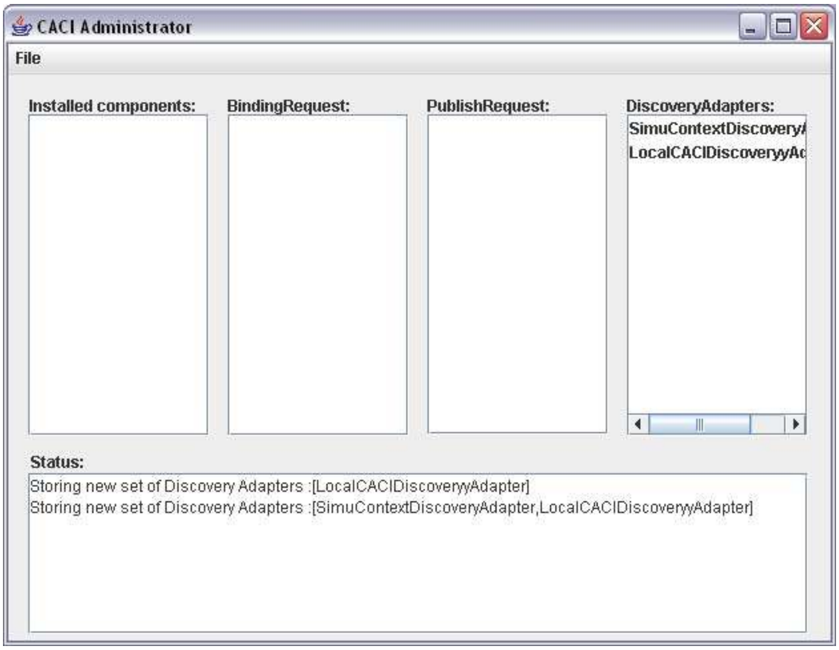
```
DiscoveryManager manager;

public void serviceChanged(ServiceEvent arg0) {
    String[] objCl = (String[]) arg0.getServiceReference().
    getProperty(org.osgi.framework.Constants.OBJECTCLASS);

    // A received event can be for a registering and unregistering discovery adapter
    if(arg0.getType() == ServiceEvent.REGISTERED){
        if(objCl.length>0&&objCl[0]==IDiscoveryAdapter.class.getName()){
                IDiscoveryAdapter          adp          =          (IDiscoveryAdapter)
bc.getService(arg0.getServiceReference());
                // add new discovery adapter to the usable set of discovery adapters
                manager.addDiscoveryAdapter(adp);
        }
    }else if(arg0.getType() == ServiceEvent.UNREGISTERING){
        if(objCl.length>0&&objCl[0]==IDiscoveryAdapter.class.getName()){
                IDiscoveryAdapter          adp          =          (IDiscoveryAdapter)
bc.getService(arg0.getServiceReference());
                // remove new discovery adapter to usable set of discovery adapters
                manager.removeDiscoveryAdapter(adp);
        }
    }
}
```

By putting part of the discovery coordinator functions inside the context binding mechanism. The remaining functions of the discovery coordinator can be optionally omitted (i.e. detection of adapter suppliers and downloading/registration of adapters). In this way also the adapter functionality can be used manually by deploying an adapter, which exposes discovery adapter service, inside an OSGi environment running CACI. The discovery adapter listener triggers on a registration event and adds the manually added adapter to the set of available discovery adapters. *Figure 6-11* presents the GUI of the CACI administrator where the available discovery adapters are listed.

Additionally, the context discovery mechanism can be used independently from the CACI infrastructure. Context-aware application can issue discovery request using the discovery service offered by the discovery coordinator.



```
Storing new set of Discovery Adapters :[LocalCACIDiscoveryyAdapter]
Storing new set of Discovery Adapters :[SimuContextDiscoveryAdapter,LocalCACIDiscoveryyAdapter]
```

Figure 6-11 CACI administrator gui showing two registered discovery adapters.

6.1.6 Related Work

Several initiatives exist to let context discovery mechanisms collaborate. First, there are mechanisms that federate multiple instances of one type of context discovery mechanism. For example context discovery mechanisms used in the mobile operators domain (Roussaki, Strimpakou et al. 2006), context discovery mechanism used for small to mid-sized environments (Hesselman, Eertink et al. 2007), or a combination thereof (José, Meneses et al. 2005). In this type of approaches, the interoperability function consists mainly of coordination rather then homogenizing or bridging heterogeneous context discovery mechanisms.

When focussing on interoperating heterogeneous context discovery mechanisms, several approaches exist that homogenize context information by proposing a homogenizing interoperability layer that uses a common context model (e.g. ontology) to uniformly exchange context information. For example, the ITransIT system (Meier, Harrington et al. 2006; Lee and Meier 2007) is built to integrate advanced transportation systems that use spatial context information (e.g. location, driving status). They propose the ITransIT tier, which offers a homogenizing layer that handles spatial context information coming from legacy systems. Hence, they propose a common spatial data model (PCM) and an ontology (PCOnt) to specify the spatial context information. (Blackstock, Lea et al. 2006) propose a common ubiquitous environment model and a homogenizing layer called the

Ubicomp Integration Framework (UIF).UIF is implemented using semantic web technology to adapt existing ubiquitous computing systems to this common model. Adapters are deployed to map legacy systems to their proposed common context information model.

When considering bridging approaches, (Lehmann, Bauer et al. 2004) integrate a context discovery mechanism for home environments (the Aware Home Spatial Service) with a context discovery mechanism for mobile operators. They focus mainly on integrating the data models used by the two types of context discovery mechanism. Contrary, (Hesselman, Benz et al. 2008) focuses on the functionality required for interoperating context discovery mechanism.

The discussed approaches focus mainly on (i) so-called semantic interoperability by proposing common context information models or (ii) on the functionality required to interoperate statically available context discovery mechanisms. Our discovery interoperability approach can be considered complementary with respect to semantic interoperability. We consider this future work and hence we could benefit from their insights. With respect to the second point, we take context discovery interoperability one step further by taking into account the dynamic availability of different context discovery mechanism.

6.1.7 Limitations and Future Work

We acknowledge some aspects that are not covered by the proposed context discovery interoperability mechanism and which we consider future research:

– *Security:* downloading ‘unknown’ code (i.e. adapters/monitors) is considered a security risk. However, mechanisms exist to overcome this issue, such as code signing and firewalling (Rubin and Geer 1998). However, more research is required to assess the security threats of deploying a context discovery interoperability mechanism.
– *Semantic interoperability:* the proposed mechanism focuses on functional and syntactic interoperability. However, interoperating different information models used by context discovery mechanisms is similarly important. Mechanisms exist to get semantic interoperability, which could be used to extend the current mechanism (e.g. shared ontologies (Blackstock, Lea et al. 2006)).
– *Performance optimization:* in the prototype, some configuration values have been estimated. Further measurements are required to optimize these values to decrease the overhead introduced by the discovery interoperability mechanism.

6.2 The SimuContext Framework

This section discusses the SimuContext framework. This is a framework to simulate context sources. It starts with a problem analysis, followed by a description on the design and prototype implementation. Consecutively, it discusses the usage of this mechanism in CACI, related work and, limitations and future work.

6.2.1 Problem Analysis

Developing context-aware applications includes creating application logic and context logic. CACI simplifies the creation of context logic. Nevertheless, for testing and demonstration of the application logic, context sources that can deliver the needed context information, are required. However, testing and demonstrating context-aware applications in a controllable and reproducible way with real context sources may prove extremely difficult. By nature, context information is highly dynamic (i.e. dynamic in value and quality). Therefore, retrieving the same context in similar situations is hard. For example, when using a GPS, standing in the same location can result in different context values over time due to changing accuracy. Also practically, it is hard to use life context information during tests or demonstrations. For example, GPS does not work inside buildings, which means that tests and demonstrations have to take place outside.

Additionally, in the testing environment of the context-aware application the context sources that have to be used are often not available. Hence, testing and demonstration of context-aware applications may include a significant extra amount of development effort to implement substituting testing/demonstration context sources or installation of the real context sources.

Ideally, application developers want to abstract from the internals of context sources and treat them as a black box. Their effort should be spend in the development of the application logic rather then in creating context sources. Hence, we propose to simulate context sources for rapid testing and demonstration of context-aware applications. For this we develop a generic context source simulation framework called SimuContext. Simulation in general provides many benefits for software development like, cost reduction, improve reliability and shorten development time (Christie 1999). The SimuContext framework facilitates application developers in testing and demonstrating context-aware applications by enabling them to specify the behaviour of context sources and using simulated context sources (i.e. that expose the specified behaviour) as inputs to their Context-aware application.

Bylund (Bylund and Espinoza 2002) distinguishes two types of context source simulation tools: (i) simulation programming suites and (ii) semi-realistic simulation environments. With the first type, application developers specify and/or program a simulated context source which produces the simulated context information to the context-aware application. With the second type, context-aware applications retrieve context information from a virtual reality world in which the application developer virtually moves the application/user (e.g. location information of a moving user in a 3D world). However, both types of tools can also be combined.

The SimuContext framework has the characteristics of a simulation programming suite and offers a generic simulation facility which can provide a configurable set of context sources that can be tailored to a specific context-aware application scenario.

6.2.2 Design

This section discusses the design of the SimuContext framework. It presents the requirements, a high-level overview and a functional decomposition of the SimuContext framework.

Requirements

The proposed context source simulation framework should support the following generic non-functional requirements:

- *Generality:* the framework should be generic enough to simulate a broad range of context sources. Furthermore, the framework should not pose severe constraints on the target application, and therefore it should be reusable for multiple applications.
- *Extensibility:* it should be easy to extend the framework to support specific context sources for specific context-aware application scenarios.

To create an accurate and realistic simulation framework for context sources, we have used our basic taxonomy of context information as presented in *Figure 2-6*. We distinguish that context consists of information on several levels of abstraction. First, context has meta-information on its quality, which is called quality of context (QoC). Second, context encapsulates information related to what it describes (relation information). This indicates that context is always related to an entity. An entity can be a human, physical, or computational object. Finally, context information encapsulates the real information. This consists of an element, describing the context identifier (e.g. Location). Then it has a value, for example 52.123/6.23123. Finally, it has a format that describes the value, for example Lat/Long. Taking these aspects together, this leads to the following functional requirement:

- The SimuContext framework should support the aspects that are encapsulated by the defined context information taxonomy. This includes QoC, the entity to which it is related and the elements that describe the contextual information (i.e. element, value, format).

Additionally, context information provisioning has some other characteristics (Broens 2004; Bunningen, Feng et al. 2005): (i) Context information is temporal and changes over time, (ii) context is spatial and changes when the entity is moving, (iii) context is imperfect, and (iv) context sources are often distributed. These characteristics trigger the following requirements for our framework to realistically simulate context sources:

- Due to its temporal and spatial nature, context information is subject to continual changes. Therefore, the simulated context source should be able to have changing values specified in an application specific, pluggable value model. Hence this value model should be extendible and easy to plug into the framework.
- Furthermore, to facilitate the user to simulate realistic context sources, our framework should support the two common types of invocation mechanisms: (i) Request – Response and (ii) Subscribe-Notify.
- To provide the Subscribe-Notify mechanism, our framework should support an event model that specifies when context change events should be generated. This event model should be extendible and easy to plug into the framework.
- Due to the imperfect nature of context information, it inherently has quality properties. Our framework should be able to express this in a QoC model. The realization of the QoC model depends on the target application therefore the QoC model should be plugable and extendible.
- The quality values of context are related to the provided context information. As this is subject to change the quality values are also subject to change. Our framework should support changing QoC values correlating with the values from the value model.

High-level Overview

Context-aware applications developers can use the SimuContext framework in two ways: (i) as a class library that is tightly coupled with the application, and (ii) as a service-oriented run-time middleware mechanism. We discuss the architecture of SimuContext from the second perspective, as this is the manner SimuContext is used in the CACI infrastructure. However, using SimuContext as a class library mainly involves direct method invocation of the (service) interfaces. *Figure 6-12* presents a high-level overview of the SimuContext framework.

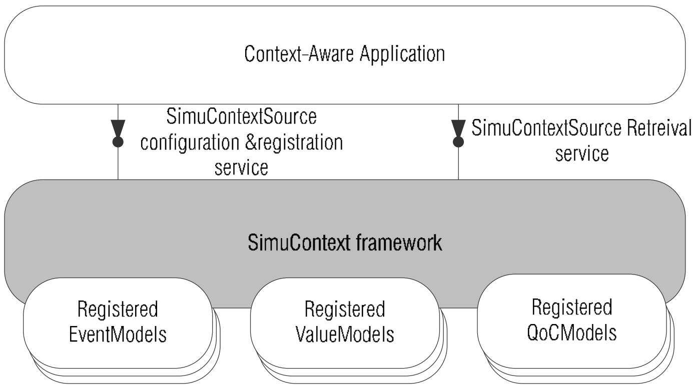

Context-Aware Application
SimuContextSource configuration &registration service
SimuContextSource Retreival service
SimuContext framework
Registered EventModels
Registered ValueModels
Registered QoCModels

Figure 6-12 High level architecture of the SimuContext architecture.

A context-aware application (i.e. service user) can interact with the SimuContext framework (i.e. service provider) using two services:

– *SimuContextSource configuration & registration service:* this service can be used to configure context sources and register them to the SimuContext repository. Additionally, it can be used to register application specific value, event and QoC models. *Table 6-6* gives the service primitives of this service.
– *SimuContextSource retrieval service:* an application can use this service to retrieve a configured context source, or discover a suitable SimuContext context source from the repository. Additionally, it can register SimuContext sources in the repository that are already configured using SimuContext as a class library. *Table 6-7* gives the service primitives of this service.

Table 6-5 Service primitives of the SimuContextSource configuration & registration service.

Direction	S/A	SP identifier	Parameters	ReturnParameters
SU-SPr	S	configSimuContextSource	SimuContextSource specification	ID
SU-SPr	S	registerValueModel	ValueModel	-
SU-SPr	S	registerEentModel	EventModel	-
SU-SPr	S	registerQoCModel	QoCModel	-

Table 6-6 Service primitives of the SimuContextSource retrieval serv ice.

Direction	S/A	SP identifier	Parameters	ReturnParameters
SU-SPr	S	getSimuContextSource	ID	SimuContextSource
SU-SPr	S	discoverySimuContextSou rces	SimuDiscoveryRequest	SimuContextSources
SU-SPr	S	registerContextSource	SimuContextSource	ID
SU-SPr	S	deregisterContextSource	ID	-

Functional Decomposition

Figure 6-13 depicts a functional decomposition of the SimuContext framework. In the configuration phase, an application developer specifies a context source it wants to simulate. The *configurator* parses the SimuContext source specification and instantiates a new SimuContext source. This includes that the configurator retrieves and instantiates the required event, value and QoC models from the model repository. The instantiated source is registered to the *SimuContextSource repository*. When an application developer registers an event, value or QoC model, this model is stored in the *model repository*.

In the operational phase, a context-aware application can retrieve simulated context sources (i.e. SimuContextSources) stored in the *SimuContextSource repository* via the *discovery manager*. When a context source is registered, the discovery manager stores this source in the *repository* and makes it available for discovery.

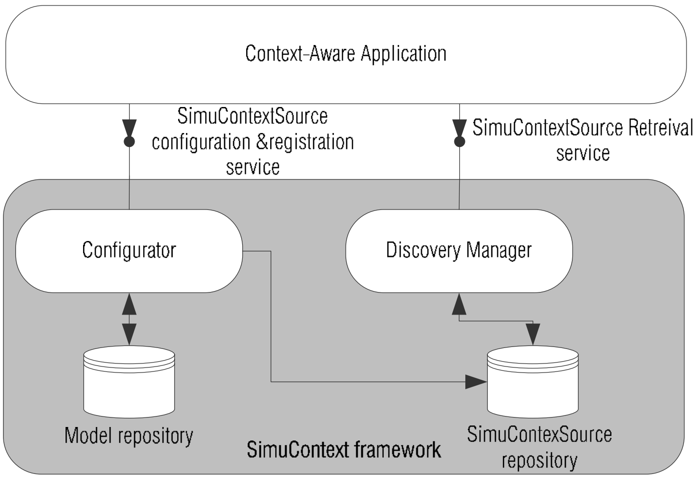

Context-Aware Application
SimuContextSource configuration &registration service
SimuContextSource Retreival service
Configurator
Discovery Manager
Model repository
SimuContext framework
SimuContexSource repository

Figure 6-13 Detailed architecture of the SimuContext framework.

A user can use a retrieved SimuContext source by invoking methods from the *ISimuContextSource* interface. This interface exposes the required two interaction mechanism: request-response (i.e. getContext()) and subscribe-notify (i.e. subscribe(), notify()). For the subscribe-notify mechanism, the user is required to provide, on registration, a callback object consistent with the *ISubCallback* interface. *Figure 6-14* gives a class diagram of these interfaces.

Context information is exchanged using *Context* objects. These objects contain the information as specified in our context taxonomy (see *Figure 2-6*). Additionally, it provides simulated *QoC* values for the QoC parameters identified by (Sheikh, Wegdam et al. 2007).

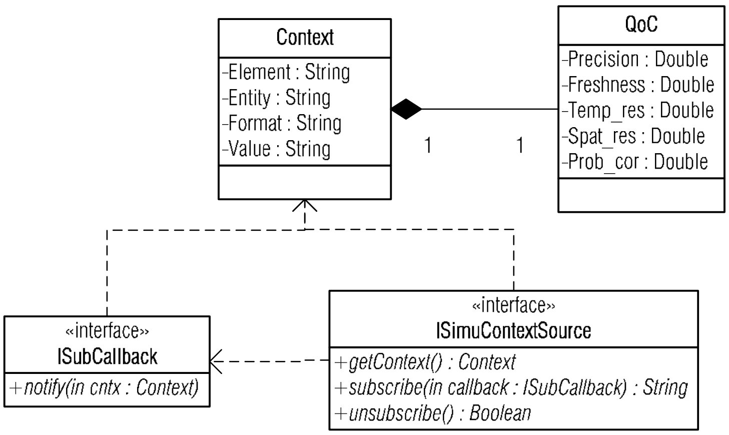


Figure 6-14 ISimuContextSource and ISubCallback interface and related data objects.

The internal structure of a SimuContextSource consists of several function blocks. *Figure 6-15* presents the internal structure of a SimuContext source. The *ServiceModel* implements the *ISimuContextSource* interface and is responsible for the interaction with the user. The *ValueModel* implements the value generator that produces the values of the context samples. The *EventModel* implements the event generator that produces the events when the subscribed user to this context source should be notified. The *QoCModel* generates the quality values of the delivered context sample. Both the ValueModel and the EventModel can be extended with application specific models. For example, we implemented a "RandomValueModel" that generates random values when context information is requested, or a "PeriodicEventModel" that generates every user-specified period an event to subscribers.

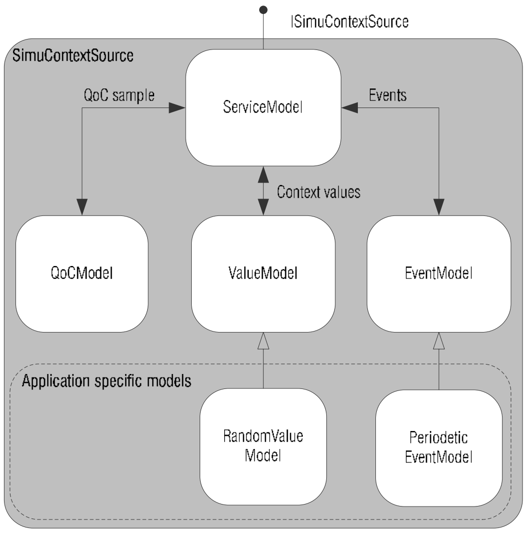


Figure 6-15 Internal architecture of a SimuContext source.

6.2.3 Implementation

The prototype of the SimuContext framework is implemented in Java and bundled as an OSGi bundle. The framework consists of approximately 2500 lines of code and the bundle has a size of approximately 75kB. The SimuContext framework is tested both on a laptop PC and a windows mobile PDA.

A user of the framework creates a SimuContext source by providing configuration information to the framework. *Example 6-4* gives an example of configuration information for a 'speed' context source. For pragmatic reasons the configuration file is a standard property file (i.e. 'property' = 'value'). This configuration file contains the basic information of the simulated context source like element, entity and format. Additionally, it contains id's of the specialized value, event and QoC models that should be used when instantiating the SimuContextSource. After the ':' possible parameters required for these models can be specified.

Example 6-4 SimuContext configuration file of a simulated 'Speed' context source.

```
id = CS1
name = Speed
format = km/hr
entity = Tom
valuemodel = nl.utwente.SimuContext.ValueModels.RandomValueModel:140
eventmodel = nl.utwente.SimuContext.EventModels.RandomEventModel:2
qocmodel = nl.utwente.SimuContext.QoCModels.MyQoCModel
```

Some pre-defined event and value models, ready to be used by the user, have been created. The following event models have been created:

- *RandomEventmodel:* triggers notification events randomly during a specified interval.
- *PeriodicEventModel:* triggers a notification event every specified interval.
- *GUIEventModel:* triggers a notification event when a user pushes the notification button.

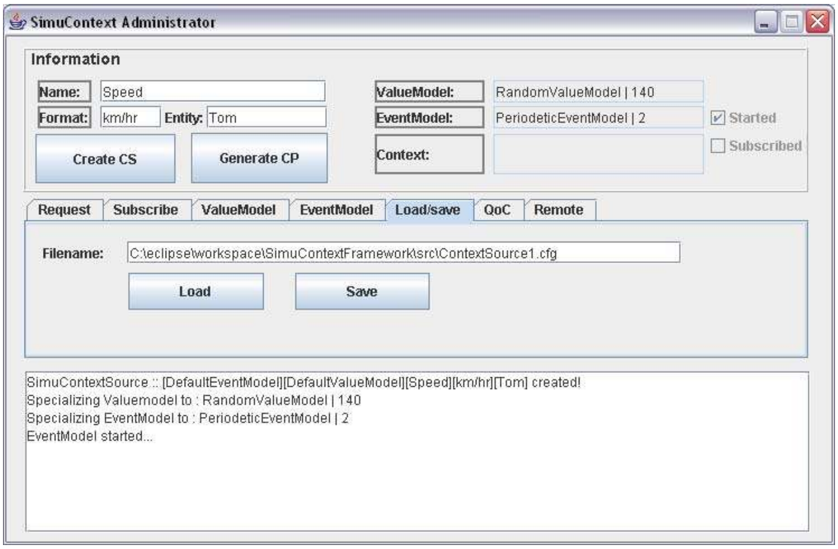
The following value models have been created:

- *RandomValueModel:* returns a random value between a specified min-max range.
- *CounterValueModel:* returns an incremental value between a specified min-max range.
- *GUIValueModel:* returns the value a user has inputted.

For testing purposes, we created two graphical interfaces (GUI):

- *SimuContext Configurator:* GUI (see *Figure 6-15*) to easily configure and test a SimuContextSource and save configuration files. Additionally, it can be used to configure a SimuContext source and automatically create a simulated context producer component that is deployed in the CACI infrastructure.

– _SimuContext Repository:_ GUI (see *Figure 6-16*) to get an overview of the SimuContext sources registered in the repository and to add or remove SimuContext sources based on configuration files. Additionally, SimuContext sources can be enabled/disabled to simulate appearing and disappearing of context sources.



SimuContext Administrator

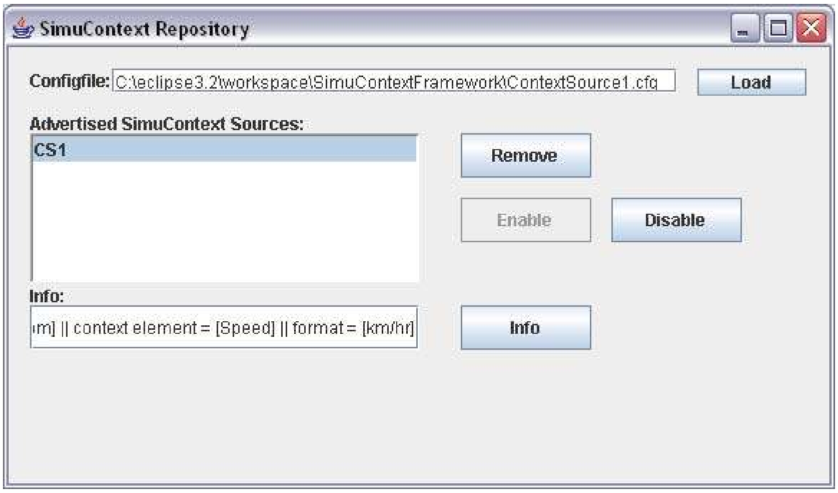
Information

Name: Speed
Format: km/hr Entity: Tom
ValueModel: RandomValueModel | 140
EventModel: PeriodeticEventModel | 2
Context:
☑ Started
☐ Subscribed

Filename: C:\eclipse\workspace\SimuContextFramework\src\ContextSource1.cfg

```
SimuContextSource :: [DefaultEventModel][DefaultValueModel][Speed][km/hr][Tom] created!
Specializing Valuemodel to : RandomValueModel | 140
Specializing EventModel to : PeriodeticEventModel | 2
EventModel started...
```

Figure 6-16 SimuContext Configurator GUI



SimuContext Repository

Configfile: C:\eclipse3.2\workspace\SimuContextFramework\ContextSource1.cfg

Advertised SimuContext Sources:
CS1

Info:
ım] || context element = [Speed] || format = [km/hr]

Figure 6-17 SimuContext Repository GUI

6.2.4 CACI and the SimuContext framework

In Section 4.5.2, we present the envisioned development trajectory of context-aware applications that benefit from the proposed CACI infrastructure and the SimuContext framework. SimuContext interacts with CACI in two ways:

– SimuContext is used as an underlying context discovery mechanism to discover context sources that can deliver the required context information. The SimuContext repository offers context information discovery capabilities to users. In the case of CACI, this means a discovery adapter is created to plug-into the context discovery interoperability mechanism (see Section 6.1).
– SimuContext is used to generate and automatically deploy simulated context source components (i.e. context producers) based on the CBDL specification of a context-aware application (i.e. context consumer). We discuss this type of use of SimuContext in some more detail in the next section.

Context Producer Generation

To enable application developers to test their application logic, we use SimuContext to automatically deploy simulated context sources that match with the context requirements of the application (i.e. specified in their CBDL description). Hence, we enable application developers to annotate their CBDL description with SimuContext information. *Example 6-5* presents an annotated CBDL document for a 'speed' context information requirement.

Example 6-5 Extending CBDL with SimuContext configuration information.

```
<CBDLDocument UserID="UserTom" ApplicationID="TomApp">
<ContextRequirement ContextRequirementID="TestBundle-Location">
    <Element>Speed</Element>
    <Entity>Tom</Entity>
    <Format>km/hr</Format>
    <SimuContext>
        <Valuemodel>
        nl.utwente.SimuContext.ValueModels.RandomValueModel:140
        </Valuemodel>
        <Eventmodel>
        nl.utwente.SimuContext.EventModels.RandomEventModel:2
        </Eventmodel>
    </SimuContext>
</ContextRequirement>
</CBDLDocument>
```

Information that is required to be added to a CBDL document to enable the deployment of simulated context sources is which value and event model (i.e. including the required parameters) the simulated context source should have. As part of the deployment phase of the context-aware application, the parser parses the CBDL document and also extracts the provided SimuContext information. When SimuContext information is available, the deployer retrieves a reference to the ISimuContextConfigurator service and invokes the configSimuContextSource() method with the information from the context requirement and added SimuContext information. Consequently, the SimuContext framework creates a matching SimuContextSource that is added to the SimuContext repository. This new context source event is notified to the binding mechanism that creates a new context producer proxy to bind the deploying application with the generated context source.

6.2.5 Related Work

Several initiatives aim to facilitate application developers in coping with real context sources (also see Chapter 3), like the Context Toolkit (Dey and Abowd 2000), JCAF (Bardram 2005) and PACE (Henricksen, Indulska et al. 2005). However, there also exist several simulation tools. Following Bylund's (Bylund and Espinoza 2002) categorization, several semi-realistic simulation environments exist of which we will give some examples in this section. However, to our knowledge, no other context simulation suites exist.

Bylund (Bylund and Espinoza 2002) discusses a tool that interactively simulates context information in real-time. Their tool, called QuakeSim, uses the popular game engine of Quake III Arena to simulate a 3D environment. In this environment virtual persons can move and interact with other persons or the environment itself. The game engine provides the context information of these virtual persons which can be used as simulated information for context-aware applications.

UbiWise (Barton and V. 2002) uses similar technology as QuakeSim. It simulates a 3D environment to simulate ubiquitous environments which include prototyping of new devices and protocols. The simulator focuses mainly on computation and communication devices.

3DSim (Shirehjini and Klar 2005) provides a tool for rapid prototyping Ambient Intelligent applications. It uses a 3D based virtual environment to represent ambient devices. Context events are passed to the system with a TCP-based eventing interface.

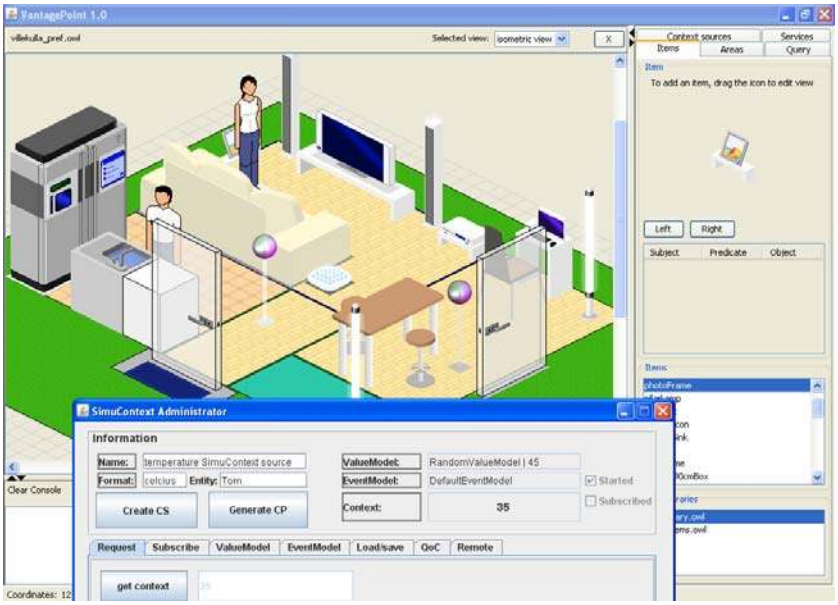
Morla (Morla and Davies 2004) discusses a simulation environment for location-based systems. They focus on component interaction, networking and location changes. Their environment supports the distribution of context events generated by distributed simulators using Web Services.

In contrast to the previous initiatives, SimuContext is a context simulation suite that enables the user to specify the behavior of context sources instead of simulating an environment where the context is inferred from. In simulation environments, context changes are produced by interaction of a user with the environment (e.g. movement). SimuContext can be less attractive for live demonstrations (i.e. not a 3D GUI), however the simulated context is better controllable and reproducible. Additionally, testing and validation in an automated manner is more convenient. Furthermore, SimuContext is a context-centric approach while some of the related approaches focus on pervasive device and network aspects and use/provide context as a side-effect. SimuContext offers a generic light-weight approach that focuses on context simulation which is based on a robust context model.

6.2.6 Limitations and Future Work

We acknowledge some aspects that are not covered by the proposed SimuContext framework, which we consider future research:

– *Integration with semi-realistic simulation environments:* integration of the SimuContext simulation programming suite with a semi-realistic simulation environment could be beneficial. Semi-realistic simulation environments are useful for attractive application demonstrations while programming suites are suited for controlled testing of applications. First steps in this direction have been taken to integrate SimuContext with Vantagepoint. Vantagepoint (Niskanen, Kalaoja et al. 2007) is a 3D environment which can be used to graphically create a semantic description of home environments. SimuContext sources (i.e. hovering globes) can be added and configured using the SimuContext Administrator. Vantagepoint exports the SimuContext retrieval service to applications. Hence, application developers can create a semantic description of the environment in which their application is going to function additional to specifying available context sources and really simulating them. *Figure 6-18* presents the GUI's of Vantagepoint and SimuContext. The orbs in the 'home' represent SimuContext sources.



Name: temperature SimuContext source
Format: celcius Entity: Tom
ValueModel: RandomValueModel | 45
EventModel: DefaultEventModel
Context: 35

Figure 6-18 Integration of SimuContext with Vantagepoint

– <u>*Simulating QoC:*</u> the design of the SimuContext framework consists of SimuContext sources that have a value, event and QoC model. The value and event model are implemented in the prototype. The prototype contains a framework for the QoC model and is hence prepared for simulating QoC parameters. However, the prototype should be extended with an implementation of the QoC model to actually simulate QoC samples.

Chapter 7

Telemedicine and Context-Awareness

This chapter gives an overview of the (electronic) healthcare domain, especially focussing on the telemedicine sub-domain. Additionally, it discusses determinants that influence the success of telemedicine applications. Finally, it discusses how context information can potentially be beneficial for telemedicine applications. Parts of this chapter are published in (Broens 2005; Broens, Huis in't Veld et al. 2007).

This chapter is structured as follows: Section 7.1 presents the background of (electronic) healthcare and discusses the increasing influence of ICT. Section 7.2 gives an overview of the telemedicine domain and the social-economical trends stimulating the development of telemedicine applications. Section 7.3 presents determinants that influence the success of telemedicine applications. Section 7.4 discusses the nature and structure of telemedicine applications. Section 7.5 discusses some current context-aware telemedicine applications. Finally, Section 7.6 reflects on how context-awareness may improve the quality of telemedicine applications.

7.1 Background on (Electronic) Healthcare

Healthcare (also called medicine) is intrinsic to human existence. Humanity has always been in need of solutions to various health related issues, such as childbirth and cure for diseases. Merriam-Webster's dictionary defines **healthcare** as "efforts made to maintain or restore health especially by trained and licensed professionals". Aspects tackled in healthcare are, amongst others, surgery, psychology and dietetics.

In early history, diseases were accounted to gods, demons and spirits. Therefore, healthcare was, at that time, a spiritual occupation performed by priests or witch doctors. In the time of Homer and Hippocrates, healthcare increasingly became a science. In the medieval and renaissance period, a combination of spirituality and science dominated healthcare. Because of the lack of knowledge about the human anatomy, healthcare focused on diets and hygiene, instead on surgery and medicines. 'Modern' healthcare is mostly based on science. The knowledge on the human anatomy has expanded exponentially, resulting in higher quality healthcare. This is shown in an enormous increase of the life expectancy[9]. Currently, healthcare is moving towards a more holistic approach in which a complete treatment of the patient is preferred over just treating the physical symptoms of the disease. Therefore, current healthcare is a complex domain in which surgery, medicine, psychology, dietetics etc., and a combination of them, play an important role (Margotta 2001).

ICT in Healthcare

In the last couple of years, Information and Communication Technology (ICT) plays an increasingly important role in healthcare. The introduction of ICT in this domain, was recognized as a valuable development to improve and enhance the healthcare provisioning process (Berg 1999; Pattichis, Kyriacou et al. 2002; Philips Medical Systems 2003). Generally, applying ICT in the healthcare process is envisioned to improve the quality and productivity of this process with similar or lower costs. More specifically, envisioned improvements of applying ICT in the healthcare process compared to traditional healthcare are:

– *Efficiency*: healthcare processes can be automated in such a way that information is processed, transferred and made available more easily to multiple domains. This can improve the efficiency of the healthcare process. For example, using electronic transfer forms for transferring of patients to different health institutions.
– *Precision*: information is stored and processed by computers which are less prone to errors by bad handwriting or misinterpretations. For example, by using an electronic patient record to administer patient information.
– *Cost savings*: ICT solutions can take over expensive human processes. For instance, ICT application can be used to decrease administrative overhead or the number of unnecessary house calls etc.

The majority of traditional healthcare disciplines use ICT in their healthcare provisioning process. This aspect of healthcare is denoted as electronic healthcare (**E-health**) (Oh, Rizo et al. 2005). For example, disciplines like surgery, psychology and dietetics use e-health technology such as electronic

[9] The life expectancy of US citizens in 1900 was 47 which increased to 77 in 2002 (source: National Centre for Health Statistics). Life expectancy of Dutch citizens increased from 70,24 in 1950 to 79,06 in 2006 (source: CBS, http://www.cbs.nl).

patient records (EPD), hospital information systems (HIS) and telemedicine to facilitate their healthcare provisioning process. For example, when a patient with a cardiac arrest is admitted to the hospital, his patient data (e.g. name, blood type, allergies) is stored in the HIS using an EPD. During his treatment in the hospital, his vital signs are monitored and his EPD in the HIS is updated accordingly. In the aftercare phase, his EPD is electronically transferred to the dietetics ward where the care continues. Possibly, when the patient returned home, his vital signs are send to the hospital using a telemedicine system to be monitored.

7.2 An Overview of the Telemedicine Domain

Telemedicine is defined as providing healthcare and sharing of medical knowledge over distance using telecommunication means (Pattichis, Kyriacou et al. 2002). A common type of current telemedicine systems are systems that deploy assistive technology for aiding the elderly (Miskelly 2001). For example, the elderly alarm button system. This system enables an elderly person, when in trouble, to push the alarm button and to contact a call centre. The operator at this centre can then help the person in need.

Early telemedicine initiatives date back to the beginning of the 19th century, where Einthoven transferred ECG signals via telephone lines. In Norway and Sweden in the 1920's, telemedicine was applied to aid troubled seamen from the shore (using radio to give advice). In 1935, the International Radio-Medical Centre was founded which provides advice and assistance to seaman during medical emergencies. In 1955 the Nebraska Psychiatric Institute used closed-circuit television to provide care from the university medical centre to a state hospital over distance. A new boost to telemedicine was given by NASA in 1960's and 70's. They measured psychological parameters from the spacecraft and space suits during missions (Doarn, Ferguson et al. 1996). Digital transmission and compression (1980's) introduced a new generation of telemedicine, mostly based on point-to-point videoconferencing. Currently, due to improved technological capabilities, real-time 24/7 monitoring and treatment of patients over distance is feasible.

Telemedicine is often used to cure sick patient but may also be used to care for healthy persons (Meystre 2005). For example, telemedicine can improve training results of athletes, astronauts can be assisted in harsh environment, overweighed persons can be stimulated to reduce weight (i.e. improve wellness) and persons that regularly use computers can be notified of overuse to prevent RSI.

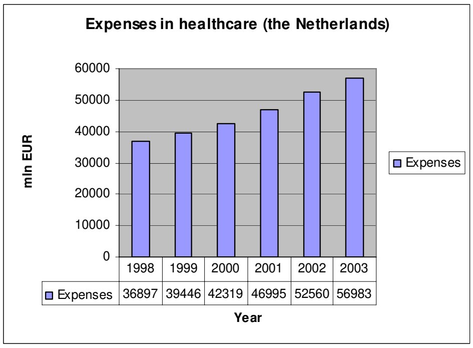
Although, telemedicine is recognized as a valuable improvement of the healthcare process, only recently technology has advanced in such a way that feasible advanced telemedicine systems can be developed (Meystre 2005). On the one hand we see the rise of high bandwidth mobile communication mechanisms (e.g., GPRS, UMTS) and on the other hand we see the miniaturization of high power mobile devices which offer increasing processing power, memory and storage capacity (e.g. PDAs, smartphones, laptops, smart clothes) (Marsh 2002). These trends enable the development of (near) real-time, high quality 24/7 mobile telemedicine systems with relative low costs.

Stimuli and Barriers for Introducing Telemedicine in Healthcare

As discussed, telemedicine has the potential for improving the healthcare process. Additionally, introduction of telemedicine application is stimulated by major social-economic developments (Dean 2004; The Telemedicine Alliance 2004):

– *Patient-centric healthcare:* For a long time, healthcare was government controlled. Now that patients become better informed, organized and educated, healthcare is shifting from offer- to demand-driven process. This requires flexibility in the healthcare provisioning process.
– *Cost savings and efficiency:* western society is aging. Currently, in Europe 16 to 18% of the population is over the age of 65. Estimations indicate that this rises to 25% in 2010 (The Telemedicine Alliance 2004). This increasing number of elderly results in an increasing number of potential healthcare consumers with a decreasing number of healthcare professionals. Furthermore, due to the standard of life in the western world there is an increasing amount of people suffering from chronic 'luxury diseases' like diabetics, vascular diseases and RSI. This result in increasing healthcare spending which can be already seen today. This is shown in *Figure 7-1*.

Figure 7-1 Healthcare spending in the Netherlands (source: CBS).



	1998	1999	2000	2001	2002	2003
▨ Expenses	36897	39446	42319	46995	52560	56983

– *Cross-domain integration:* To provide more efficient and cost effective patient-centric healthcare, it is recognized that healthcare must be organized as a value-chain that integrates multiple domains in healthcare (e.g. hospital – care institution – general practitioner). This requires a form of transparency between domains and mechanisms to effectively integrate them.

An example that supports the previous discussed developments is the current trend of increased extra-mural care compared to institutional care (Ross 2004). Patients are treated as long as possible in their home environment rather then in care institutions. When they are hospitalized, the period of stay in the institution is minimized and there is a longer process of post-care at home. This is both to save costs of hospitalization and to improve the patient's wellbeing by offering him care in his own environment.

Although the previously sketched developments, we also distinguish some high-level barriers in the healthcare domain that limit the acceptance of innovative telemedicine systems:

– *Conventional area:* healthcare is a conventional area where, traditionally, changes are not accepted quickly. For example, policy and legislation is not tailored to this novel type of applications.
– *Limited budget:* at this moment already cost and efficiency play an important role in healthcare. Therefore, there is a limited budget for introducing costly innovations.
– *Non-transparent domain:* currently, health organisations have a rather closed and individual nature. This makes the introduction of crosscutting inter-organisational systems hard.

Therefore, developing and introducing telemedicine systems is complex and challenging. In the next section, we elaborate more on aspects that influence the success of telemedicine systems.

7.3 Determinants Influencing the Success of Telemedicine Systems

Despite the previously discussed promising effects of telemedicine on future healthcare, many telemedicine systems do not survive the research phase or become a failure in daily practice (Tanriverdi and Iacono 1998). Berg (Berg 1999) shows that more than 75% of the developed telemedicine systems fail during the operational phase. In (Broens, Huis in't Veld et al. 2007), we analyze current telemedicine systems and identify determinants that influence the success of telemedicine systems.

We perform a qualitative literature study on 45 telemedicine articles published in the supplement of the Journal of Telemedicine and TeleCare (Wootton 2005). We consider this sample representative for telemedicine research in Europe. Two reviewers read all studies independently. The reviewed studies are qualitatively analysed on determinants that influence the implementation of the reviewed system, which may influence future implementation of these telemedicine systems. To generalize and classify the identified determinants, we employ the knowledge barriers categorization of Tanriverdi and Iacono (Tanriverdi and Iacono 1998). They identify the following knowledge barrier categories:

– *Technical:* technical expertise on how to develop, deploy, and use telemedicine systems.
– *Behavioural:* attitude of the involved stakeholders (e.g. patients, doctors) towards telemedicine systems.
– *Economical:* economical arrangements required for deploying a telemedicine system.
– *Organizational:* changes in medical practice and workflow due to usage of telemedicine systems.

Identified Determinants

Based on the theoretical model of Tanriverdi and Iacono (Tanriverdi and Iacono 1998), our study results in a more detailed classification of determinants that influence the success of telemedicine systems. Our classification consists of a top-level category that consists of determinants (see *Table 7-1*). Additionally, compared to Tanriverdi and Iacono, we introduced a top-level category on 'policy and legislation' and corresponding determinants.

Table 7-1 Comparison of determinant categorization.

Tanriverdi and Iacono (Tanriverdi and Iacono 1998)	Broens et al. (Broens, Huis in't Veld et al. 2007)
Technical	Technology - Support - Training - Usability - Quality
Behavioural	Acceptance - Attitude and usability - Evidence based medicine - Diffusion and dissemination
Economical	Financial - Provider and structure
Organizational	Organizational - Intramural and extramural work practices

	Policy and Legislation - Legislation and policy - Standardization - Security

We identify the following determinants in the following categories (for the literature justification see (Broens, Huis in't Veld et al. 2007)):

- **Technology:**
  - Support: The analysis shows that a major issue for technological acceptance of telemedicine systems is the availability of support to its users. This includes support for the deployment phase as well as the support throughout the operational phase. Support should be offered on the technical level on how to install and sustain the system but also on how to deal with errors and problem situations. Without support, problem situations during the use of the system lead to de-motivation and a high probability of abandoning the system.
  - Training: Training was also seen as an important requirement for the usage of telemedicine systems. Generally, users are not familiar with these new types of systems, which often include the use of difficult equipment. The analysed articles indicated that there is a need for training of users on how to use these novel types of systems. Such training is needed on all layers of the system: from the managers who interpret data, to doctors who view vital signs to nurses that have to administer the practical parts of the telemedicine system.
  - Usability: The analysis indicated that usability of the system is a major factor for success. Patients should be comfortable wearing novel kinds of (mobile) monitoring and treatment devices, which do not hinder them in their daily life. Supporting staff and doctors should be able to operate the devices and should have flexible access to services offered by the telemedicine system. Currently the information and the used modality are not tailored to the situation and skills of the patient and medical personnel.
  - Quality: Technical problems showed as being a major barrier for successful implementation of telemedicine systems. Technical problems consisted of non-connecting or malfunctioning devices, power loss, cable breakings etc. There is a need for robust systems and their supporting infrastructures, which can scale from the pilot phase to a real-life operational situation. Poor technical feasibility often results in distrust of end-users and low levels of satisfaction.
- **Acceptance:**
  - Attitude and usability: Results of the analysis show that technology acceptance of both patients and professionals are influenced

considerably by the patient's and professional's attitude towards telemedicine technology. In more detail, involvement of patients and professionals in the requirements analysis and the design process is crucial in order to understand how to fit telemedicine into their daily work practices. Feelings of ownership, enjoyment, self-efficacy, and feelings of pride could be augmented by involving end-users in the early stages of the developmental process. Another frequently reported aspect in relation to acceptance is to communicate meaningful (correct, relevant and up to date) information and ideally personalize this information, especially for professionals. They should be able to possess the right patient information at the right time. Previous experience of patients and professionals with computers and associated computer skills should be taken into account in developing a telemedicine service as well as level of education and age.

  - *Evidence Based Medicine:* Among several studies, evidence-based medicine is regarded to be a requirement for acceptation of a new drug or treatment. It is recommended to apply the methodology with the highest quality, which is considered to be the randomized controlled trial (RCT). Results of the present analysis show that alternative designs are needed to evaluate the efficacy of telemedicine systems and to convince professionals, policy makers, and insurance companies of the usefulness of telemedicine systems.
  - *Diffusion and dissemination:* Deployment of telemedicine systems is easier when telemedicine implementations are generic, i.e., applicable to other (unexpected) patient populations. Another condition necessary for the diffusion and dissemination of telemedicine initiatives is to create familiarity of the system among the stakeholders/interested parties. The stimulating role of leading champions who are willing and motivated to experiment with the new technology are essential in the process of creating familiarity and enthusiasm. As becomes clear from the literature, different stages exist in the introduction of telemedicine interventions, which might affect the process of diffusion. In general, two phases of usage of the telemedicine technology are common. Initially, there is enthusiasm but thereafter the consideration phase begins, which effects the end-users motivation of working with the telemedicine intervention either positively or negatively.
- **Financing:**
  - *Provider and structure:* Costs associated with telemedicine implementation are related to (i) investments, (ii) maintenance and (iii) operational costs of the new system. In the research stages of telemedicine, these costs are funded. However, as soon as the

research projects are ended, there is a lack of continuing funding due to a lack of, or unsuitable financing structure. Due to the novel approach of telemedicine, most third party financers do not have standard tariffs. Furthermore, it is often unclear who take the cost and benefits of introducing and running a telemedicine implementation. Sometimes the cost and benefits are taken by different parties. Additionally several studies state that comprehensive cost-effectiveness studies are essential in developing future financing structures.

- **Organization:**
  - *Intramural and extramural work practices:* As becomes clear from the analysis, telemedicine implementation is hampered by the fact that working protocols for telemedicine implementations are lacking. In addition, the introduction of telemedicine often influences the structure of the individual organization (intramural) combined with extended collaborations with other health care organizations (extramural). For instance, telemedicine might require changes in collaboration and (team) roles, rights and responsibilities. Furthermore, the novel working practices introduced by telemedicine do not always fit with existing traditional working protocols in health care.
- **Policy and Legislation:**
  - *Legislation and policy:* Legislation and policy forms a prerequisite for telemedicine implementations. The analysis indicates that legislation and policy for certain aspects of telemedicine implementations are not available. Furthermore, legislation and policy in its current form seems not suitable for all aspects of novel telemedicine implementations. The analysis indicates that deployment of wide-scale telemedicine implementations is hard without suiting legislation and policy. Additionally, conforming to legislation and policy implies additional development effort which increases time-to-market and costs compared to domains less influenced by legislation and policy.
  - *Standardization:* Standards form a mechanism to ensure quality and uniform practice. Standards are required for effective cooperation between partners in the value chain and to be able to scale-up implementations from the pilot phase. The analysis shows that standards are not yet available for all aspects of telemedicine implementations. Interoperability between telemedicine implementations is important to support the current trend of extramural work practices and is not guaranteed without globally accepted standards.

– Security: The analysis shows that security is important in two ways: (i) patient physical safety and (ii) patient information security. For acceptance of telemedicine implementations adequate security mechanisms have to be taken into account. These security mechanisms should support the crucial trust relation between healthcare providers and patients. Results show that there is also need for secure information transfer and, authentication and authorization mechanisms.

Discussion

Figure 7-2 shows that different stakeholders from different domains are influenced by the identified determinants. Healthcare customers (e.g. patients) and healthcare professionals require to accept a telemedicine system. Regulation bodies (e.g. government) may impose policy and legislation on the deployment and use of a telemedicine system. Third party financers (e.g. insurance companies) provide the financial framework for usage of a telemedicine system. Technology providers need to develop technology to create a telemedicine system. Finally, healthcare organizations (e.g. hospital) need to tailor their organization structure to comply with the work practices required for deploying a telemedicine system. Hence, developing, deploying and using telemedicine systems in an operational setting is a multidisciplinary activity.

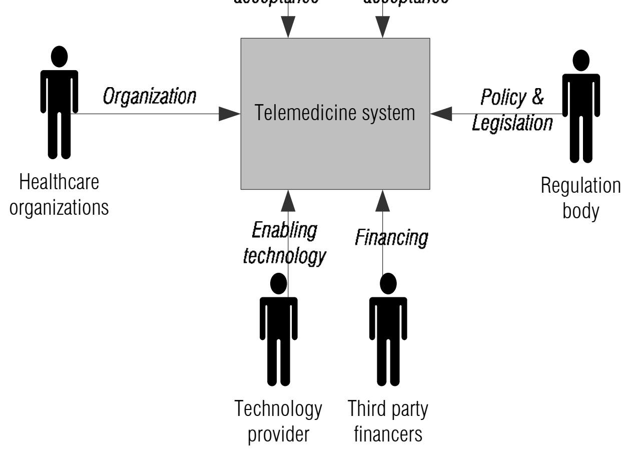
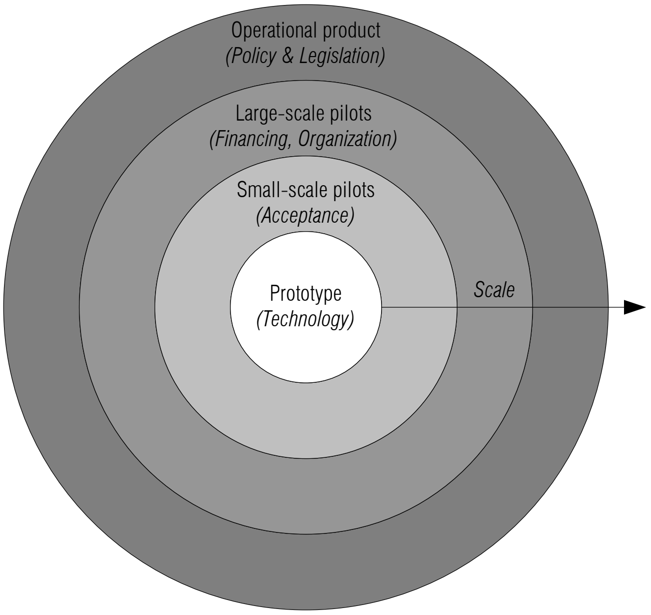


Figure 7-2 Identified determinants and the stakeholders that are influenced by them.

Therefore, it is necessary to collect domain-specific knowledge on the different determinants by involving domain-specific stakeholders. However, the main challenge for telemedicine implementation is not only to address the domain specific issues but also to integrate the different related domains by inter-organizational collaboration (e.g. business, government and health care). This collaboration is different from market collaboration as in telemedicine most often the participants remain relatively autonomous and must be convinced to act even though mutual interests (e.g. business versus quality of care) and a legitimate authority is lacking.

In order to cope with the multidisciplinary complexity, we propose a layered implementation model in which throughout the development life cycle of the telemedicine implementation, the primary focus on individual determinants change. Different determinants should gain focus during the maturity of the telemedicine implementation (see *Figure 7-3*). However, the other determinants should not be ignored to be able to anticipate on future stages in the development life cycle. In the prototyping phase, the evaluation deals mainly with the technological feasibility like the availability, quality and support of the used technology. In the small-scale pilot phase, users need to work with the system, which shifts the focus to acceptance. When small-scale telemedicine pilots move to a larger scope, financing and organization become increasingly important. When the systems become an operational product, policy issues already must have been tackled. This does not mean that when the scale of telemedicine implementations increases, determinant categories in lower layers are not of interest in higher layers, only the focus shifts to specific determinant in that layer.

Figure 7-3 Layered implementation model.

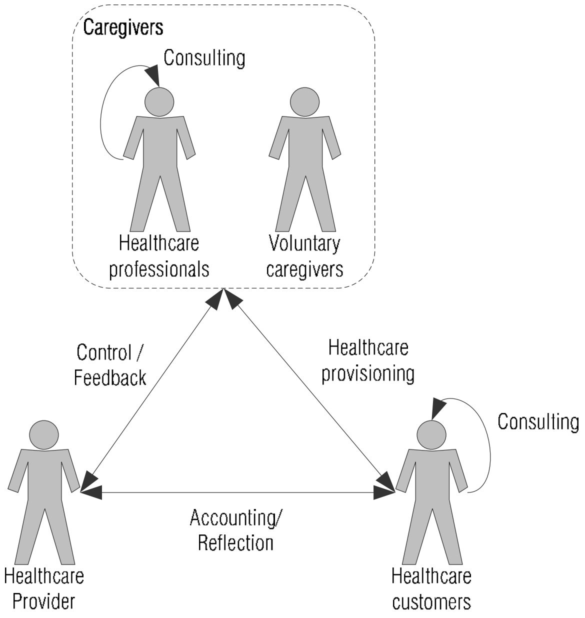


Concluding, telemedicine implementations imply a visionary approach, which goes beyond tackling specific issues in a particular development phase. Parallel efforts towards the next phases of the telemedicine life cycle can increase the probability of success: "start small, think big". When gaining maturity (i.e. scaling) the determinants shift from being specific to an individual implementation to more generic problems common in the telemedicine domain. Therefore, efforts to solve these determinants should not be solely restricted to the individual implementations but can also benefit from interaction with other initiatives. As stakeholders come to share a vision of the implementation problem and see themselves, collectively, as part of the solution it might produce mutual agreement upon directions and boundaries which then become more permanent structures surviving even after the project (funding) has ended.

7.4 Analysis of Telemedicine Systems

From the system perspective, we distinguish four primary stakeholders involved in telemedicine systems (see *Figure 7-4*). There are the **healthcare customers**, which represent the actors that are in need of healthcare. This can be patients that have one or more diseases (i.e hospitalized or living at home), or non-diseased persons that require healthcare counselling/guidance in their daily life (i.e. wellness). Consumers can communicate with other consumers (consulting), for instance with chat or forums. Healthcare is provided by **healthcare professionals**. This stakeholder group represents the actors that are responsible for the primary healthcare provisioning process. Examples are general practitioners, physicians, surgeons, dentists and diabetic consultants. Generally, they are paid for their service. Healthcare professionals can have mutual consulting, for instance by videoconferencing. Additionally, healthcare is provided by **voluntary caregivers**. These are often relatives of the healthcare customers that provide simple healthcare (i.e. first aid) until a healthcare professional, when needed, can take over. In general, they are not paid for their services. We denote the combined group of healthcare professionals and voluntary as **caregivers**. Finally, the healthcare provisioning process is controlled (secondary healthcare process) by **healthcare providers**. The stakeholder group consists of the management of the care institution of the caregivers. This group provides requirements and objectives for the healthcare provisioning process. Furthermore, the healthcare managers are responsible for accounting of the provided healthcare to the healthcare customer or other parties and to reflect general healthcare spending to the healthcare customer. Additionally, other parties like the government and insurance companies influence the telemedicine healthcare process. However for simplicity they are omitted from the stakeholder model.

Figure 7-4 Stakeholders in the telemedicine healthcare process.



Several concrete types of telemedicine systems exist like tele-surgery, tele-psychiatry, tele-dermatology, tele-oncology etc. In general, we distinguish three categories of systems in telemedicine:

- *Tele-monitoring*: systems that transfer vital signs from a patient to the healthcare professional. Typically, this is unidirectional communication. This knowledge is analyzed by the healthcare professionals who can make a medical diagnosis.
- *Tele-treatment*: system that transfers vital signs and feedback information between patient and healthcare professionals. Typically, this is a bidirectional communication. First the patient's vital signs are transferred to the healthcare specialist who can make a diagnosis. Based on this diagnosis feedback is given to the patient to improve his healthcare situation.
- *Tele-consulting*: systems that focus on healthcare related human interaction using ICT.

Another possible dimension to categorize telemedicine systems is a division based on involved stakeholders:

- *Patient – Patient systems:* systems that focus on information exchange between healthcare customers (i.e. patients).
- *Patient – Professional systems:* systems which are aimed at information and communication exchange between a healthcare professional and customer.
- *Professional – Professional systems:* systems that focus on information exchange between healthcare professionals.

In *Table 7-2* we give examples of the different categories of telemedicine systems.

Table 7-2 Examples of telemedicine systems.

	Patient-Patient	Patient-Professional	Professional - Professional
Tele-monitoring	n/a	Vital sign monitoring	Tele-surgery
Tele-treatment	n/a	Chronic pain feedback	n/a

Figure 7-5 shows a high-level (telemedicine) healthcare process. In case of a healthcare request of a healthcare customer (e.g. emergency, visit to a general practitioner), a diagnosis phase is started. In this phase the healthcare professional performs anamnesis to collect patient information such as subjective problem description, healthcare history of the patient (e.g. earlier treated diseases, known allergies), and current family situation. Furthermore, the healthcare professional observes the patient (e.g. manual measurement of blood pressure and temperature). If a diagnosis cannot be made directly the patient can be equipped with a tele-monitoring application to further observe the patients during a certain period of time (e.g. vital signs, movement data).

Manual observations, tele-monitored data and the patient information are used to make diagnosis and to decide on the plans for treatment of the patient. These plans include a treatment plan (i.e. when to give what feedback) and monitoring plan (i.e. what vital signs are important to monitor to be able to give the right feedback). The patient is equipped with a tele-treatment application (i.e. including tele-monitoring). Based on the tele-monitored data and possible intervention of the healthcare professional, feedback is given to the patient. Reviewed tele-monitored data by the healthcare professional can lead to a re-evaluation diagnosis and possibly of adaption of the treatment and monitoring plans.

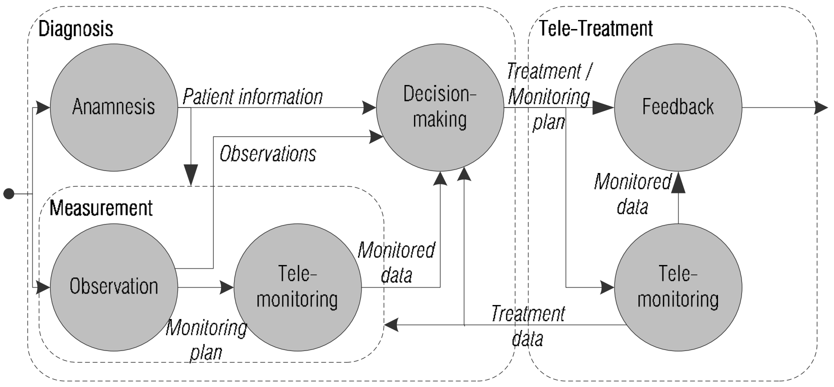


Figure 7-5 Visualization of a Telemedicine healthcare process.

7.5 Current Context-Aware Telemedicine applications

Currently, there exist several (research) context-aware telemedicine applications. In this section, we give an overview by discussing a small subset of examples.

Many telemedicine applications use location information to adapt their behaviour (so-called location-aware application). Boulos (Boulos 2003) proposes a location-aware system that adapts the presented information to the user location. In this way, they overcome the overload of the user with unnecessary information, to improve the decision-making process. User location is determined by mapping the users IP address onto a physical location. Helal et al. (Helal, Winkler et al. 2003) proposes a location-aware telemedicine application. This application has as goal to promote an independent lifestyle for elderly. Therefore, they define a smart home that proactively reacts on location changes of the elderly person. They determine the indoor location of persons by using ultrasound technology. Liska et al. (Liszka, Mackin et al. 2004) discusses a remote arrhythmia montoring application developed at NASA. This system collects real-time ECG signals from a patient combining them with user-location context information (i.e. GPS based). These signals are transmitted to a remote station for monitoring and decision-making. Lee (Lee, Lim et al. 2005) proposes a baby-care system that detects possible dangerous situation and then notifies nearby caregivers with the location of the baby such that they may prevent this situations.

There also exist applications that use a combination of location and other context information. Stanford (Standford 2002) proposes a context-aware elderly care application. This application has as goal to provide the elderly person with high quality care without losing his autonomy. Context information used in this system is user location (i.e. using locator badges), weight (i.e. weight sensor embedded in the person's bed), activity (i.e. inference on sleep/non-sleep periods using pressure sensors in the person's bed). Bardram (Bardram 2004) discusses the usefulness of context-awareness in hospitals. They present an application that uses the location and identity of the patient, caregiver, and objects in the surrounding of both, to personalize the information provisioning at the bed of the patient.

Finally, there are initiatives to capture recurring development problems of context-aware telemedicine system in an infrastructure. Zhang et al. (Zhang, Yu et al. 2004) proposes an infrastructure for delivery, management and deployment of context-aware personalized healthcare services. This infrastructure offers support functions related to device access, service interoperability, and context management. Hence, it provides generic support for a broad range of context information. Jones et al. (Jones, Mei et al. 2007) discusses a generic context-aware telemedicine infrastructure developed in the Dutch AWARENESS project. The patient wears a so-called Body Area Network (BAN), which collects vital signs of a mobile patient that are sent to a remote location for monitoring. Several neurology applications are being developed. One aspect discussed in (Jones, Mei et al. 2007) is power management of the mobile devices worn by the patient, based on device capacity context information.

7.6 Usefulness of Context-Awareness for Telemedicine Applications

When considering the general social-economical healthcare trends discussed in Section 7.2, introducing context-awareness may be beneficial to cope with some of the consequences of these trends. For example, increased patient-centric healthcare can be achieved by providing telemedicine applications that adapt to the context of the patient and caregivers. Additionally, when considering the trend of cost savings and efficiency improvements, adapting to the context of the patient becomes increasingly important when patients are treated as long as possible in their home environment (i.e. extramural care).

In Section 7.3, we present key determinants that influence the success of telemedicine applications. As context-awareness is a technical solution for personalized application behaviour, it mainly influences the first two phases of a telemedicine development initiative. Both in the technology and acceptance category, availability of the right information, at the right time, using the right modality, was identified as a factor that influences the success of telemedicine applications. As stated in Section 2.4, context information can be used in applications to (i) producing higher quality outputs, (ii) replace, minimize or tailor the user inputs (iii) internal adaptation. Hence, using context information in telemedicine application has the potential to provide the personalized behaviour as identified in the determinant analysis.

Finally, when considering the telemedicine workflow discussed in Section 7.4, we identify possible examples of applying context-aware application in the phases of this workflow (see Table 7-3). In the presented table 'n/a' denotes not applicable (i.e. no applications are used in this phase), '-' denotes no foreseen influence of context information, '√' denotes foreseen influence of context information. One row in the table should be read as follows: "an application used during the {X} phase by stakeholder {Y} {could | could not} be influenced by context information to provide {higher quality outputs | replaced, minimized, tailored inputs | internal adaptation}, for example by {example}".

Table 7-3 Examples of the usage of context-aware applications in the telemedicine workflow.

	Higher quality outputs	**Replace, minimize, tailored inputs**	**Internal adaptation**
Observation	n/a	n/a	n/a
Anamnesis (caregiver)	√ e.g. filtered anamnesis report based on patient identification	√ e.g. automatic patient identification and selection of patient records	-
Decision-making (caregiver)	√ e.g. filtered anamnesis, observation and monitoring report	√ e.g. automatic selection of patient data	-
Feedback (patient)	√ e.g. adapted timing and output modality of feedback based on patient activity	√ e.g. adapted input modality based on location and availability of input devices	-

Tele-monitoring (caregiver/patient)	√ e.g. annotated monitored data with context information	√ e.g. automatic inclusion of context information in the monitored data	√ e.g. optimized transfer of monitoring data based on bandwidth context

Context-Aware Telemedicine Applications in Emergency Situations

To give an example of the usefulness of context-awareness for telemedicine applications, we want to especially mention the potential of context-aware telemedicine applications in emergency situations, where there is a 'life-or-death' situation for patients. A common concept applicable in these situations is the 'golden hour' (Jones, Bults et al. 2001; Lerner and Moscati 2002). The golden hour is the first sixty minutes after an emergency occurs. It is believed that the care provided in this hour highly influences the survival and recovery of the patient. Hence, it is important to use this hour as effectively as possible.

Key aspects that influence the efficiency of the golden hour are: (i) the time between occurrence of the emergency and the treatment of the patient, and (ii) the availability of relevant treatment information (e.g. patient history, vital signs). Incorporating context information in emergency telemedicine applications may reduce travelling time and offer more relevant treatment information.

Considering the first aspect, actions that consume precious time are finding available caregivers and locating and travelling to the patient(s) (e.g. by an ambulance (Peters and Hall 1999)). In case of a known and equipped patient population (e.g. pregnant, high blood pressure patients, equipped with a telemedicine system) the physical location context information of the patient can be transferred together with the vital signs to possible caregivers. This information can be used to better locate the patients and hence decrease travelling overhead (e.g. using smart-signs (Lijding, Benz et al. 2006)). Furthermore, activity context information of the caregiver can be incorporated in the caregiver dispatching decision to quicker dispatch available caregivers and to decrease false dispatching of unavailable caregivers. When considering the second aspect, filtering unnecessary information is time consuming and may decrease treatment quality. Context-depended information provisioning may therefore improve the treatment process. For example, when bandwidth conditions decrease, the throughput of the complete set of vital signs is limited. Such a situation may result in the loss of vital sign measurements or delay of transfer. Hence, it might be better to incorporate bandwidth context information to decide to reduce the sampling frequency or omit certain (less relevant) data to still provide relevant information to the caregivers.

Chapter 8

Evaluation

The main objective of this thesis is to develop infrastructure-based mechanisms to *improve the development process of context-aware applications*. This chapter evaluates possible improvements when using the proposed context binding infrastructure. Parts of this chapter are published in (Broens, Sinderen et al. 2007).

This chapter is structured as follows: Section 8.1 describes and motivates the applied evaluation approach. Section 8.2 describes the results of a conducted user expectation survey. Section 8.3 discusses a telemedicine case and discusses how to implement the corresponding application with CACI, based on the development guidelines proposed in Chapter 4. Section 8.4 compares the development process of the telemedicine application when developed with CACI and with a different context middleware. Finally, Section 8.5 contains a general discussion on the performed evaluations.

8.1 Evaluation Approach

This section discusses and motivates the evaluation approach. It starts with an overview of evaluation approaches and gives our general evaluation direction. Subsequently, it discusses evaluation criteria and finally the adopted approach.

General Approach

Evaluation of a development process of a system can be divided into evaluating (i) the process to realize this system and (ii) the quality of the resulting system.

We start the discussion from the perspective of evaluating the resulting system. In computer science there are generally three approaches to evaluate the development of a proposed system (Dodig-Crnkovic 2002;

Gokturk 2007): (i) analytical modelling, (ii) simulations and (iii) experiments. In the first approach a mathematical model of the system is created and this model is used to formally reason about the system's capabilities. In the second approach the proposed system is modelled and executed in a simulation environment to estimate the system's capabilities in a controlled environment. In the third approach a prototype of the system itself is created and experiments are done to acquire measurements on its capabilities. These approaches have their own particular advantages and drawbacks (Skadron, Martonosi et al. 2003).

In practice, an evaluation can consist of a combination of these approaches. For example, the system on which experiments are performed is typically a partial implementation of the full system and is complemented with simulated parts. This kind of evaluation is called emulation (Gokturk 2007).

In this thesis, we perform an emulation approach. We use a combination of simulation and experiments to evaluate applications created with the context binding infrastructure. We use the created prototype of the context binding infrastructure in combination with simulated context sources to perform experiments.

From the perspective of evaluating the process to develop a system, ideally, the evaluation should involve multiple third-party development teams that create multiple realistic 3rd generation context-aware applications, with and without the proposed context binding infrastructure. The development process of these teams and the quality of the developed applications, with and without the use of the context binding infrastructure, should then be compared. However, there are several reasons why this is not feasible:

- The envisioned world in which context sources are widely available is not yet realized. Currently, context sources are specifically chosen and deployed for individual context-aware applications. Hence, 3rd generation context-aware applications, which benefit the most from our proposed context binding mechanism, are not yet being developed.
- To avoid unwanted interference in the evaluation, a sufficiently mature (feature complete and bug-free) context binding infrastructure is required. This requires development effort not feasible in the timeframe of this thesis.
- The business value for third party developers of an evaluation of the proposed context binding infrastructure is limited.

For these reasons and due to timing and resource constraints of this research, we took a pragmatic approach, in which the author acts as the experimenter (this approach is called assertion), combined with limited field studies with possible users (Zelkowitz and Wallace 1997). In our case users are (context-aware) application developers.

Evaluation Criteria

The goal of the present evaluation is to measure the capabilities of the context binding infrastructure to facilitate the development of context-aware applications. We do this by 'measuring' if improvement in the development process of context-aware applications can be realized. However what does 'improvement' mean?

When taking the software engineering perspective, measurements can be done on three subjects (i) processes: software related activities that take place over time, (ii) products: artefacts which arise out of the processes and (iii) resources: artefacts which are inputs to processes (Fenton 1994). These measurements that can be done on: (i) internal attributes and (ii) external attributes. Internal attributes can be measured purely in terms of the product, process or resource itself, while external attributes are also related to other entities in the environment. For example, 'lines of code' of an application are considered an internal attribute because it only depends on the software product itself. Contrarily, time spent to develop an application is considered an external attribute as it not only depends on the development process but also on, amongst others, the knowledge level of the application developer. In general, external attributes are harder to measure and interpret than internal attributes.

We believe 'improvement of the development process of context-aware applications' can be evaluated by a set of evaluation criteria. We are interested in: (i) usefulness of the context binding infrastructure (i.e. resource), (ii) the development effort of creating a context-aware application (i.e. process) and (iii) the software quality of the resulting context-aware application (i.e. product). All are external attributes that not only depend on the product and process itself but also on the application developer who is using the context binding infrastructure.

For reasons mentioned earlier, we are limited in doing full fledged measurements with a target audience of application developers. However, to still get an insight in the usefulness of the context binding infrastructure and the development effort of creating a context-aware application with the context binding infrastructure, for possible users, we perform a user survey. Additionally, to estimate the development effort and software quality of context-aware applications using our context binding infrastructure, we implement an application as part of a case study, which we evaluate using the de-facto software quality standard ISO/IEC 9126.

Adopted Approach

The adopted approach is visually represented in *Figure 8-1*. Rounded rectangles present evaluation steps, while rectangles present artefacts used in the evaluation steps. Grey coloured figures indicate steps and artefacts developed in the scope of this thesis, while white figures indicate in literature available artefacts. Arrows indicate a "used by" relation.

In our approach we evaluate the development process using the proposed context binding infrastructure in three steps:

1. *User expectation survey*: this experiment provides ratings of application developers on their expectations on the usefulness of the proposed transparency and context binding infrastructure. Additionally, it identifies evaluation criteria, which are rated as important by the possible users.
2. *Case-study with CACI*: this experiment provides a feasibility study on how to use CACI to implement a context-aware application. As part of this case study, context sources are simulated. The results of this experiment form the basis for a comparison of the development effort and software quality with a case-study without CACI.
3. *Case-study without CACI:* this experiment describes the development process of a context-aware application with a currently available context middleware. These results are compared with the case study when using CACI.

Finally, these results are evaluated using the criteria identified by the user expectation survey and the criteria proposed by the de-facto software quality standard ISO/IEC9126.

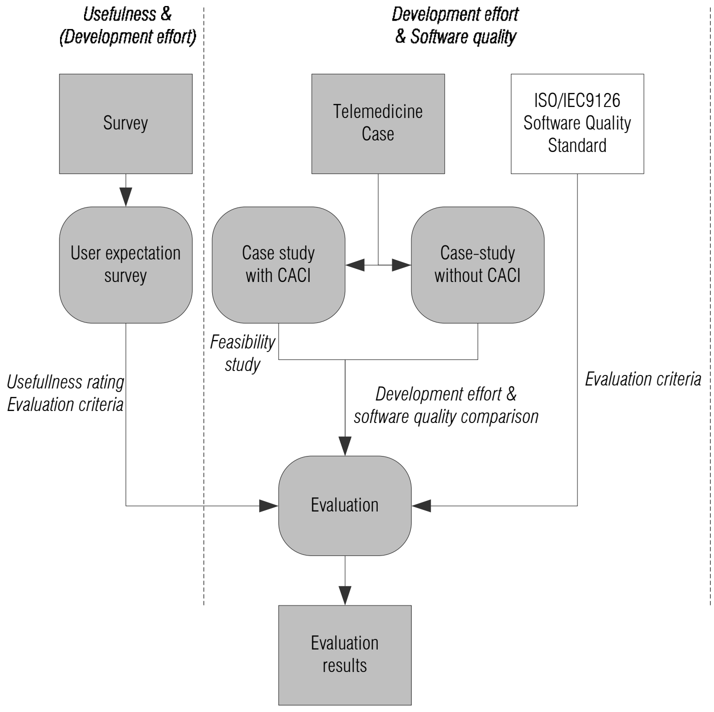

Usefulness & (Development effort)
Development effort & Software quality
Survey
Telemedicine Case
ISO/IEC9126 Software Quality Standard
User expectation survey
Case study with CACI
Case-study without CACI
Feasibility study
Usefullness rating Evaluation criteria
Development effort & software quality comparison
Evaluation criteria
Evaluation
Evaluation results

Figure 8-1 Visual representation of the evaluation approach.

8.2 User Expectation Survey

We conduct a user expectation survey to get insight in the expectation of possible users concerning the usefulness of the proposed context binding infrastructure. We start with discussing the applied method, followed by the results of the survey and finally a discussion.

8.2.1 Method

A survey is a quantitative approach to collect data from a large target audience. This data is analyzed using statistical methods to be able to provide generic statements (Gable 1994).

The target audience of the user expectation survey are developers of (context-aware) applications both originating from research and industry. To reach a broad range of application developers, we chose to perform an anonymous questionnaire to solicited and non-solicited respondents. The approached industrial target audience ranges from large international companies, such as Philips, Alcatel-Lucent, Microsoft, Océ, Thales, Appear networks, ETHZ and VTT, to smaller national firms, such as Topicus, Trimm and TSi solutions. A subset of the approached research target audience consists of: several groups of the University of Twente, research partners in the AMIGO and AWARENESS project, TU Vienna, University of Quebec and Trinity College.

The questionnaire was provided in a paper-based and web-based version. The web-based questionnaire was published on an electronic survey system called Sirvay[10]. The paper-based questionnaire was offered to the visitors of the EUNICE'07[11] and ACT4SOC'07[12] conferences, after presentations of the context binding infrastructure by the author. The results of the paper-based questionnaire are as-is submitted to the Sirvay system, to enable convenient data processing. The web-based questionnaire was open to the visitors of the website of the author. Additionally, the social network of the author is approached with a request to complete the web-based questionnaire and to forward it to possible interested other audiences. The web-based questionnaire contains a short explanation of the proposed context binding transparency and binding infrastructure, and links to further readings. The web-based questionnaire was available to respondents in the period July 2007 to December 2007.

The questionnaire itself is presented in Appendix A. It consists of 11 multiple-choice and open questions. The multiple choice questions are mainly used to (i) determine the characteristics of the target audience and

[10] http://www.sirvay.nl/, http://www.dkss.nl/

[11] http://www.ctit.utwente.nl/conferences/eunice2007/

[12] http://www.icsoft.org/ICSOFT2007/ACT4SOC.htm

(ii) to grade the expected usefulness of the proposed context binding transparency and context binding infrastructure. The open questions are mainly used to (i) retrieve opinions on limiting factors of the proposed transparency and context binding infrastructure and (ii) to retrieve other remarks.

8.2.2 Results

We estimate the total size of the (solicited) target audience on 300 persons. In total the web-based questionnaire received 201 visits. From these visits, 72 respondents completed the questionnaire. This is a response rate of approximately 36% compared to the visits. When taking into account the total target audience, this is a response rate of 24%.

We estimate that approximately 55% of the respondents originate from research while 30% originate from industry. The origin of 15% of the respondents could not be determined.

Respondents Characteristics

The first part of the questionnaire, question 1 to 5, is used to determine the characteristics of the respondent. From the results of this part of the questionnaire, we distinguish the following overall characteristics of the respondents:

– *Table 8-1* presents the results of question 1. One respondent did not answer question 1. Approximately 69,4% of the respondents do research on Context-Awareness.

Table 8-1 Results question 1.

Q1: Do you perform research in the area of context-awareness or related areas (e.g. ubiquitous, pervasive computing, ambient intelligence)?		
	Abs.	Perc. (%)
Yes	50	69.4
No	21	29.2
Totals	71	98.6

– *Table 8-2* presents the results of question 2, in which we aggregate the mentioned research aspects with a count on how often this aspect is mentioned. The research areas of the respondents are diverse. However, many respondents indicate research areas directly related to context-awareness and/or middleware.

Table 8-2 Results question 2.

Q2: In what area do you perform research?	
Research Area	**Abs.**
Context Middleware	17
None-Context Middleware	5
Context-Awareness	5
Mobile Computing	4
(Ambient) in-home systems	4
Mobile health applications	3
Security and Privacy	2
Information Systems	2
Management of Optical Networks	1
Databases	1
Lighting Scenarios	1
Multimedia Services	1
Human monitoring	1
Service composition	1
Quality of Service	1
Service discovery	1

– *Table 8-3* presents the results of question 3. Approximately 44,4% of the respondents developed context-aware applications before. 20,8% has not developed (context-aware) applications before but is planning to.

Table 8-3 Results question 3.

Q3: Have you ever developed a (context-aware) software application?		
	Abs.	Perc. (%)
Yes, I developed context-aware applications.	32	44.4
Yes, I developed non-context-aware appplications.	11	15.3
No but I am planning to.	15	20.8
No and I am not planning to.	14	19.4
Totals	72	100

– *Table 8-4* presents the results of question 4. One respondend did not answer this question. Approximately 47,2 % of the respondents have used some form of middleware to develop applications.

Table 8-4 Results question 4.

Q4: Have you ever used middleware (e.g. context discovery) to develop (context-aware) applications?		
	Abs.	Perc.(%)
Yes	34	47.2
No	37	51.4
Totals	71	98.6

– *Table 8-5* presents the result of question 5, in which we aggregate the mentioned used middleware technologies with a count on how often these technologies are mentioned. The respondents use a wide variety of middleware to develop applications. Context management systems are widely used by the respondents.

Table 8-5 Results question 5.

Q5: What specific type of middleware have you used before?	
Used Middleware	**Abs.**
Web Services	40
Context management	21
Jini	19
Corba	17
Service discovery	12
Java RMI	5
OSGi	4
DCOM	2
UPnP	2
J2EE	2
.NET	1
XML-RPC	1
Multimedia frameworks	1

Ratings and Comments

The second part of the questionnaire, question 6 to 11, is used to rate the usefulness of the context binding transparency and the proposed context binding infrastructure. Ratings are requested for the usefulness of the overall context binding transparency and the three elements: CBDL, context binding mechanism, and context discovery interoperability mechanism, that we propose to realize the CBT.

From the results of this part of the questionnaire, we distinguish the following ratings:

– *Table 8-6* presents the results of question 6. The majority of the respondents (~68%) estimate that the proposed context binding transparency highly improves (rating > 3) the development process of context-aware applications. On average the rating is 4.04 with a standard deviation of 0.73.

Table 8-6 Results question 6.

Q6: Do you think the proposed context binding transparency can simplify the development of context-aware applications (1 = not at all, 5 = very much)?		
	Abs.	**Perc. (%)**
1	0	0.0
2	3	4.2
3	5	6.9
4	36	50.0
5	13	18.1
Don't know.	14	19.4
Totals	71	98.6

– *Table 8-7* presents the results of question 7. The majority of the respondents (~72%) estimate that the CBDL language is highly useful (rating >3) for the development of context-aware applications. On average the rating is 4.11 with a standard deviation of 0.81.

Table 8-7 Results question 7.

Q7: How useful is the specification of context requirements in a specification language and resolving of the requirements in the binding middleware, rather than programming this in the application (1 = not at all, 5 = very much)?		
	Abs.	Perc. (%)
1	0	0.0
2	3	4.2
3	8	11.1
4	31	43.1
5	21	29.2
Don't know.	9	12.5
Totals	72	100.0

– *Table 8-8* presents the results of question 8. The majority of the respondents (~68%) estimate that a binding mechanism that maintains context bindings, highly useful (rating > 3). On average the rating is 4.18 with a standard deviation of 0.87.

Table 8-8 Results question 8.

Q8: How useful is the automatic adaptation to the availability and quality of context sources by the binding middleware (1 = not at all, 5 = very much)?		
	Abs.	Perc. (%)
1	1	1.4
2	1	1.4
3	9	12.5
4	24	33.3
5	25	34.7
Don't know.	12	16.7
Totals	72	100.0

– *Table 8-9* presents the results of question 9. The majority of the respondents (~69) estimate the usefulness of a context discovery interoperability mechanisms as highly useful (rating > 3). On average the rating is 4.20 with a standard deviation of 0.98.

Table 8-9 Results question 9.

Q9: How useful is the automatic interoperability between context discovery mechanisms by the binding middleware (1 = not at all, 5 = very much)?		
	Abs.	Perc. (%)
1	2	2.8
2	2	2.8
3	5	6.9
4	23	31.9
5	27	37.5
Don't know.	12	16.7
	71	98.6

– *Table 8-10* presents the results of question 10, in which we aggregate the mentioned aspects with a count on how often this aspect is mentioned. The respondents indicated a multitude of aspects that might influence the usefulness of the Context Binding Transparency. The top three of mentioned aspects are: (i) usability, (ii) learning curve and (iii) performance.

Table 8-10 Results question 10.

Q10: What aspects do you think will influence the success of the Context Binding Transparency?	
Aspects	Abs.
Usability	16
Learning curve	15
Performance (on mobile systems)	14
Standardization and business value	6
General applicability	4
Security, Privacy and Trust	4
Availability of context-aware applications, context sources	3
Integration with other sollutions	3
Expresiveness of context requirement language	2
Scalability	2
Perceived control	1
Determining offered QoC	1
Making QoC understandable to all parties	1
Adaptability	1
Context modelling	1
Stability	1

8.2.3 Discussion

From the results, we conclude that the respondents, total amount of 72, provide a suitable target audience to evaluating the usefulness of the context binding transparency. Firstly, the respondents originate both from research and industry. Secondly, the majority of respondents are knowledgeable on the area of context-awareness and middleware. Finally, almost half of them has built context-aware applications and therefore can be expected to have experienced some of the complexities of developing context-aware applications. Hence, the group of respondents is a mixture of largely knowledgeable and some non-knowledgeable persons, originating from both research and industry.

In *Table 8-11*, we summarize the average ratings and standard deviations on the overall concept of CBT and the three elements that we propose to realize a CBT.

Table 8-11 Summary of the survey's rating results.

Usefulness off…	**Average rating**	**Standard deviation**
Context Binding Transparency	4.04	0.73
Context requirement specification language (CBDL)	4.11	0.81
Context binding maintenance (Context binding mechanism)	4.18	0.87
Context discovery interoperability (Context Discovery interoperability mechanism)	4.20	0.98
Legend rating: 1= not useful … 5 very much useful		

We conclude that the proposed context binding transparency is appreciated by possible users. The overall CBT and all individual aspects are rated as possibly highly useful for the development of context-aware applications. Additionally, the respondents have indicated factors, such as usability, learning curve, performance and business value, which might limit the usefulness of the CBT. These factors are used as evaluation criteria in the evaluations in the remainder of this chapter.

Limitations of this user expectation survey are the limited size of the target audience. Furthermore, the respondents rate the usefulness of the CBT based on theoretical knowledge rather than practical experience. Consequently, their knowledge on the CBT is limited. Hence, the results of this analysis are purely indicative and should not be considered independently of other evaluations.

8.3 Case-study using CACI

In this section, we demonstrate the feasibility of a CACI-based development of a context-aware application by discussing the development of a telemedicine case system. In this section, we discuss how this system can be implemented using the CACI infrastructure. This discussion is based on the proposed development guidelines as presented in Section 4.5.

8.3.1 Case Description: the Epilepsy Safety System

The Epilepsy Safety System (ESS) is a system that supports epilepsy patients in their daily life. Epilepsy is a neurological disorder in which nerve cells of the brain occasionally release abnormal electric pulses, so called seizures. Due to the unexpected nature of these seizures, epileptic patients have a strong feeling of insecurity and are therefore seriously limited in their daily life. For example, in their mobility and social contacts. The ESS offers seizure detection and notifies caregivers which can offer first-aid. This enables an epilepsy patient to have a more active participation in society and have a higher quality of life.

The ESS deploys a sensor system on the patient's body, called a Body Area Network (BAN), which collects and transfers vital signs when a seizure is detected. This data is stored and analyzed in a healthcare centre for diagnosis, first-aid and treatment.

Context can play a major role in improving the healthcare process of the ESS by (i) tailoring of ESS functionality and (ii) tailoring of the ESS information. Amongst others, possible beneficial context types in the ESS are: patient and caregiver location, caregiver availability and patient BAN bandwidth usage.

Location information helps to decrease travelling time to the patient in case of emergencies. First, because the precise location of the patient (destination) is known and second because a nearby caregiver can be dispatched to the patient. Availability information of caregivers helps to decrease false dispatches of unavailable caregivers. Bandwidth usage information assists to tailor the transferred vital sign data to decrease costs in case of a non-emergency situation, while this information also assists to prevent congestion and failing transfer of vital sign data in case of emergency situations.

Consider we create the context-aware ESS first-aid system that helps patients in emergency situations. *Figure 8-2* schematically shows such a system. This requires application parts to be located at the patients, caregivers and healthcare centre. The parts need context bindings to several context sources. The patient application needs bandwidth information to decide which vital signs to send with which sampling frequency. The caregiver application needs the location of itself and the location of the patient having an emergency to determine the route to the patient in need. The healthcare-centre needs the location of the patient and caregiver, and availability of caregivers to dispatch the right caregiver to the patient. This context information can be provided by multiple and changing context sources. For example. location can be provided by RFID sensors or GPS, availability can be provided by a context source that reasons on the appointments in an Outlook calendar. The physical context sources that create/acquire context information are out of the scope of this case study.

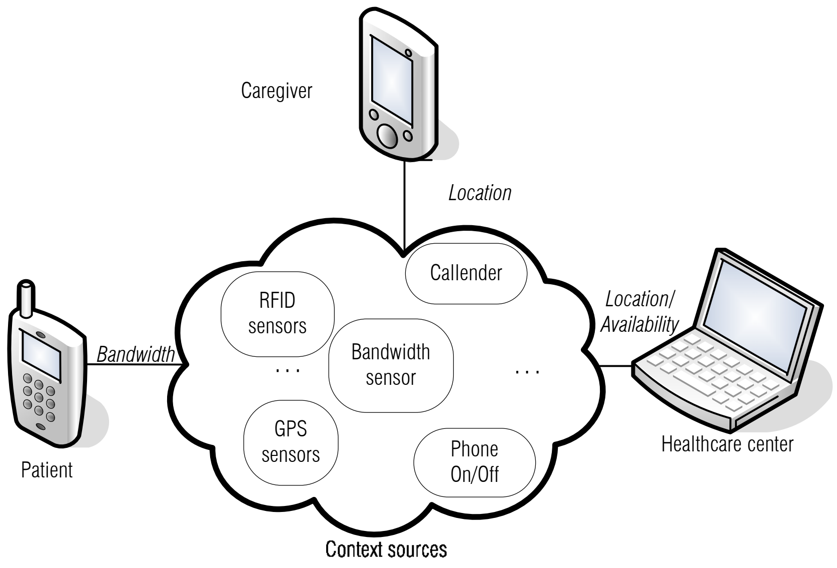


Figure 8-2 ESS application parts and required context information.

8.3.2 Developing the ESS using CACI

We discuss the development of the ESS with CACI, according to the development guidelines as proposed in Sections 4.5. We start from the situation in which application requirements for the ESS are available.

Design

In the design phase, the application developer creates the design for the application logic of the ESS application parts and determines their context requirements. In summary, the developer creates a design for the application logic of (i) the *patient application* to detect seizures and notify possible seizure alarms to the health-care centre and send the patients vital signs, (ii) the *health-care centre application* to receive alarm notifications and the patients vital signs and search nearby and available caregivers and send notifications to the selected caregiver, and (iii) the *caregiver application* to receive alarm notifications and, location and route information to the patient.

Determining the context requirements consists of a couple of steps. First the developer determines what type of context information is required for the context-aware application to execute (Step 1a). *Table 8-12* presents our choice of required context types for the ESS.

Table 8-12 Required context types in the ESS.

Application part	Context Requirement	Context type
Healthcare centre	HC-PL HC-CL HC-CA	Patient location (PL) Caregivers location (CL) Caregivers availability (CA)
Patient	PA-AB	Available bandwidth (AB)
Caregiver	CG-CL	Caregiver location (CL)

In Step 1b from the guidelines, the developer adapts the application design to incorporate situations in which the required context is unavailable. This results in n ^ 2 behaviors, where n is the number of required context types. Possible application behaviors in case of unavailable context information for the healthcare centre, are presented in *Table 8-13*. For the caregiver and patient applications similar tables (i.e. however with less rows because for both only one type of context information is required) can be made (see Appendix D).

Table 8-13 Application logic behaviours in case of unavailability of context information for the healthcare centre application.

Healthcare centre application			
HC-PL	**HC-CL**	**HC-CA**	**Application logic behaviour,** ***[HC-CAB#] are behaviours adapted from the default behaviour [HC-DB].***
x	x	x	[HC-DB]: The healthcare centre application's default behaviour. Emergency notifications are received by the healthcare centre application, the application shows the vital signs to the dispatcher which can manually dispatch caregivers stored in the caregiver database by sending a notification to their caregiver application.
v	x	x	[HC-CAB1]: The application now additionally shows the patient locations on a map and the dispatcher can manually dispatch a caregiver. Additionally the application can forward the patient location to the selected caregiver application.
x	v	x	[HC-CAB2]: The application now additionally shows the caregiver locations on a map and the dispatcher can manually dispatch a caregiver.
x	x	v	[HC-CAB3]: The application now additionally shows the availability of caregivers using availability icons and the dispatcher can manually dispatch a caregiver.
v	v	x	[HC-CAB4]: The application shows both the patient and caregiver location on a map. The application ranks the caregiver on distance to the patients. The application asks the dispatcher if the closest caregiver should be dispatched to the location of the patient. Patient location and route information are forwarded to the selected caregiver.

v	x	v	[HC-CAB5]: The application shows the location of the patient and the availability of the caregivers. The caregivers are ranked on availability. The dispatcher can manually choose an available caregiver to dispatch and forward location information of the patient.
x	v	v	[HC-CAB6]: The application shows the location and availability of the caregiver. The caregivers are ranked according to closeness to the patient and availability. The closest available caregiver is automatically put into contact with the dispatcher and possibly the patient to retrieve location and route information.
v	v	v	[HC-CAB7]: The application shows the location of the patient and caregivers on a map. The caregivers are ranked according to closeness to the patient and availability. The closest available caregiver is automatically notified of the patient's emergency and location and route information is send to his application.
Legend: 'x' = context information is unavailable 'v' = context information is available, […] = id of the behaviour.			

In Step 1c, the developer has to determine the required binding behaviour. This includes indicating the level of notification, binding policy and discovery scope for every identified context requirement. In this case, we choose to have the highest level of notification (level 3), a dynamic re-binding policy and a global discovery scope. These are the default options requiring no CBDL entries.

In Step 2 and 3, for every context requirement the application developer has to determine the entity of the required context type and the supported context format(s). This is summarized in *Table 8-14*. The entities for the healthcare centre's context requirements consist of the set of patients and caregivers known to the system, represented with {(Patient|Caregiver).*}. The entity of the patient context requirements consists of the device that is hosting the patient application. The entity of the caregiver's context requirements consists of the caregiver itself. The supported formats consist of Lat/Long coordinates for location, Boolean for availability and kb/s, expressed in a Long value, for the available bandwidth.

Table 8-14 Required context information.

Application part	Context Type	Entity	Format
Healthcare centre	Patient location (PL)	{Patient.*}	Lat/long
	Caregivers location (CL)	{Caregiver.*}	Lat/long
	Caregivers availability (CA)	{Caregiver.*}	Boolean
Patient	Available bandwidth (AB)	Device.Patient.X	Long (kb/s)
Caregiver	Caregiver location (CL)	Caregiver.X	Lat/long

Step 4a and b consists of determining possible quality levels that influence the application behaviour of the application parts. First, the minimum QoC criteria are determined followed by possible additional QoC levels. When taking the QoC criteria as proposed in the CBDL use cases (see Chapter 4), this results in the following QoC levels, with default notification and re-binding options. *Table 8-15* identifies QoC levels for the different context types.

Table 8-15 Identified QoC levels for the context types.

Application part	Context Type	QoC level
Healthcare centre	Patient location (PL)	HC-PL.min: HC-Pl.precision < 5m HC-PL.level0: HC-PL.precision > 5m HC-PL.level1: 1m< HC-PL.precision <5m HC-PL.level2: HC-PL.precision < 1m
	Caregivers location (CL)	HC-CL.min: HC-CL.precision < 100m HC-CL.level0: HC-CL.precision > 100m HC-CL.level1: 1m < HC-CL.precision < 100m HC-CL.level2. HC-CL.precision < 1m
	Caregivers availability (CA)	-
Patient	Available bandwidth (AB)	-
Caregiver	Caregiver location (CL)	-

These quality levels influence the application behaviours in the case that the corresponding context types are available. Hence, in these cases, the number of possible behaviours increases to incorporate the identified QoC levels (Step 4c). *Table 8-16* presents the relationship of QoC levels to application behaviours. For the patient and caregiver applications the number of behaviours do not change as there are no requirements on the QoC levels specified. Additionally, the application designer has to determine what happens in case the QoC of the retrieved context information is not available (Step 4d). Here he has three options: (i) consider the QoC to be at the lowest level, (ii) consider the QoC to be at the highest level or (iii) consider the QoC to be at a specified intermediate level. Here, we choose to apply situation one, which is the default setting.

Table 8-16 Relationships of context-aware behaviours with QoC levels.

Healthcare centre application			
HC-PL	**HC-CL**	**HC-CA**	**Application logic behaviour,** *[HC-CAB#-*] are behaviours adapted from behaviours [HC-CAB#]*
v	x	x	HC-PL.level0 → [HC-DB] HC-PL.level1 → [HC-CAB1] HC-PL.level2 → [HC-CAB1-1]: The application shows besides the patient location on the map also an estimated street name.
x	v	x	HC-CL.level0 → [HC-DB] HC-CL.level1 → [HC-CAB2] HC-CL.level2 → [HC-CAB2-1]: The application shows besides the caregiver location on the map also an estimated street name.
v	v	x	HC-PL.level0 + HC-CL.level0 → [HC-DB] HC-PL.level0 + HC-CL.level1 → [HC-CAB2] HC-PL.level0 + HC-CL.level2 → [HC-CAB2-1] HC-PL.level1 + HC-CL.level0 → [HC-CAB1] HC-PL.level1 + HC-CL.level1 → [HC-CAB4] HC-PL.level1 + HC-CL.level2 → [HC-CAB4-1]: The application shows besides the patient and caregiver location on the map also an estimate of the street name of the caregiver. HC-PL.level2 + HC-CL.level0 → [HC-CAB1-1] HC-PL.level2 + HC-CL.level1 → [HC-CAB4-2]: The application shows besides the patient and caregiver location on the map also an estimate of the street name of the patient. HC-PL.level2 + HC-CL.level2 → [HC-CAB4-3]: The application shows besides the patient and caregiver location on the map also an estimate of the street name of the caregiver and patient.
v	x	v	HC-PL.level0 → [HC-DB] HC-PL.level1 → [HC-CAB5] HC-PL.level2 → [HC-CAB5-1]: The application shows besides the patient location on the map also an estimated street name.
x	v	v	HC-CL.level0 → [HC-DB] HC-CL.level1 → [HC-CAB6] HC-CL.level2 → [HC-CAB6-1]: The application shows besides the caregiver location on the map also an estimated street name.

v	v	v	HC-PL.level0 + HC-CL.level0 → [HC-DB] HC-PL.level0 + HC-CL.level1 → [HC-CAB6] HC-PL.level0 + HC-CL.level2 → [HC-CAB6-1] HC-PL.level1 + HC-CL.level0 → [HC-CAB5] HC-PL.level1 + HC-CL.level1 → [HC-CAB7] HC-PL.level1 + HC-CL.level2 → [HC-CAB7-1]: The application shows besides the patient and caregiver location on the map also an estimate of the street name of the caregiver. HC-PL.level2 + HC-CL.level0 → [HC-CAB5-1] HC-PL.level2 + HC-CL.level1 → [HC-CAB7-2]: The application shows besides the patient and caregiver location on the map also an estimate of the street name of the patient. HC-PL.level2 + HC-CL.level2→ [HC-CAB7-3]: The application shows besides the patient and caregiver location on the map also an estimate of the street name of the caregiver and patient. Estimated time of arrival of the caregiver at the location of the patient is calculated and send to the patient.
Legend: x = context information is unavailable v = context information is available, [...] id of the behaviour.			

Finally, in Step 5 the collected information on the context requirements is transformed in CBDL documents. *Example 8-1* shows the XML-based CBDL document of the healthcare centre application, which can be created using the CBDL XML Schema. In this document, we present one requirement for a patient Tim and a caregiver John. Requirements have to be made, in a similar fashion, for the other patients and caregivers required in the system. This could be automated by taking information from a patient/caregiver repository. Appendix D presents the CBDL documents of the patient and caregiver applications.

Example 8-1 CBDL document of the Healthcare centre application.

```
<?xml version="1.0" encoding="UTF-8"?>
<CBDLDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CBDL-schema.xsd" UserID="Healthcarecentre"
ApplicationID="ESS_Healthcarecentre">
    <ContextRequirement BindingID="HC-PL">
        <Element>Location</Element>
        <Entity>Patient.Tim</Entity>
        <Format>lat/long</Format>
        <QualityLevel id=”HC-PL.level0”>
            <QoCCriteria><Precision>&gt; 5m</Precision></QoCCriteria>
        </QualityLevel>
        <QualityLevel id=”HC-PL.level1”>
            <QoCCriteria>
              <Precision>&gt; 1m &amp; &lt; 5m</Precision>
            </QoCCriteria>
        </QualityLevel>
        <QualityLevel id=”HC-PL.level2”>
            <QoCCriteria><Precision>&lt; 1m</Precision></QoCCriteria>
        </QualityLevel>
    </ContextRequirement>
    <ContextRequirement BindingID=”HC-CL” >
        <Element>Location</Element>
        <Entity>Caregiver.John</Entity>
        <Format>lat/long</Format>
        <QualityLevel id=”HC-PL.level0”>
            <QoCCriteria><Precision>&gt; 100m</Precision></QoCCriteria>
        </QualityLevel>
        <QualityLevel id=”HC-PL.level1”>
            <QoCCriteria>
              <Precision>&gt; 1m  &amp; &lt; 100m</Precision>
            </QoCCriteria>
        </QualityLevel>
        <QualityLevel id=”HC-PL.level2”>
            <QoCCriteria><Precision>&lt; 1m</Precision></QoCCriteria>
        </QualityLevel>
    </ContextRequirement>
    <ContextRequirement BindingID="HC-CA">
        <Element>Availability</Element>
        <Entity>Caregiver.John</Entity>
        <Format>boolean</Format>
    </ContextRequirement>
</CBDLDocument>
```

The result of the design phase is: (i) CBDL documents describing the context requirements of the application parts and (ii) design of the application parts. *Figure 8-3* shows a possible high-level design of the healthcare centre application.

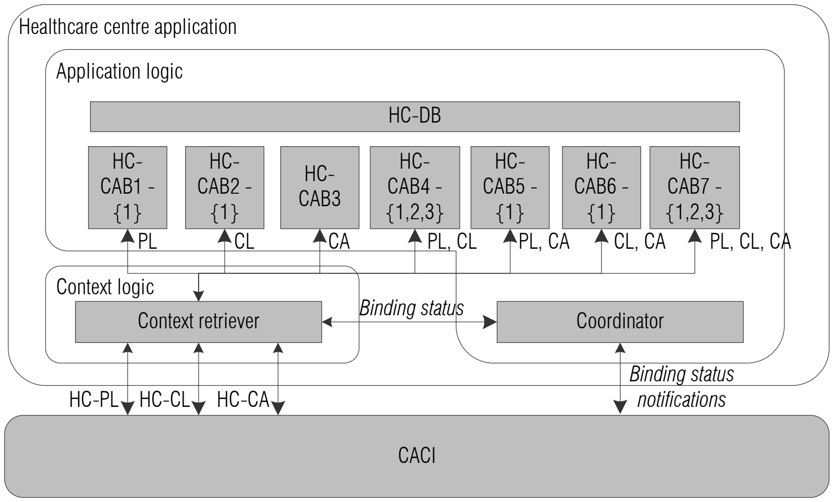


Figure 8-3 High level layered design of the healthcare centre application.

The application logic of the healthcare centre application consists of a default behaviour in case no context information is available. This basic behaviour is augmented with additional behaviours when the specific context information is available. For example, when only 'patient location' is available HC-CAB1 is enabled. In this way a hierarchy of application behaviours can be distinguished. This is represented in *Figure 8-4*. The context logic is responsible for two things: (i) retrieve the required context information and (ii) forwarding binding status information to coordinating behaviour that controls the execution of context-aware behaviours.

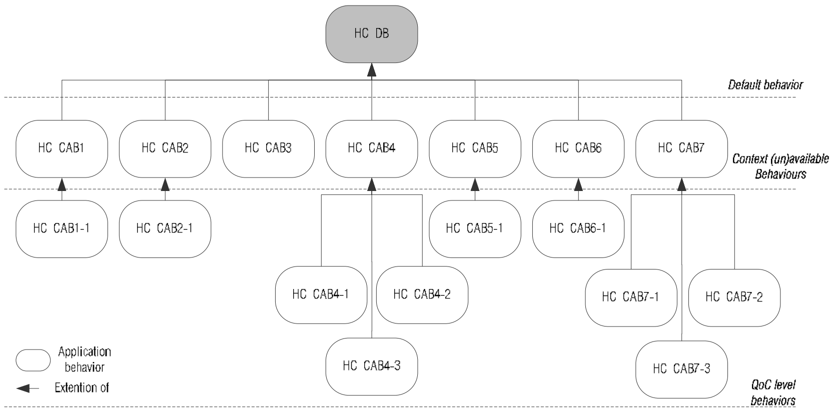


Figure 8-4 Tree of application behaviours of the healthcare centre application.

Implementation

The implementation of the healthcare centre application consists of implementing: (i) the context-aware application behaviours, (ii) the coordinator behaviour and (iii) the context retriever behaviour. We consider the first and second out of the scope of this case (i.e. Step 7 of the proposed guidelines). We focus here on the implementation of the context retriever behaviour to interact with CACI to retrieve context information and binding status notifications.

Following Step 6a, the developer uses OSGi capabilities to retrieve a handle to the context retrieval service. *Example 8-2* shows the required Java code for retrieving this handle. This handle can be used to retrieve all the required context information. Hence, retrieving this handle is done only once during the life-span of the application.

Example 8-2 Discovery and retrieval of a handle to the context retrieval service.

```
// Handle to the OSGi container provided by OSGi on deployment of the component.
BundleContext bc;

// Discovery of the ‘context retrieval OSGi service’ based on the context retrieval
// interface signature.
ServiceReference ref = bc.getServiceReference(IContextRetrievalService.class.getName());

// Retrieval of a handle to the context retrieval service.
IContextRetrievalService retriever = (IContextRetrievalService) bc.getService(ref);
```

The handle to the context retrieval service and the specified binding ID’s can be used to subscribe a callback object to CACI, for notification of a context binding proxy object that can be used to retrieve context information. *Example 8-3* shows the Java code to subscribe a callback object to CACI. Again, this subscription has to be done once in the lifetime of the application.

Example 8-3 Subscription to CACI to be notified of available context producer proxy objects.

```
// Callback to-be created by the application developer.
IContextProducerCallback cb;

try{
  // Use the context retrieval service to subscribe a callback to CACI for notification of context
  // binding proxy objects using the binding ID’s specified in the CBDL document.
  retriever.subscribe("HC-PL", cb);
  retriever.subscribe("HC-CL ", cb);
  retriever.subscribe("HC-CA", cb);
}catch(ConsumerSubscribeException e){
  System.out.println("Wrong binding ID.");
}
```

The application developer has to create an application specific callback to receive notifications of available producer proxies and updates of the binding status. This callback has to implement the 'IContextProducerCallback' interface. *Example 8-4* shows the Java code of such a callback object.

Example 8-4 Callback which receives notifications of CACI of available context producer proxy

```
// Specific callback interface to-be implemented by the application developer.
public class SpecificCallback implements IContextProducerCallback {

    // Notification of an available context producer proxy.
    public void notify(String bindingID, IContextProducer prod) {
        IContextProducer producer = prod;

        // Example of retrieving context information in a request-response manner.
        ContextInfo contextinfosample = prod.getContext();
        // Example of retrieving context information in a subscribe-notify manner.
        ISubCallback callback;
        prod.subscribe(callback);
    }

    // Notification of changes in the binding to a context producer.
    public void notifyStatus(String bindingID, int status) {
        // These states can be:
        // IContextProducerCallback.UNBOUND
        // IContextProducerCallback.BOUND
        // IContextProducerCallback.REBINDING
    }
}
```

As part of the 'notify()' and 'notifyStatus()' method, the developer has to develop code, which connects the application logic with the context logic (Step 7). In this way the application logic can retrieve the required context information for its execution. However this is out of the scope of this case.

Deployment

In the deployment phase the developer has to package: (i) the developed code from the implementation phase and (ii) the CBDL document from the design phase in a CACI-enabled component. This consists of creating a JAR file of the code and CBDL document, including adding a manifest specifying Java, OSGi and CACI properties (Step 8). *Example 8-5* shows the manifest of the healthcare centre application component.

We refer to the OSGi standard (OSGi Alliance 2005) for the description of the OSGi properties. The 'context-requirement-spec' property enables application developers to specify the filename of the CBDL document that describes the context requirements of the corresponding component. This CBDL document has to be incorporated in the application JAR file.

Example 8-5 Jar manifest of the healthcare centre application.

```
// Java standard manifest properties
Manifest-Version: 1.0
Created-By: 1.6.0 (Sun Microsystems Inc.)

// OSGi manifest properties
Bundle-Name = ESS_HC_Component
Bundle-Description =Healthcare centre application part of the ESS system
Bundle-Vendor = Tom Broens
Bundle-Version = 1.0
Bundle-UpdateLocation = http://ewi554.ewi.utwente.nl/obr/ESS_HC.jar
Bundle-Activator =nl.utwente.ESS.ESS_HC.Activator
Import-Package = nl.utwente.CACI.Common, nl.utwente.CACI.Common.Interfaces

// CACI manifest property, specifying the file name of the CBDL document corresponding to this
// component.
Context-requirement-spec = ESS_HC_CBDL.xml
```

Installing the CACI container encompasses installing a Java virtual machine and a Java-based OSGi environment. The OSGi container lifecycle functions are used to deploy and run the CACI component that instantiates the virtual CACI container (Step 9). The healthcare centre application component can be deployed and run by using the OSGi container lifecycle functions. The CACI container intercepts the deploying CACI-enabled component and starts the binding process.

Testing

If the application developer wants to test the developed application and context logic, he can decide to simulate context sources using the SimuContext framework (Step 11 & 12). For this he has to: (i) deploy and run the SimuContext bundle in the OSGi container and (ii) extend the already created CBDL document with simulation specifications, or use the run-time SimuContext services. *Example 8-6* shows an extended context requirement as part of the healthcare centre application CBDL document.

Example 8-6 CBDL code to simulate a location context source using SimuContext.

```
…
<ContextRequirement BindingID="HC-PL">
        <Element>Location</Element>
        <Entity>Patient.Tim</Entity>
        <Format>lat/long</Format>
       …. QoC levels …
       <SimuContext>
           <Valuemodel>
           nl.utwente.SimuContext.ValueModels.FileValueModel:values.log
           </Valuemodel>
        <Eventmodel>
         nl.utwente.SimuContext.EventModels.RandomEventModel:2
        </Eventmodel>
       </SimuContext>
</ContextRequirement>
…
```

8.4 Comparing CACI and Non-CACI based Development

In this section, we discuss the development of a context-aware application with a currently available and representative context discovery middleware, namely the Context Management System (CMS) (Ramparany, Poortinga et al. 2007). Additionally, we qualitatively compare the development effort and software quality of the CMS-based ESS with the previously discussed CACI-based ESS.

8.4.1 Using the CMS for Context Discovery in the ESS

This section discusses the implementation of the ESS using the Context Management System (CMS). Section 3.2.5 elaborates more on the CMS itself. The CMS is part of the middleware developed in the AMIGO project[13]. Our discussion starts from a situation in which there is a running instance of the CMS, in which multiple context sources are registered. Code development is based on the CMS tutorial[14].

The application developer of the ESS has to develop the application and context logic of the application parts. As part of the context logic, the application developer has to program code to interact with the CMS. This consists of the following steps:

1. Find a CMS context broker.

[13] http://www.hitech-projects.com/euprojects/amigo/

[14] http://www.hitech-projects.com/euprojects/amigo/tutorials.htm

2. Ask the context broker for registered context sources that match the specified context requirements.
3. Subscribe to a context source, selected from a set of suitable context sources returned by the context broker.
4. React on context change notifications received from the context source.

The first three steps are reflected in *Example 8-7*. The example code presents the starting point of the application for the discovery of context sources and retrieval of context information. It contains code statements to find a CMS context broker (findContextBroker), discover context sources (findContextSource) and subscribe to a selected context source (subscribeCS). The latter two have to be repeated for every required context type. Hence in this case, these statements have to be repeated for the patient location, caregiver location and caregiver availability. In the remainder of this section, we discuss a possible implementation of these methods in more detail.

Example 8-7 Initialize a context source discovery and subscription to context information.

```
// Context requirements for the patient location, caregiver location and availability, which
// should be specified in RDF.
String HC_PL_req;
String HC_CL_req;
String HC_CA_req;

public void init() {
        /* 1. try to find a CMS context broker */
        AmigoService broker = findContextBroker();

        /* 2. ask the context broker for a reference to suitable context sources */
        AmigoService HC_PL_CS = findContextSource(broker, HC_PL_req);
        AmigoService HC_CL_CS = findContextSource(broker, HC_CL_req);
        AmigoService HC_CA_CS = findContextSource(broker, HC_CA_req);

        /* 3. subscribe to the found context sources  */
        if (source != null){
            subscribeCS(HC_PL_CS);
            subscribeCS(HC_CL_CS);
            subscribeCS(HC_CA_CS);
            // now the context-aware application is ready to be notified by the context sources
           // of changes in the context information.
        }
}
```

The context requirements of the application are expressed in strings in the RDF[15] format. Such a RDF string is used to specify the type of context information the application requires. *Example 8-8* shows the RDF string for the context requirement of the 'patient location' (HC_PL_req). Similar RDF strings have to be specified for the other types of context requirements (caregiver location and availability). The information model used by CMS is specified in a context ontology (i.e. http://amigo.gforge.inria.fr/owl/Context.owl) in the OWL[16] format. The concepts used in the RDF strings have to be specified in the Amigo context ontology to guarantee correct working of the CMS.

Example 8-8 Patient location context requirement specification in RDF.

```
private static String HC_PL_req = ""+
"<?xml version=\"1.0\"?>"+
"<rdf:RDF"+
"   xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\""+
"   xmlns:rdfs=\"http://www.w3.org/2000/01/rdf-schema#\""+
"   xmlns:owl=\"http://www.w3.org/2002/07/owl#\""+
"   xmlns:j.0=\"http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#\""+
"   xmlns:j.1=\"http://amigo.gforge.inria.fr/owl/Domotics.owl#\""+
"   xmlns:daml=\"http://www.daml.org/2001/03/daml+oil#\""+
"   xmlns:j.2=\"http://amigo.gforge.inria.fr/owl/Context.owl#\">"+
"  <j.2:ContextSourceRegistration>"+
"    <j.2:contextType>PatientLocation</j.2:contextType>"+
"  </j.2:ContextSourceRegistration>"+
"</rdf:RDF>";
```

Finding a CMS Context Broker

Before being able to discover context sources, the application has to find a CMS context broker. *Example 8-9* shows an implementation of the 'findContextBroker' method. This consists of a web service look-up of the 'ContextBroker' service using the Amigo middleware. Additionally, some exception handling is needed in case of an error or when no broker can be found. The process of finding a context broker has to be done only once per application part. A context broker can be reused for finding multiple context sources.

[15] http://www.w3.org/RDF/
[16] http://www.w3.org/2004/OWL/

Example 8-9 Implementation of the findContextBroker() method.

```
private AmigoService findContextBroker() {
    AmigoService broker = null;
    try {
        // Web service lookup of a 'ContextBroker' service
        broker = lookup.lookupFirstService("urn:amigo","ContextBroker");
        if (broker == null) {
            logger.debug("No ContextBroker discovered");
        }
    } catch (AmigoException e) {
        e.printStackTrace();
    }
    return broker;
}
```

Finding Suitable Context Sources

The application has to invoke a discovery request on the found ContextBroker service to retrieve suitable context sources that can deliver the required context information. *Example 8-11* shows an implementation on the 'findContextSource' method. This includes invoking the 'discoverContextSource' method of the ContextBroker web service. A parameter of this method is the earlier specified context requirement.

This method returns a list of references to the network location of the discovered context sources, in a proprietary XML format. *Example 8-10* shows an example of such a list, containing references to three location context sources. This list has to be parsed and a selection of a suitable context source has to be made. In the example, the first one is selected from the list. Subsequently, a reference to the service of the selected context source has to be made.

Example 8-10 Returned string of references to discovered context sources.

```
< ?xml version='1.0' encoding='UTF-8' ?>
<listref>
  <ref> http://130.89.11.57:8080/ksoap2/LocationContextSource1<\ref>
  <ref> http://130.89.11.57:8080/ksoap2/LocationContextSource2<\ref>
  <ref> http://130.89.11.57:8080/ksoap2/LocationContextSource3<\ref>
<\listref>
```

Example 8-11 Implementation of the findContextSource method

```
private AmigoService findContextSource(AmigoService broker, String context_req) {
        AmigoService contextSource = null;
        try {
            // Invoke the discoveryContextSource method
            String result = (String) broker.getGenericStub().invoke("discoverContextSource",
                    new String[]{"contextInfoDesc"},
                    new Object[]{context_req});
            // Parsing of the first reference (selection of a context source)
            String csRef = null;
            int index_start = result.indexOf("<ref>");
            if (index_start != -1) {
                int index_end = result.indexOf("</ref>", index_start);
                if (index_end != -1) {
                    csRef = result.substring(index_start+"<ref>".length(), index_end);
                }
            }
            if (csRef != null) {
                // Creating a reference to the selected context source service
                contextSource          =          AmigoImportedService.createService(new
AmigoReference(AmigoReference.SOAP, csRef));
            }else{
                logger.warn("No suitable Context Source found!");
                contextSource = null;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return contextSource;
}
```

Subscribing to a Context Source

The selected context source can then be used to subscribe to changes in the context information acquired by this source. *Example 8-12* shows an implementation of the 'subscribeCS' method. This includes invoking the 'subscribe' method of the context source web service. Parameters of this method are a query string and a notification key. The query string can be used to ask for specific context information from a context source. This string should be formatted in the SPARQL[17] format. *Example 8-13* gives a query string that asks for the location of patient Tim and the timestamp of the context information. The notification key is used to identify from which

[17] http://www.w3.org/TR/rdf-sparql-query/

subscription a notification is received. Besides the subscription, a notification callback object has to be registered to the context source.

Example 8-12 Subscribe to the selected context source.

```
private void subscribeCS(AmigoService source){
        try{
            // Subscribe to the found context source with a specific query.
            String eventID = (String)source.getGenericStub().invoke("subscribe",
                    new                    String[]{"contextSubscriptionCharacterisation",
"contextSubscriptionReference"},
                    new Object[]{queryString,notificationKey});
            // Register a notification callback object
            source.getSubscriptionManager().subscribe(callbackobject,eventID);
        }catch(Exception ex){
            ex.printStackTrace();
        }
}
```

Example 8-13 SPARQL query.

```
final private String queryString = ""+
"PREFIX amigo: <http://amigo.gforge.inria.fr/owl/AmigoICCS.owl#> "+
"PREFIX context: <http://amigo.gforge.inria.fr/owl/ContextTransport.owl#>" +
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> "+
"SELECT ?location ?time WHERE { "+
   "?id rdf:type context: PatientLocation "+
   "?id context:location ?location . "+
   "?location context:identifier 'Patient.Tim' . "+
   "?id context:timestamp ?time ." +
"}";
```

Reacting on Context Information Changes

Finally, the application developer has to implement a callback object. This object receives notification of changes in the context information that is acquired by the selected context source. *Example 8-14* shows an implementation of the 'notify' method, which is called by the subscribed context source. The changed context information is a parameter of the notification method. The context information is formatted as a SPARQL result string. A SPARQL helper class can be used to parse and interpret the received context information. When the context information is parsed it can be passed to the application logic.

Example 8-14 Implementation of the 'notify' method.

```
public void notify(NotificationData data) {
    //use the notificationKey used earlier for the subscription, to retrieve the context information
    String result = (String) data.get(notificationKey);
    // now create a SparqlResultHelper to process the SPARQL result
    SparqlResultHelper srh = new SparqlResultHelper();
    Results rslts = srh.process(result);

    /*
     *  Examine the SPARQL Results for new temperature information by searching for
     * "room", "temp" and "time" in the result (these where the variables we defined in
     *  the SPARQL Query)
              */
    String patient_location = null;
    for (int i=0;i<rslts.size();i++) {
        for (int j=0;j<((Result)rslts.get(i)).size();j++) {
            Binding b = (Binding) ((Result)rslts.get(i)).get(j);
            if (b.getName().equals("location")){
                patient_location = b.getValue();
            }
        }
    }
    // Use the received context information in the application logic of the ESS.
    ESS_carecentre.notify(patient_location);
}
```

8.4.2 Comparison

Figure 8-5 presents, from an application developer perspective, a visual representation of a development process of a context-aware application when using (i) current context discovery mechanisms, and (ii) CACI. In both cases, the application requirements have to be transformed in application logic. Also, in both cases, context logic has to be developed to retrieve the required context information. The context logic uses capabilities offered by the context discovery middleware or CACI, respectively. For a CACI-based application part of the context logic consists of a CBDL specification. For a non CACI-based application, part of the context logic may consist of creating (multiple) specific specifications which are specific per discovery mechanism. In this section, we qualitatively compare the development effort (process) and software quality (product) of the CMS-based ESS with the developed CACI-based ESS.

Our hypothesis is that the context logic required for a CACI-based application is smaller than when using a currently available context discovery middleware. Additionally, we estimate that the quality of the application improves due to the features it inherits from CACI.

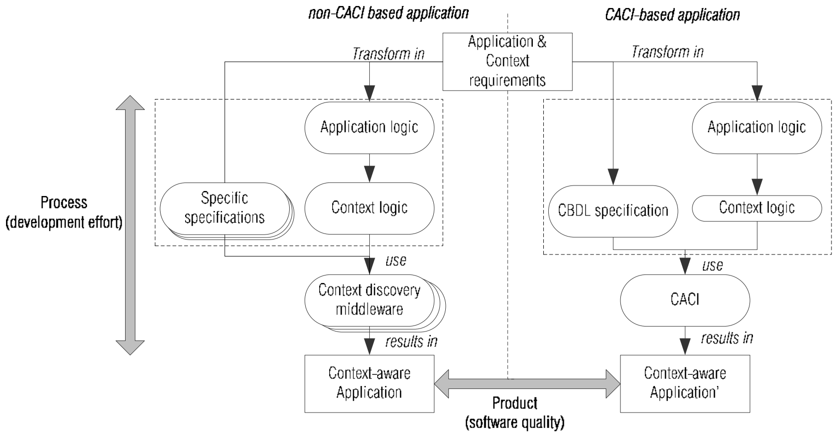

non-CACI based application
CACI-based application
Application & Context requirements
Transform in
Transform in
Application logic
Application logic
Specific specifications
Context logic
CBDL specification
Context logic
Process (development effort)
use
use
Context discovery middleware
CACI
results in
results in
Context-aware Application
Context-aware Application'
Product (software quality)

Figure 8-5 Comparing a CACI and non-CACI based development process of context-aware applications.

Development Effort

In both implementations of the ESS, the application developer has to implement the application logic. In both cases, this application logic should consist of the same behaviour that fulfils the application requirements. However, CACI provides development guidelines that enable developers to create the application logic in a structured manner. Additionally, these guidelines make the developer explicitly aware of the possibility of (un)availabile context sources and fluctuating QoC of the provided context information. The CMS does not provide guidelines for the design of the context-aware applications. We believe such guidelines positively influence the development of structured and realistic 3rd generation context-aware applications.

More explicit differences can be seen in the implementations of the context logic. *Table 8-17* shows the lines-of-code (LOC) that are required for implementing the context logic of the healthcare centre application part when using the CMS and when using CACI.

Table 8-17 Comparing required LOC for the context logic of the healthcare centre application part.

Type of code	**LOC CMS**	**LOC CACI**
Specifications (RDF,SPARQL, CBDL documents)	72	40
Java code	50	10
Total	122	50

The LOC of the context logic of the healthcare centre application part of the ESS, is for the CMS-based implementation approximately 72 lines of specification and 50 lines of Java code. For the CACI-based healthcare centre application part this is 40 lines of specification and 10 lines of Java code. Hence, when using CACI, the implementation of the context logic of the ESS requires less effort than when using the CMS. Although for this simple example, the absolute required LOC for the CMS-based ESS is not that high, relatively, it is more than one times more compared to a CACI-based ESS.

Table 8-18 shows the technologies that a developer has to know before he can use the capabilities of CMS or CACI. For both, this includes gaining knowledge of the middleware specific API's. Additionally, knowledge has to be gained on how-to specify context and/or context requirements. For the CMS this includes learning the Amigo context ontology, while for CACI this includes learning the CBDL XML Schema. For the CMS, several supporting technologies have to be learned like XML, RDF, OWL and SPARQL. For CACI only XML, as the foundation of CBDL, has to be learned. Finally, the underlying deployment middleware has to be learned. For CMS this is the proprietary Amigo middleware based on OSGi. For CACI, this includes learning OSGi. We estimate the learning curve of CACI to be less steep compared to that of CMS. First, because the number of technologies to be learned is less and secondly, the technologies used by CACI are more common. For example, we estimate the possibility that a developer has basic knowledge of XML is higher than the possibility that a developer knows SPARQL.

Table 8-18 Background knowledge required for using CMS and CACI.

Types	CMS-based application	CACI-based application
API's	CMS API's	CACI API's
Context (requirement) specification	Amigo ontology	CBDL XML Schema
Supporting technology	XML, RDF, OWL, SPARQL	XML
Deployment middleware	Amigo middleware, OSGi	OSGi

Software Quality

Table 8-19 compares the features of the CMS and CACI-based ESS applications. As discussed before, CACI offers a structured development process for the application logic. Furthermore, both CMS and CACI offer ways for an application to discover context sources that can deliver the required context information. However, CACI additionally enables the application to discover context sources from multiple dynamically available context discovery mechanisms. CACI performs the selection of a suitable context source on behalf of the application based on a default or application-specific selection algorithm. In case of a failing binding, CACI tries to re-establish the binding to enable continued availability of context information to the application. Finally, contrary to CACI, CMS offers a context ontology to uniformly specify context concepts and semantics.

Table 8-19 Groupware categories and roughly equivalent names.

Features	CMS-based application	CACI-based application
Structured application logic according to guidelines	-	●
Discovery of context sources	●	●
Discovery of context source from different context discovery mechanisms	-	●
Automatic selection of a suitable context source	-	●
Rebinding in case of a failing binding	-	●
Uniform context concepts and semantics specified in a context ontology	●	-
Legend: '●' = offers, '-' = does not offer		

Summarizing this section, we estimate that when a developer uses CACI to create its context-aware application, the application is more structured, costs less development effort to create, and has a higher quality compared to a CMS-based application. An important feature that CMS offers to a context-aware application is a context ontology to uniformly specify context concepts and semantics. However, we consider this an orthogonal aspect that can be added to CACI. However, this requires further research.

8.5 Discussion

In this chapter we have evaluated the possible improvement of the development process of context-aware applications when using CACI. First, we showed the general interest of application developers in a context binding transparency and a context binding infrastructure. Secondly, we showed the feasibility of creating a context-aware application using CACI, by implementing a telemedicine case based on the CACI development guidelines and prototype. Thirdly, we presented the implementation of the same case with a currently available context discovery middleware and compared the development effort and software quality of both case studies. This comparison showed that for the given case, CACI provides an infrastructure to create more structured and higher quality applications, which cost less development effort to create.

In the remainder of this section, we abstract from the cases and reflect in general on possible improvements of the development process of context-aware applications.

General Reflection

By using CACI the application developer is relieved from the responsibility of developing software for creating and maintaining context bindings to retrieve context information. Application developers can focus on their primary task of creating application logic needed for their application.

Additionally, application developers become more aware of the situations in which their applications possibly function. When using the proposed guidelines in the development process, developers take situations into account in which context information is not available or certain quality criteria are not met. This results in a set of application behaviours corresponding to these situations. CACI indicates the application of transitions between these situations. For example, CACI notifies the application of a failed binding or QoC level transitions. It is the responsibility of the application developer to implement these behaviours and coordinate changes in behaviour based on the indicated situation changes.

Although the number of possible behaviours, which an application developer has to take into account, can become large depending on the number of required context types and quality levels, the application design becomes more realistic. Furthermore, not all possible behaviours are unique and may overlap with other behaviours, depending on the application requirements. For example, the number of possible behaviours for the healthcare centre application is 38, from which after design 18 behaviours remain. Additionally, these behaviours are not independent and are extensions of each other. Further research is required to determine the best way to design and implement these behaviours and to create a coordinator that facilitates the enabling of these behaviours.

For specifying the context requirements of a context-aware application, CACI offers a simple descriptive language expressed in XML. The patient and caregiver side application each need approximately 7 lines of simple CBDL XML description while for the healthcare centre-side application 40 lines of XML code are needed. Additionally, all three sides need between 5-10 lines of Java code to basically embed CACI in their application and retrieve the context information. In total this results in 74 lines of code to create and maintain context bindings of the three application parts which make up the ESS system (see *Table 8-20*).

To estimate the code a developer has to produce for similar context binding functionality without CACI, we use an Eclipse Metrics plug-in[18] to determine the lines of code of CACI's Binder, Discover Manager, Monitor and Decider functional blocks. Together these blocks consist of approximately 497 lines of code which have to be replicated and specialized for the three different parts. In total this would result in approximately 1497 lines of code. Hence, a possible large code reduction can be achieved when using CACI.

Table 8-20 Lines of code of the different ESS application parts.

LOC / Application part	CBDL	Context logic	CACI
Healthcare centre	40	10	497
Patient	7	5	497
Caregiver	7	5	497
Totals	54	20	1491

Developing a CACI-based context-aware application, the developer has to learn how-to use CACI. We do not think this presents a serious drawback, for two reasons. First, CACI provides simple interfaces and XML schemas to ease this process. Possible future extensions could include developing a GUI that enable developers to graphically generate CBDL descriptions and generate CACI integration code as part of the context retriever. Second, context-aware application development without CACI also requires similar or more learning effort to cope with underlying discovery mechanism.

Summary based on ISO/IEC 9126

Finally, we end this discussion with a general summary on the potential improvement of the development process of context-aware applications, using the ISO/IEC 9126 standard. *Table 8-21* presents this summary. Appendix C gives an overview of the definitions of the different ISO/IEC 9126 characteristics.

Table 8-21 expected improvement of the development process of context-aware applications.

Characteristics	Improved	Explanation
Functionality		
-- Suitability	●	CACI relieves the application developer from developing code for creating and maintaining context bindings, hence he can focus on his primary task of creating application logic.

[18] http://metrics.sourceforge.net/

-- Accurateness	●	By using the guidelines as presented in Chapter 4, the application developer becomes more aware in his design, of the situations in which his context-aware application operates (i.e. availability of context and QoC levels). Hence his application can become more ‘accurate’ to the real world situations.
-- Interoperability	●	The discovery interoperability mechanism enables the application to transparently retrieve context information from domains that offer heterogeneous context discovery mechanisms.
-- Compliance	□	The CACI prototype complies with the OSGi specification. The used OSGi component framework offers a component-based approach of creating context-aware applications.
-- Security	-	Interoperating with different context discovery systems from different domains by downloading domain specific discovery adapters forms a security risk. Further research is needed to limit/overcome this risk.
Reliability		
-- Maturity	●	Incorporating in the design and implementation of the context-aware application that context information can become unavailable and quality criteria cannot be met, results in a more mature application.
-- Fault Tolerance	●	Unavailability of context sources leads to a rebinding process by CACI.
-- Recoverability	●	One of the main features of CACI is to re-bind to other context sources in case of them disappearing or decreasing QoC of the context information they offer.
Usability		
-- Understandability	n/a	CACI does not directly influence the understandability of the resulting context-aware application.
-- Learnability	n/a	CACI does not directly influence the learnability of the resulting context-aware application.
-- Operability	n/a	CACI does not directly influence the operability of the resulting context-aware application.
Efficiency		
-- Time behaviour	-	Adding CACI introduces a level of indirection that creates performance overhead. However, as discussed in Chapter 5 and 6 this overhead is limited.
-- Resource behaviour	-	Adding CACI introduces a level of indirection that creates resource overhead. However, as discussed in Chapter 5 and 6 this overhead is limited.

Maintainability		
-- Analyzability	●	By using CACI, the context logic of the application decreases. Hence the overall application becomes less comprehensive and possibly better understandable for the application developer.
-- Changeability	●	By separating the application from the context requirements using the CBDL language, the application developer can easily change his context requirements without changing the application code.
-- Stability	-	More research is needed to estimate the risk of oscillating behaviour due to the re-binding process and ways to overcome this behaviour.
-- Testability	●	The Simucontext framework can be used to test the application against simulated context sources.
Portability		
-- Adaptability	●	The discovery interoperability mechanism enables the application to transparently retrieve context information from domains the application moves through.
-- Installabilitty	●	By using a standard component framework as the foundation of CACI, the component framework lifecycle capabilities can be used to easily install and update applications.
-- Conformance	□	OSGi is a Java based technology suitable on multiple hardware platforms such as laptop's and pda's.
-- Replaceability	n/a	-

Legend: expected improvement of the development process of the context-aware application, '●', improves, '□' similar ,'-' worsened, n/a not applicable

Chapter 9

Conclusions

This chapter presents the conclusions of this thesis and identifies topics that we believe are relevant for future research. This chapter is structured as follows: Section 9.1 presents general considerations on our research in relation to the context-awareness domain. Section 9.2 discusses the main research contributions. Section 9.3 discusses the extent to which we have answered our initial research questions. Finally, Section 9.4 presents directions for future research.

9.1 General Considerations

The world is increasingly equipped with high-capacity, interconnected, mobile and embedded computing devices. Context-awareness provides an attractive approach to personalize applications such that they better suit the user's needs in such a ubiquitous computing environment.

Context-awareness is a comprehensive and challenging research area. In (Wac, Broens et al. 2008) we map the context-awareness research domain by identifying and categorizing relevant research topics. We distinguish a multitude of research topics related to context-awareness, such as theoretical foundations, context management, security, context reasoning, etc. The AWARENESS project deals with a subset of these topics and focuses on an infrastructure that enables rapid and easy development of context-aware applications in a secure and privacy-conscious manner. Based on this research, we discuss general lessons learned on developing context-aware mobile applications (Wegdam, Broens et al. 2008). These lessons include, amongst others, that different application environments require different context management solutions, the importance and influence of Quality of Context on the functioning of context-aware applications, use of statistical and rule-based methods for context reasoning to provide more or better context information , and how to deal with the trade-off between the user privacy and context-awareness.

This thesis focuses on an important aspect related to the development of context-aware applications. We researched ways to facilitate the exchange of context information required for the execution of context-aware applications. The exchange of context information requires a context binding between a context consuming context-aware application and suitable context producing context sources.

In this thesis, we have argued that developing context-aware applications is a challenging task. Especially, developing mechanisms for creating and maintaining context bindings is complex. This justifies the development of abstractions and infrastructure-based mechanisms to support developers of context-aware applications in creating and maintaining context bindings. Hence, we have developed: (i) an abstraction, coined the Context Binding Transparency, which hides the complexities of creating and maintaining context bindings for the application developer, and (ii) a context binding infrastructure, coined CACI, that realizes this transparency. We developed a proof-of-concept prototype of CACI.

Additionally, we argue that the way context-aware applications should use context information to adapt, and how they should deal with varying quality and (un)availability of context information, remains mainly a responsibility of the application developer. However, we claim that a context binding infrastructure can support the application developer to deal with these development aspects. When using a context binding infrastructure the application developer can better focus on his core task: developing the application logic of its application.

We used the CACI prototype for evaluating possible improvements regarding the development process of context-aware applications. This evaluation made plausible that the development process of context-aware applications can indeed be improved by using our context binding infrastructure. First, we showed, based on the results of a survey, the general interest of application developers in a context binding transparency and a context binding infrastructure. Secondly, we showed the feasibility of creating a context-aware application using CACI, by implementing a telemedicine case based on the CACI development guidelines and prototype. Thirdly, we presented the implementation of the same case using an existing context discovery middleware and we qualitatively compared the development effort and software quality of both case studies. This comparison showed that for the given case, CACI provides an infrastructure to develop more structured and higher quality applications, which requires less development effort. Finally, we made an overall analysis of the capabilities of CACI and the impact of CACI on the development of context-aware applications. This analysis showed that it is reasonable to expect that the use of CACI results in less development effort per application and higher quality applications. Additionally, it showed that CACI can provide improvements with respect to the majority of software quality characteristics specified in the ISO/IEC 9126 standard.

9.2 Research Contributions

The research presented in this thesis addresses the design and implementation of context-aware applications using infrastructure-based support mechanisms. Our main contributions are:

– definitions, concepts and models;
– a context binding transparency;
– a context binding infrastructure;
– an analysis of the Telemedicine domain.

Definitions, Concepts and Models

In this thesis several definitions, concepts and models are developed, which give general insights in context-awareness and (context) middleware.

In Chapter 2 and 3, we present basic concepts, terminology and models, and the state-of-the-art on context middleware. Specifically, the following contributions are made:

– Definitions of context, context information and context-awareness. These definitions stress the importance and implications of some underexposed inherent characteristics of context information: context information is offered by context sources which can become (un)available and that context information is offered with a certain QoC.
– A generic architectural model of context-aware applications, which distinguishes the basic functions encompassed in context-aware applications. Additionally, we identify the role and architectural position of a context binding infrastructure that supports context-aware applications to create and maintain context bindings.
– Model of a context binding process. We identify the phases and capabilities in a comprehensive context binding process, which are required for creating and maintaining context bindings. Additionally, we categorize current context middleware in terms of the context binding capabilities required to support this process.

Context Binding Transparency

In Chapter 4, we discuss transparencies and describe the context binding transparency. This transparency proposes an implementation independent specification of infrastructure-based context binding functions that mask the complexities of creating and maintaining context bindings for developers of context-aware applications. Specifically, the following contributions are made:

- Discussion of the concepts 'transparency' and 'context binding transparency', which results in general insights on applying transparencies in the development process of context-aware applications. In addition, the relation of the context binding transparency with the distribution transparencies as defined in the ODP reference model is explained.
- Specification of the context binding transparency in terms of context retrieval and publishing services. These services can be used by application developers to retrieve required context information for their application without being aware of the creation and maintenance process of the required context bindings.
- Definition of a context requirement specification language, coined Context Binding Description Language (CBDL) that enables application developers to specify their context requirements at a high-level of abstraction rather then in programming code.
- Development guidelines that application developers can use to develop a context-aware application, which is based on a context binding infrastructure that realizes a context binding transparency. In addition, a generic discussion is given on the development process of context-aware applications. This discussion stresses the importance of distinguishing situations in the application design in which no or low quality context information is available.

Context Binding Infrastructure

In Chapter 5 and 6, we discuss the design and implementation of the context binding infrastructure. This infrastructure realizes the Context Binding Transparency. Specifically, the following contributions are made:

- Overview of the generic structure of our context binding infrastructure, coined Context-Aware Component Infrastructure (CACI), which is based on the component-based middleware paradigm.
- Design and prototype implementation of a mechanism to support developers in retrieving context information based on CBDL specifications, coined the context binding mechanism.
- Design and prototype implementation of a mechanism to enable the context binding mechanism, or individual applications, to transparently interoperate with available context discovery mechanisms, coined the context discovery interoperability mechanism.
- Design and prototype implementation of a context simulation programming framework, coined SimuContext, which enables developers to configure/program simulated context sources. Integration of the SimuContext framework with the CBDL language and CACI

infrastructure to extend the support for a complete development life-cycle of a context-aware application.

In Chapter 8, we determine if the development process of context-aware applications is facilitated by using the proposed context binding infrastructure. Specifically, the following contributions are made:

- A user expectation survey, which rates the expected usefulness of a context binding infrastructure by potential application developers.
- Implementation of an elaborated telemedicine case study using CACI, to illustrate the feasibility of the proposed context binding infrastructure.
- Implementation of the same telemedicine case study using a current context discovery mechanism. This is done to compare the development effort of the development processes and the software quality of the resulting applications with/without using CACI.
- Overall analysis on the capabilities of CACI and the impact of CACI on the development of context-aware applications.

Telemedicine Domain Analysis

In Chapter 7, we give an extensive overview of the telemedicine domain. Specifically, the following contributions are made:

- A model that identifies determinants, which influence the success of telemedicine applications. Additionally, these determinants are related to the life-cycle of these applications.
- Discussion on the relevance and possible usefulness of context-awareness for telemedicine applications. This analysis showed that using context information to provide the right medical information at the right time can be beneficial for the quality of a telemedicine application. Especially, for applications used in emergency situations this potentially offers benefits.

9.3 Reflection on the Research Questions

In this section, we reflect on the research questions, as introduced in Chapter 1.

1. How do context-aware applications differ from non-context-aware applications and how does this influence the development process of these applications? How does the proposed context binding transparency influence the design of context producers and consumers?

Context-aware applications use context information to adapt their behaviour to offer a higher quality service to their users. The application logic of a context-aware application reacts on context inputs additional to application inputs. Context input differs from application inputs because they deal with context information rather than application data. Context information is not required for the application to function while application data is. When context information is available to the application, the application can use it to offer a higher quality service. Else it offers default behaviour. Furthermore, context information describes the situation of an entity. How well it describes this real-world situation is captured in its Quality of Context (QoC). Context information is typically offered by third party context sources while application data can also be provided by the application users. Due to the dynamic nature of these context sources, context information is arbitrarily available. Both the dynamic availability and fluctuating quality influences in what way the context-aware application can tailor its behaviour to the situation of the user.

Hence, context-aware applications consist of, besides application logic, context logic to acquire and process context information coming from context sources (see Chapter 2). We argue that the application logic of a context-aware application exhibits a default behaviour that adapts based on context information itself, but should also adapt based on the availability and quality of this context information.

Although, we develop a transparency and infrastructure mechanism to improve the continued availability of high quality context information, there may still be situations in which such context information is not available. Hence, application developers should develop a context-aware application considering also the unavailability of context information by developing a basic context-unaware behaviour that is extended with context-aware behaviour in case of available context information (see Chapter 4).

The context binding transparency hides for application developers some of the complexities of creating and maintaining context bindings. We claim that context bindings do not have to be programmed by application developers but can be generated based on their context requirements. Hence, the proposed context requirement language (i.e. CBDL, see Chapter 4) enables application developers to focus on their primary task of developing application logic. The proposed context binding mechanism uses the context requirement specification to create and maintain context bindings. The development guidelines presented in Chapter 4 illustrate the process of using the CBT and the context binding mechanism to create a context-aware application.

2. What context requirements can application developers have? What elements are needed in a context requirement specification language such that application developers are able to specify context requirements suitable for their context-aware applications?

As part of the context binding transparency, the application developer has to specify his context requirements when using the context retrieval service to retrieve context information. We proposed a language, coined the *Context Binding Description Language* (CBDL), to enable application developers to specify their context requirements at a high level of abstraction rather than in programming code. In this way, the specification of context requirements and the implementation of these requirements in context logic is separated from the development of the actual application logic.

Based on an analysis of current context middleware infrastructures and case studies (see Chapter 3), we analyzed the type of context requirements an application developer of a context-aware application can have. This analysis has led to the development of the CBDL language meta-model (see Chapter 4).

A CBDL document consists of three types of information: (i) context specification, (ii) quality criteria and (iii) binding options. Context specifications consist of the basic information required by a context-aware application such as context type, the entity from which the context information describes a situation, and the required format. The quality criteria specifications consist of a combination of maximal cost criteria and the (minimal) required quality level of the required context information. These quality levels are specified by QoC parameters adopted from literature, such as freshness, precision, probability of correctness, spatial resolution and temporal resolution. Finally, the binding options consist of binding preferences used to configure the context binding process.

The CACI prototype can handle XML-based CBDL documents (see Chapter 5). The language is used in the evaluation to express the context requirements of the different application parts from the telemedicine case study (see Chapter 8). This showed that CBDL is capable of expressing the context requirements of a semi-realistic application and that the document can be used to create and maintain context bindings.

3. What operational interfaces should a context binding mechanism offer, such that application developers can deploy and test their context-aware applications?

In Chapter 4, we model the operational interfaces of the context binding infrastructure in terms of the context retrieval and publishing services. These services realize the context binding transparency. The developer of the context binding infrastructure is confronted with a trade-off between the amount of hiding his system can perform and the possibility for control it still offers to the application developer. Assuming on the one hand, the more the problem of creating and maintaining context bindings is hidden for the application developer, the easier the development process for the application developer becomes. However, on the other hand, the more complex the infrastructure becomes, this may introduce performance overhead, security risks or other unwanted effects. Additionally, the application developer may still require a form of control to fulfil its application specific needs, such that complete hiding of the problem of creating and maintaining context bindings is unwanted.

The primitives of the context retrieval and publishing services can roughly be categorized in primitives that can be used to: (i) create and destroy bindings, (ii) retrieve and publish context information, in a request-response or subscribe-notify manner, and (iii) notify the status of the binding.

Internally, the proposed context binding infrastructure adopts a component-based middleware approach (see Chapter 5). Besides the modular development of application components and potential improved reuse of these components, it also enables initialization of context bindings at deploy time. At deploy-time of an application component, the incorporated CBDL document is used to create an initial context binding. Hence, the create binding primitives of the context retrieval and publishing services are implicitly invoked by the application developer when deploying the context-aware application components. The destroy binding primitives are implicitly invoked when un-deploying the component. Context information retrieval and publishing has to be performed explicitly in the application logic of the components by invoking the middleware services of the context binding mechanism.

Testing of the context-aware application can be performed in two ways by using the context binding infrastructure: (i) debugging the notification mechanisms of the context binding mechanisms and (ii) using the developed context simulation framework (see Chapter 6) to test the application logic against simulated context sources. For the latter, we developed an extension to the CBDL language to specify configuration parameters of the context sources to be simulated. At deploy-time the context binding mechanism automatically generates simulated context sources that can be bound to the context-aware application. Additionally, the simulation framework offers a simulated context source configuration, registration and retrieval service such that it can be used independently from the context binding mechanism.

4. How configurable should a context binding mechanism be to enable application developers to develop flexible context-aware applications?

Although creating and maintaining context bindings can be handled in a generic manner by an infrastructure-based context binding mechanism, it may still require certain application specific configuration. To enable the context binding mechanism to be tailored to application specific requirements, the context binding mechanism has been developed in a modular fashion using a component-based middleware approach (see Chapter 5). The binding mechanism can be tailored by enabling the addition of application specific plug-ins. The following types of plug-ins are supported: (i) deployment interceptors which intercept the deployment of incoming components in specific component frameworks, (ii) parsers, which parse different types of context specification languages, (iii) selectors, which select suitable context sources using application specific selection algorithms and (iv) deciders, which decide when to re-bind. Configuration of the infrastructure is done at start-up based on configuration parameters specified by the application developer.

In the prototype, we used the OSGi framework as the underlying component framework and implemented specific deployment interceptors for this framework. Additionally, we developed a parser for the CBDL language, created a simple syntactic selector and developed a simple re-binding algorithm.

5. How can a context binding mechanism create a suitable context binding based on a context requirement specification?

The structure and behaviour of the context binding mechanism is explained in Chapter 5. The design facilitates a complete binding process, as explained in Chapter 2, ranging from discovery, selection and association to monitoring and releasing.

The context requirement specification is bundled together with the application component. The deployment of the component has to be intercepted by the specific deployment interceptor. The specification is extracted from the component and parsed by the specific parser. A context requirement specification can consist of multiple context requirements, which have to be distilled. There are two types of context requirements: (i) context retrieval requirements for context consuming components, and (ii) context publishing requirements for context producing components.

Every context publishing requirement results in advertisement of the context offerings of the component in a local context source repository.

Every context retrieval requirement results in a context discovery request to the available context discovery mechanisms. The results retrieved from the context discovery mechanisms are collected and a selection of a suitable context source is made by the specific selector. A corresponding context producer proxy is generated, which acts as a middleman between the application component and the selected context source. This proxy is monitored for changing availability of the context source and the quality of the context information.

6. How can a context binding mechanism maintain a created context binding in an environment where context producers can appear, disappear, and have fluctuating quality?

Based on analysis of context-aware applications, we distinguish four situations in which an established context binding becomes less valid for the context-aware application (see Chapter 5): (i) the bound context source becomes unavailable due to a de-registration of the context source at the context discovery mechanism, (ii) a new context source appears with a higher QoC offering due to a registration at the context discovery mechanism, (iii) on retrieval of context information by the context-aware application there is a retrieval exception and (iv) the actual QoC of the retrieved context information degrades below the required QoC.

These situations should be recognized by a context binding mechanism and a re-binding decision process should be started. When a context source becomes unavailable (i.e. situation i, iii, iv) possibly a complete new context binding process (see Chapter 2) should be started, beginning from a new context source discovery phase. When a new context source becomes available (i.e. situation ii) the context binding process can be shortened by starting from the selection phase in which the new source is compared to the already bound context source.

In the present research we focussed on the first three situations. Context discovery mechanisms can notify the context binding mechanism of (de-)registration events (situation i and ii) by using the notify primitives offered by the context retrieval and publishing services. Successively, a discovery and/or selection process is started inside the context binding mechanism. When a context retrieval error occurs (situation iii) this is hidden for the application component by the context producer proxy. According to a re-binding decision algorithm, the proxy notifies the context binding mechanism to trigger a new context binding process, starting from the discovery phase. Optimizing the algorithm to decide if and when to start the re-binding process is out of the scope of this research and needs more research. Also degrading actual QoC (situation iv) could be recognized by the context producer proxy. However more research has to be done on this aspect.

7. How can a context discovery interoperability mechanism deal with multiple heterogeneous and dynamically available context discovery mechanisms offering context producers?

A context-aware application of a mobile user encounters different administrative environments in its life-span. These environments might provide different heterogeneous context discovery mechanisms. Hence, to get context information, the applications need to interoperate with different dynamically available context discovery mechanisms.

By taking a client-side approach for the context binding infrastructure, we enable a homogenizing approach for interoperating context-aware application with context discovery mechanisms (see Chapter 5). This approach proposes a context discovery interoperability mechanism that acts as a single point of access for context source discovery to the context-aware application (see Chapter 6). The mechanism coordinates the discovery process between the context-aware application and dynamically available context discovery mechanisms. Hence, it offers a common context discovery interface to the context-aware application. It offers a context discovery adapter interface to the context discovery mechanisms. Infrastructure developers create specific adapters for their discovery mechanism conforming to the adapter interface. By running a discovery adapter supplier in the network of the specific discovery mechanism, the discovery interoperability mechanism can dynamically download and plug-in the specific discovery adapters. These adapters are responsible for translating the specific discovery request and results to ones which can be handled passed to the application.

8. How can the telemedicine domain benefit from context-aware applications? How can the context binding infrastructure be used for developing context-aware telemedicine applications?

Several social-economical trends stimulate the increasing use of ICT in healthcare. Amongst others, telemedicine applications, which are applications that have as goal to provide healthcare and sharing of health data over distance using ICT, are promising to enhance the future healthcare. In Chapter 7, we give a domain analysis of the telemedicine domain. One trend in healthcare is the drive for patient-centric healthcare. Context-awareness provides an opportunity to tailor the provided healthcare to the situation of the user and make them more patient-centric. Furthermore, due to efficiency and cost reasons, there is an increasing trend of extramural care. By adapting applications to the patient's home situation, a feeling of comfort and safety can be created which could improve patient treatment and recovery.

Additionally, based on an extensive literature study, we present a model containing generic determinants that influence the success of telemedicine application. These determinants are categorized in the following categories: technology, acceptance, organization, financing and, policy and legislation. Additionally, context-awareness can play a role for improving the technology and user acceptance aspects of telemedicine applications. Literature indicated that it is important for the success of telemedicine applications that the right information is available, at the right time using the right communication modality, where 'right' refers to the user's need. We present multiple examples of the use of context in a generic telemedicine process.

Finally, we identified that especially for emergency situations, the context of a patient is particularly important. By having context information, such as location of the patient and availability of caregivers, the time between the occurrence of the emergency and treatment of the patient might become shorter and the availability of the right information can become better. Hence, the 'golden hour', which is the first hour of a patient after an emergency that highly influences the recovery of the patient, could be used more efficiently.

9.4 Future Research

Throughout this thesis, we give specific directions for future work concerning: the CBDL language (Chapter 4), the CACI infrastructure (Chapter 5), the context binding mechanism (Chapter 5), the context discovery interoperability mechanism (Chapter 6) and the SimuContext framework (Chapter 6). Below, we summarize these directions and we present some more general issues in context-awareness:

- *Reasoning:* The current context binding infrastructure can only create a context binding in case there is a precise match between the requirements of a context-aware application and the offering of a context source. However, when the required context information cannot be offered by any single context source, no context binding can be established. We propose to investigate reasoning techniques to improve upon this situation. For example, the context binding mechanism could be extended with vertical context reasoning techniques that can infer the required context information, by combining information originating from multiple different context sources. Additionally, horizontal context reasoning techniques could be used to maintain the quality level of context information.
- *Privacy:* The use of context information violates a user's privacy when it this is done for unwanted purposes. It is important for users to be able to control, who, when and how their context information is used. Hence, we propose to investigate how to enable users to specify and control their privacy policies. Furthermore, future research is how the context binding infrastructure could be enhanced with privacy enforcement functions that enforce the specified policies. For developers, a possible way to specify privacy policies is by extending the CBDL language to include privacy constructs. The binding mechanism

could be extended to enforce these CBDL-based policies. However, this requires more research.

- *Re-binding decision algorithm:* Besides the static QoC offering of a context source, the actual QoC of the provided context information determines the usefulness of that context information for the context-aware application. Additionally, context retrieval may fail due to unavailability or a failure state of the bound context sources. Both situations could be transitory and unconditional rebinding might result in undesirable or inefficient behaviour. Both situations require a re-binding decision algorithm to determine the optimal time to start re-binding. We propose to investigate such re-binding algorithms. In this thesis, we present a simple re-binding algorithm, which is merely an example of such an algorithm.
- *Semantic interoperability:* The current context binding mechanism syntactically matches context offerings of context sources with context requirements posed by the application developer. Besides syntactic interoperability, the extent to which semantic interoperability is realized influences the quality of this matching process. We propose to research mechanisms to semantically specify and match context offerings and context requirements.
- *Development of context sources:* The vision of a ubiquitous environment in which a rich spectrum of context sources is embedded in the environment and available to context-aware applications, is not yet realized. Before deploying context-aware applications, a considerable development effort has to be put into the challenging task of encapsulating common-of-the-shelf sensors into more generic context sources or transforming information sources into context sources. The majority of these sources/sensors have proprietary ways to retrieve (context) information. More research is needed on standardized ways to retrieve (context) information. This might include researching a standardized context model and retrieval API's.
- *Business models:* Context-awareness offers opportunities for a novel type of commercial applications. On the one hand this creates possible business opportunities, on the other hand current business models might have to be adapted to accommodate this new type of application. We estimate a more complex value chain with the introduction of third party context providers. Research has to be performed on who can benefit from context-aware applications, business models that facilitate this value chain and technical solutions to support these business models.

Appendix A

User Expectation Survey

This appendix contains the questionnaire, which is used both as a paper and web-based version, used in the user expectation survey. The results of this questionnaire are discussed in Chapter 8.

Questionnaire

Towards a Context Binding Transparency

User Expectation Survey

1. Do you perform research in the area of context-awareness or related areas (e.g. ubiquitous, pervasive computing, ambient intelligence)?

Yes ☐ No ☐

2. In what area do you perform this research?

____

3. Have you ever developed a (context-aware) software application?

☐ Yes, I developed context-aware applications.
☐ Yes, I developed non-context-aware applications.
☐ No, but I am planning to.
☐ No and I am not planning to.

4. Have you ever used middleware (e.g. context discovery) to develop (context-aware) applications?

Yes ☐ No ☐

5. What specific type of middleware have you used before (e.g. corba, web services, context management, service discovery)?

____

6. Do you think the proposed context binding transparency can simplify the development of context-aware applications (1 = not at all, 5 = very much)?

Don't know ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

7. How useful is the specification of context requirements in a specification language and resolving of the requirements in the binding middleware, rather than programming this in the application (1 = not at all, 5 = very much)?

Don't know ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

8. How useful is the automatic adaptation to the availability and quality of context sources by the binding middleware (1 = not at all, 5 = very much)?

Don't know ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

9. How useful is the automatic interoperability between context discovery mechanisms by the binding middleware (1 = not at all, 5 = very much)?

Don't know ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

10. What aspects do you think will influence the success of the context binding transparency (e.g. learning curve, performance)?

____

11. Other remarks:

____

____

____

____

Appendix B

CBDL Use Cases & Implementation

This appendix describes some of the use cases that are used for the requirement analysis of the CBDL language. Additionally, it provides details on the realization of CBDL using a XML Schema.

B.1 Use-cases

(i) Healthcare use-case: Epilepsy Safety System (ESS)

The ESS monitors vital signs of epilepsy patients and determines upcoming epileptic seizures. When a likely seizure is detected, the system notifies nearby and available caregivers with instructions on the location (e.g. in lat/long context format) of the patient and route information to the patient. The application uses context information on the location of the patient and the caregiver and context information on availability of the caregivers to provide this functionality. The quality of the location data of the patient should have a minimal precision of 5m (i.e. the specified location of the patient may differ 5m from the actual location) to be able to dispatch caregivers to the right location. The location data of caregivers only has to be minimally 100m precise to be able to determine which one is nearby.

Additionally, the vital signs of the patient are transferred to the healthcare centre where care professionals monitor the patient's state and stays in contact with the dispatched caregiver. Context information on the available bandwidth (e.g. in kb/s) of the patient's device is used to tailor the granularity of transferred vital signs (e.g. increase or decrease sample frequency) and the amount of vital signs (e.g. decrease the number of send channels) to ensure transfer of vital signs to the healthcare centre.

(ii) Office use-case: My idea recorder (MIR)

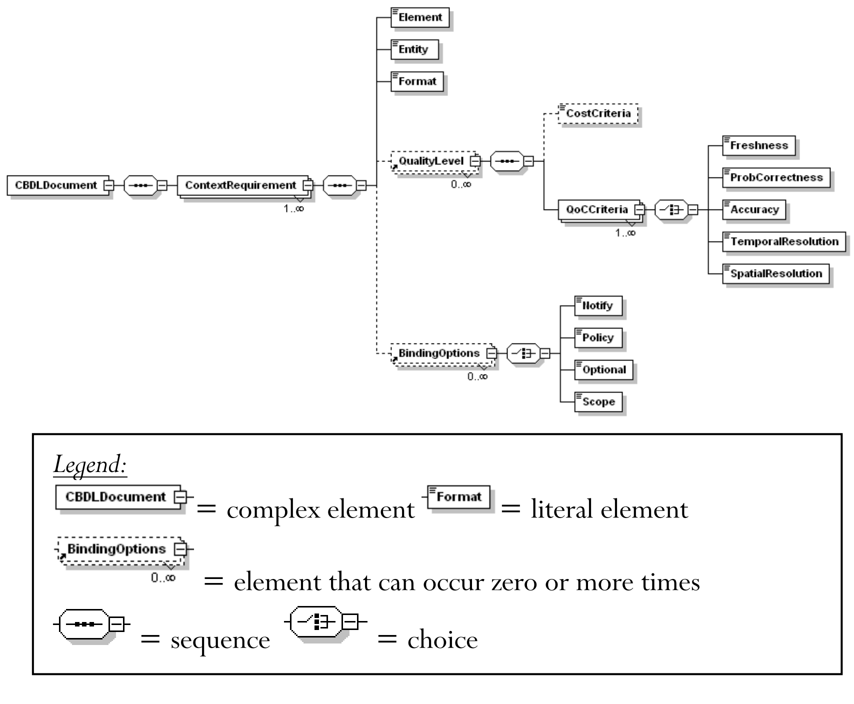
During meetings, users can use their camera phones to take high-resolution pictures of whiteboard sketches to capture their ideas for future use. The

MIR system distributes copies of these pictures to meeting participants. The phone automatically determines the persons that are currently in the meeting based on meeting information (e.g. in Boolean context format) from user's calendars and nearby Bluetooth devices. When the meeting information is not at least 75% correct (i.e. probability of 75% that the participant is actually in/out a meeting), the application asks the participant if he is in the meeting. The system delays the data transfer until an adequate network becomes available (i.e. GPRS, UMTS, WLAN or Bluetooth) taking into account the cost and bandwidth characteristics of each network type and the battery status of her phone.

B.2 CBDL XML Schema

We implement the CBDL language using XML. Hence, we define a XML Schema to specify the structure of the CBDL language. *Figure B-1* gives a graphical overview of the XML schema derived from the CBDL meta-model (see Chapter 4). All the classes in the meta-model map to xml elements in the schema definition. The attributes in the meta-model map to xml attributes of the corresponding xml elements (however not visible in the figure). The granularity of the relationships between classes in the meta-model are mapped to occurrences (min/max) in the schema definition. For association relationships in the meta-model, sequences are used in the schema definition. For inheritance relationships in the meta-model, choices in the schema definition are used.

Figure 9B-1 XML Schema of the CBDL language



Appendix C

ISO/IEC 9126 Standard

This appendix discusses software quality characteristics as proposed by the ISO/IEC 9126 standard. This standard has as goal to provide a framework for the evaluation of software quality. *Table C-1* presents the software quality characteristics (and corresponding sub-characteristics) and their definition. These definitions are reproduced from the ISO/IEC 9126-1 standard (ISO/IEC 2001) based on [19] and [20].

Table C-1 ISO 9126 software quality characteristics

Characteristics	**Definition**
Functionality	"A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs."
-- Suitability	"Attributes of software that bear on the presence and appropriateness of a set of functions for specified tasks."
-- Accurateness	"Attributes of software that bear on the provision of right or agreed results or effects."
-- Interoperability	"Attributes of software that bear on its ability to interact with specified systems."
-- Compliance	"Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions."
-- Security	"Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions."
Reliability	"A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time."

[19] http://www.cse.dcu.ie/essiscope/sm2/9126ref.html
[20] http://en.wikipedia.org/wiki/ISO_9126

-- Maturity	Attributes of software that bear on the frequency of failure by faults in the software.
-- Fault Tolerance	"Attributes of software that bear on its ability to maintain a specified level of performance in case of software faults or of infringement of its specified interface."
-- Recoverability	"Attributes of software that bear on the capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it."
Usability	"A set of attributes that bear on the effort needed for use, and on the individual assessment of such use by a stated or implied set of users."
-- Understandability	"Attributes of software that bear on the users' effort for recognizing the logical concept and its applicability."
-- Learnability	"Attributes of software that bear on the users' effort for learning its application."
-- Operability	"Attributes of software that bear on the users' effort for operation and operation control."
Efficiency	"A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions."
-- Time behaviour	"Attributes of software that bear on response and processing times and on throughput rates in performances its function."
-- Resource behaviour	"Attributes of software that bear on the amount of resource used and the duration of such use in performing its function."
Maintainability	"A set of attributes that bear on the effort needed to make specified modified modifications."
-- Analyzability	"Attributes of software that bear on the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified."
-- Changeability	"Attributes of software that bear on the effort needed for modification, fault removal or for environmental change."
-- Stability	"Attributes of software that bear on the risk of unexpected effect of modifications."
-- Testability	"Attributes of software that bear on the effort needed for validating the modified software."
Portability	"A set of attributes that bear on the ability of software to be transferred from on environment to another."
-- Adaptability	"Attributes of software that bear on the opportunity for its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software considered."

-- Installabilitty	“Attributes of software that bear on the effort needed to install the software in a specified environment.”
-- Conformance	“Attributes of software that make the software adhere to standards or conventions relating to portability.”
-- Replaceability	“Attributes of software that bear on opportunity and effort using it in the place of specified other software in the environment of that software.”

Appendix D

Additional information on the development of the ESS case

This appendix presents additional information on the development of the ESS case, as presented in Chapter 8.

D.1 Context information unavailability tables

Here we present the descriptions of application logic behaviours of the two additional application parts; the patient application and caregiver application.

Table D-1 Application behaviours in case of unavailability of context information for the patient application.

Patient application	
PA-AB	**Application logic behaviour,** *[PA-CAB#] are behaviours adapted from the default behaviour [PA-DB].*
x	[PA-DB]: In case of a detected epileptic seizure, a notification is send to the healthcare centre and the minimal required set of vital signs, measured on the lowest required frequency, is transmitted to the healthcare centre.
v	[PA-CAB1]: In case of a detected seizure, depending on the available bandwidth, the transmitted set of vital signs and the sample frequency is increased.
Legend: x = context information is unavailable v = context information is available, […] id of the behaviour.	

Table D-2 Application behaviours in case of unavailability of context information for the caregiver application.

Caregiver application	
CG-L	**Application logic behaviour,** *[CG-CAB#] are behaviours adapted from the default behaviour [CG-DB].*
x	[PA-DB]: In case of a received notification, a connection is made with the healthcare centre and possibly the patient. When possible a map is shown with the location of the patient.

v	[PA-CAB1]: A map with route information from the location of the caregiver to the location of the patient is shown.
Legend: x = context information is unavailable v = context information is available, […] id of the behaviour.	

D.2 CBDL documents of the patient and caregiver applications

Below the CBDL document of a patient and caregiver application are presented. For every patient and caregiver in the ESS system these documents need to be replicated and customized.

Example D-1 CBDL document of a patient application of patient Tim

```
<?xml version="1.0" encoding="UTF-8"?>
<CBDLDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CBDL-schema.xsd" UserID="Patient.Tim"
ApplicationID="ESS_PatientTim">
    <ContextRequirement BindingID="PA-AB">
        <Element>Bandwidth</Element>
        <Entity>Device.Patient.Tim</Entity>
        <Format>kb/s</Format>
</CBDLDocument>
```

Example D-2 CBDL document of a caregiver application of caregiver John

```
<?xml version="1.0" encoding="UTF-8"?>
<CBDLDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CBDL-schema.xsd" UserID="Caregiver.John"
ApplicationID="ESS_CaregiverJohn">
    <ContextRequirement BindingID="CG-CL">
        <Element>Location</Element>
        <Entity> Caregiver.John</Entity>
        <Format>lat/long</Format>
</CBDLDocument>
```

References

Aarts, E. and S. Marzano (2003). The New Everyday: View on Ambient Intelligence. Rotterdam, 010 Publishers.

Aarts, J. (2006). Towards a Context Discovery Interoperability Mechanism: Achieving interoperability between dynamically available context discovery mechanisms through a homogenising middleware layer. Enschede, University of Twente. MSc.

Abdul-Rahman, A. and S. Hailes (2000). Supporting trust in virtual communities. 33rd Hawaii Internation Conference on System Science (HICSS33). Hawaii.

Ailisto, H., P. Alahuhta, et al. (2002). Structuring Context Aware Applications: Five-Layer Model and Example Case. Workshop in Ubicomp, Gothenburg, Sweden.

Alonso, G., F. Casati, et al. (2004). Web Services: Concepts, Architectures and Applications, Springer.

Anderson, T., T. Roscoe, et al. (2004). "Preventing Internet denial-of-service with capabilities." ACM SIGCOMM Computer Communication Review 34(1): 39-44.

Apache Felix Project. (2006). "Apache Felix Project website." from http://cwiki.apache.org/FELIX/index.html.

Apache Log4J project. (2006). "Log4j Logging Service." from http://logging.apache.org/log4j/docs/.

Apache Webservices project. (2003). "XML-RPC specification." from http://www.xmlrpc.com/spec#update3.

Apache WSIF project. (2006). "Web Service Invocation Framework." http://ws.apache.org/wsif/, from http://ws.apache.org/wsif/.

Baldauf, M., S. Dustdar, et al. (2004). "A survey on context-aware systems." International Journal of Ad Hoc and Ubiquitous Computing.

Baldauf, M., S. Dustdar, et al. (2007). "A survey on context-aware systems." International Journal of Ad Hoc and Ubiquitous Computing 2(4).

Banavar, G. and A. Bernstein (2002). "Software infrastructure and design challenges for ubiquitous computing applications." Communications of the ACM 45(12): 92-96.

Bardram, J. (2004). Applications of Context-Aware Computing in Hospital Work - Examples and Design Principles. ACM Symposium on Applied Computing, Cyprus.

Bardram, J. (2005). The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. Pervasive Computing. Munchen, Germany.

Barton, J. and V. V. (2002). UBIWISE, A Ubiquitous Wireless Infrastructure Simulation Environment. HP Labratory Technical Report HPL-2002-303.

Bazire, M. and P. Brezillon (2005). Understanding Context Before Using It. 5th International and Interdisciplinary Conference CONTEXT 2005. Paris, France.

Bellavista, P., A. Corradi, et al. (2003). "Dynamic Binding in Mobile Applications." IEEE Internet Computing March-April: 34-42.

Benerecetti, M., P. Bouquet, et al. (2000). "Contextual Reasoning Distilled." Journal of Expirimental Artificial Intelligence 12(3): 41-67.

Benz, H., C. Hesselman, et al. (2006). Context Discovery and Exchange. Freeband AWARENESS Dn2.1. P. Pawar and J. Brok.

Benz, H., C. Hesselman, et al. (Freeband AWARENESS Dn2.1, 2006). Context Discovery and Exchange. Freeband AWARENESS Dn2.1. P. Pawar and J. Brok.

Berg, M. (1999). "Patient care information systems and health care work: a socialtechnical approach." International Journal of Medical Informatics 55: 87-101.

Bernstein, P. (1996). "Middleware: A Model for Distributed System Services." Communications of the ACM 39(2): 86-98.

Birrel, A. and B. Nelson (1984). "Implementing Remote Procedure Calls." ACM Transactions on Computer Science 2(1): 39-59.

Blackstock, M., R. Lea, et al. (2006). Towards Wide Area Interaction with Ubiquitous Computing Environments. 1st European Conference on Smart Sensing and Context (EuroSSc'06), Enschede, the Netherlands.

Blair, G. and J. Stefani (1998). Open Distributed Processing and Multimedia, Addison-Wesley.

Bohn, J., V. Coroama, et al. (2005). Social, Economic, and Ethical Implications of Ambient Intelligence and Ubiquitous Computing, Springer.

Bottaro, A. and A. Gerodolle (2006). Extended Service Binder: Dynamic Service Availability Management in Ambient Intelligence. International Workshop on Future Research Challenges for Software and Services (FRCSS'06). Vienna, Austria.

Boulos, M. (2003). "Location-based health information services: a new paradigm in personalised information delivery." International Journal of Health Geographics 2(2).

Bradley, N. and M. Dunlop (2003). Towards a multidisciplinary user-centric design framework for context-aware applications. 1st UK-UbiNet Workshop, London, UK.

Brey, P. (2005). "Freedom and Privacy in Ambient Intelligence." Ethics and Information Technology 7(3): 157-166.

Broens, T. (2004). Context-aware, Ontology based, Semantic Service Discovery. Enschede, University of Twente. MSc.

Broens, T. (2005). Supporting the developers of context-aware mobile telemedicine applications. On the Move to Meaningful Internet Systems 2005: OTM Workshops, Ph.D. Student Symposium, Agia Napa, Cyprus, Springer Berlin / Heidelberg, LNCS 3762.

Broens, T. and A. Halteren (2006). SimuContext: Simply Simulate Context. International Conference on Autonomic and Autonomous Systems (ICAS'06). Silicon Valley, USA.

Broens, T., A. Halteren, et al. (2006). Infrastructural Support for Dynamic Context Bindings. 1st European Conference on Smart Sensing and Context (EuroSSc'06), Enschede, the Netherlands, Springer LNCS 4272.

Broens, T., A. v. Halteren, et al. (2007). "Towards an application framework for context-aware m-health applications." International Journal of Internet Protocol Technology (IJIPT) 2(2): 109-116.

Broens, T., R. Huis in't Veld, et al. (2007). "Determinants for successful telemedicine implementations: a literature study." Journal of Telemedicine and Telecare 13(6): 303-309.

Broens, T., S. Pokraev, et al. (2004). Context-aware, ontology-based service discovery. 2nd European Symposium on Ambient Intelligence (EUSAI'04). Eindhoven, the Netherlands, Springer Lecture Notes 3295.

Broens, T., R. Poortinga, et al. (2007). Interoperating Context Discovery Mechanisms. 1st Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC'07), Barcelona, Spain.

Broens, T., D. Quartel, et al. (2007). Capturing Context Requirements. EuroSSC'07. Kendall, England.

Broens, T., D. Quartel, et al. (2007). Towards a Context Binding Transparency. 13th EUNICE Open European Summer School, Enschede, the Netherlands, Springer LNCS 4606.

Broens, T., M. v. Sinderen, et al. (2007). Dynamic Context Bindings in Pervasive Middleware. Middleware Support for Pervasive Computing Workshop (PerWare'07). White Plains, USA.

Broens, T. and A. van Halteren (2006). SimuContext: simulating context sources for context-aware applications. Intl. Conference on Networking and Services (ICNS06), Silicon Valley, USA.

Brok, J. (2006). Cumular Context Sollutions. Freeband AWARENESS Dn2.5.

Brown, P., J. Bovey, et al. (1997). "Context-Aware Applications: From the Laboratory to the Marketplace." IEEE Personal Communications 4(5): 58-64.

Buchholz, T., A. Kupper, et al. (2003). Quality of Context: What it is and why we need it. 10th Workshop of the HP OpenView University Association (HPOVUA03), Geneva, Switzerland.

Bucholz, T., A. Kupper, et al. (2003). Quality of Context: What It Is And Why We Need It. Workshop of the HP OpenView University Association 2003 (HPOVUA 2003). Geneva.

Bunningen, A. v., L. Feng, et al. (2005). Context for Ubiquitous Data Management. International Workshop on Ubiquitous Data Management (UDM'05), Tokyo.

Buschmann, F., R. Meunier, et al. (1996). Patter-oriented software architecture: a system of patterns, Wiley.

Bylund, M. and F. Espinoza (2002). "Testing and Demonstrating Context-Aware Services with Quake III Arena." Communications of the ACM 45(1): 46-48.

Campanelli, P. (2007). "Status of open source OSGi containers." from http://reader.feedshow.com/show_items-feed=e6f4497e0dd7abcc62619492a82ae3e0.

Campbell, R., J. Al-Muhtadi, et al. (2002). Towards Security and Privacy for Pervasive Computing. Theories and Systems, Mext-NSF-JSPS International Symposium (ISSS'02), Tokyo, Japan, Springer.

Cervantas, H. and R. Hall (2004). Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. 26st International Conference on Software Engineering. Edinburgh, Scotland.

Chalmers, M. (2004). "A historical view of context." Computer Supported Cooperative Work (CSCW) 13(3-4): 223-247.

Chan, A., P. Wong, et al. (2004). CRL: A Context-Aware Request Language for Mobile Computing. International Symposium on Parallel and Distributed Processing and Applications (ISPA'04). Hong Kong, China.

Chen, G. and D. Kotz (2000). A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Darthmouth College.

Chen, G. and D. Kotz (2002). Solar: An open platform for context-aware mobile applications. International Conference on Pervasive Computing, Zurich, Zwitserland.

Chen, G. and D. Kotz (2003). Context Sensitive Resource Discovery. First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003). Forth Worth, USA.

Chen, G., M. Li, et al. (2004). Design and Implementation of a Large-Scale Context Fusion Network. First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004). Boston, USA.

Chen, H., T. Finin, et al. (2005). The SOUPA Ontology for Pervasive Computing.

Christie, A. (1999). "Simulation: An Enabling Technology in Software Engineering." CROSSTALK Journal of Defense Software Engineering.

da Costa, C. M., M. da Silva Strzykalski, et al. (2005). A reflective middleware architecture to support adaptive mobile applications. ACM symposium on Applied computing Santa Fe, New Mexico, ACM Press.

Davies, N. and H. Gellersen (2002). "Beyond Prototypes: Challenges in Deploying Ubiquitous Systems." Pervasive Computing 1(1): 26-35.

Dean, K. (2004). Connected Health: essays from health innovators, Cisco though Leaders series.

Dey, A. (2000). Providing Architectural Support for Context-Aware applications, Georgia Institute of Technology. PhD.

Dey, A. and G. Abowd (2000). The Context Toolkit: Aiding the Development of Context-Aware Applications. Workshop on Software Engineering for Wearable and Pervasive Computing. Limerick, Ireland.

Dey, A., J. Mankoff, et al. (2000). Distributed Mediation of Imperfectly Sensed Context in Aware Environments. GVU Technical Report;GIT-GVU-00-14, Georgia Institute of Technology.

Dey, A., D. Salber, et al. (2001). "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications." Human Computer Interaction Journal 16(2-4): 97-166.

Doarn, C., E. Ferguson, et al. (1996). Telemedicine and Telescience in the US Space Program. 20th International Symposium on Space Technology and Science, Gifu, Japan.

Dockhorn Costa, P. (2007). Archtictural Support for Context-Aware Applications: From Context Models to Services Platforms. Enschede, University of Twente. PhD.

Dockhorn Costa, P., G. Guizzardi, et al. (2006). Situations in Conceptual Modeling of Context. Workshop on Vocabularies, Ontologies, and Rules for the Enterprise (VORTE 2006) at IEEE EDOC 2006. Hong Kong.

Dodig-Crnkovic, G. (2002). Scientific Methods in Computer Science. Conference for the Promotion of Research in IT Skovde, Sweden.

Ebling, M., D. Guerney, et al. (2001). Issues for Context Services for Pervasive Computing. Workshop on Middleware for Mobile Computing. Heidelberg, Germany.

Ebling, M., G. Hunt, et al. (2001). Issues for Context Services for Pervasive Computing. Advanced Topic Workshop Middleware for Mobile Computing at the IFIP/ACM Middleware Conference, Heidelberg, Germany.

Emmerich, W., M. Aoyama, et al. (2007). "The impact of research on middleware technology." ACM SIGOPS Operating Systems Review 41(89-112).

Equinox, E. (2006). "Equinox website." from http://www.eclipse.org/equinox/.

Etter, R., P. Dockhorn Costa, et al. (2006). A Rule-Based Approach Towards Context-Aware User Notification Services. International Conference on Pervasive Services (ICPS'06). Lyon, France.

Fenton, N. (1994). "Software Meaurement: A Necessary Scientific Basis." IEEE Transactions On Software Engineering 20(3): 199-206.

Gable, G. (1994). "Integrated case study and survey research methods: an example in information systems." European Journal of Information Systems 3(2): 112-126.

Gamma, E., R. Helm, et al. (1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

Gloss, B. (2005). System Architecture of a Mobile Message Transport System. 11th Open European Summer School - Networked Applications (EUNICE05), Madrid, Spain.

Gokturk, E. (2007). A Minimalistic, Component-Based Approach to Realization of Network Simulators and Emulators. Faculty of Mathematics and Natural Sciences. Oslo, University of Oslo.

Gray, P. and D. Salber (2001). Modelling and Using Sensed Context Information in the Design of Interactive Applications. Engineering for Human-Computer Interaction: 8th IFIP International Conference, EHCI 2001. Toronto, Canada.

Guizzardi, G. (2006). Ontological Foundations for Structural Conceptual Models. EWI. Enschede, University of Twente. PhD.

Helal, S., B. Winkler, et al. (2003). Enabling location-aware pervasive computing applications for the elderly. Pervasive Computing (Percom'03). Dallas, Texas.

Henricksen, K. and J. Indulska (2004). Modelling and Using Imperfect Context Information. Second IEEE Annual Conference on Pervasive Computing and Communications, Workshop on Context Modelling and Reasoning (CoMoRea'04).

Henricksen, K., J. Indulska, et al. (2005). Middleware for Distributed Context-Aware Systems. DOA 2005. Agia Napa, Cyprus, Springer Verlag.

Hesselman, C., H. Benz, et al. (2008). Bridging Context Management Systems for Different Types of Pervasive Computing Environments. First International Conference on MOBILe Wireless MiddleWARE. Innsbruck, Austria.

Hesselman, C., H. Eertink, et al. (2007). Privacy-aware Context Discovery for Next Generation Mobile Services. 3rd SAINT2007 Workshop on Next Generation Service Platforms for Future Mobile Systems (SPMS 2007). Hiroshima, Japan.

Hesselman, C., A. Tokmakoff, et al. (2006). Discovery and Composition of Services for Context-Aware Systems. 1st European Conference on Smart Sensing and Context (EuroSSc'06), Enschede.

Hong, J. (2002). The Context Fabric: An Infrastructure for Context-Aware Computing. Doctoral Workshop, Human Factors in Computing Systems (CHI'02). Minneapolis, USA.

Hong, J. and J. Landay (2004). An Architecture for Privacy-Sensitive Ubiquitous Computing. Proceedings of the 2nd international conference on Mobile systems, applications, and services. Boston, USA.

Hulsebosch, R., A. Salden, et al. (2005). Context Sensitive Access Control. SACMAT'05. Stockholm, Sweden.

IBM (1999). "Pervasive Computing." IBM System Journal 38(4).

ISO/IEC (2001). Software engineering -- Product quality -- Part 1: Quality model. ISO/IEC 9126-1:2001.

Joaquin.net. (2007). "Open Distributed Processing Reference Model." from http://www.joaquin.net/ODP/.

Jones, V., R. Bults, et al. (2001). Healthcare PANs: Personal Area Networks for trauma care and home care. Fourth International Symposium on Wireless Personal Multimedia Communications (WPMC'01). Aalborg.

Jones, V., H. Mei, et al. (2007). Context Aware Body Area Networks for Telemedicine. Pacific-Rim Conference on Multimedia (PCM'07). Hong Kong.

José, R., F. Meneses, et al. (2005). Integrated Context Management for Multi-domain Pervasive Environments. First International Workshop on Managing Context Information in Mobile and Pervasive Environments (MCMP-05). Ayia Napa, Cyprus.

Knoplerfish.org. (2005). "Knoplerfish OSGi website." from http://www.knopflerfish.org/index.html.

Kon, F., F. Costa, et al. (2002). "The Case for Reflective Middleware." Communications of the ACM 45(6): 33-38.

Korkea-aho, M. (2000). "Context-Aware Applications Survey." from http://users.tkk.fi/~mkorkeaa/doc/context-aware.html.

Kranenburg, H. and H. Eertink (2005). Processing Heterogeneous Context Information. Next Generation IP-based Service Platforms for Future Mobile Systems workshop. Trento, Italy.

Kranenburg, H. v., M. Bargh, et al. (2006). "A Context Management Framework for Supporting Context-Aware Distributed Applications." IEEE Communications Magazine 44(8): 67-74.

Kranenburg, H. v., A. Salden, et al. (2005). Grounded Contextual Reasoning enabling Innovative Mobile Services. 5th Workshop on Applications and Services in Wireless Networks (ASWN'05). Grenoble, France.

Kummerfeld, B., C. Quigley, et al. (2003). Merino:Towards an intelligent environment architecture for multi-granularity context description. workshop on User Modelling for Ubiquitous Computing.

kXML project. (2006). "kXML website." from http://kxml.sourceforge.net/.

Lamming, M. and M. Flynn (1994). "Forget-me-not" - Intimate Computing in Support of Human Memory. Int. Symp. on Next Generation Human Interface.

Lee, D. and R. Meier (2007). Primary-Context Model and Ontology: A Combined Approach for Pervasive Transportation Services. Fifth

Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07). New York, USA.

Lee, H., S. Lim, et al. (2005). Design and implementation of baby-care service based on context-awareness for digital home. The 7th International Conference on Advanced Communication Technology (ICACT 2005)

Lehmann, O., M. Bauer, et al. (2004). From home to world - supporting context-aware applications through world models. 2nd IEEE Annual Conference on Pervasive Computing and Communications (PERCOM'04). Orlando, Floriday, USA.

Lerner, E. and R. Moscati (2002). "The Golden Hour: Scientific Fact or Medical "Urban Legend"?" Academic Emergency Medicine 8(7): 758-760.

Lieberman, H. and T. Selker (2000). "Out of context." IBM System Journal 39(3): 617-632.

Lijding, M., H. Benz, et al. (2006). Smart Signs: Showing the way in Smart Surroundings. TR-CTIT-06-20. Enschede, Centre for Telematics and Information Technology, University of Twente.

Liszka, K., M. Mackin, et al. (2004). "Keeping a Beat on the Hearth." Pervasive Computing 3(4): 42-49.

Long, S., R. Kooper, et al. (1996). Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study. 2nd ACM International Conference on Mobile Computing and Networking (MobiCom'96), New York, USA.

Maes, P. (1994). "Agents that reduce work and information overload." Communications of the ACM 37(7): 30-40.

Margotta, R. (2001). History of Medicine, Chancellor Press.

Marsh, A. (2002). 3G Medicine - The Integration of Technologies. ICCS'02.

Mei, H. and I. Widya (2007). A framework for smart processing of health signals. Freeband AWARENESS D4.25.

Meier, R., A. Harrington, et al. (2006). Spatial Programming Model for Real Global Smart Space Applications. 6th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 06). Bologna, Italy.

Meystre, S. (2005). "The Current State of Telemonitoring: A Comment on the Literature." Telemedicine Journal and e-Health 11(1): 63-69.

Miskelly, F. (2001). "Assitive technology in elderly care." Age and Ageing 30(6): 455-458.

Morla, R. and N. Davies (2004). "Modeling and Simulation of Context-Aware Mobile Systems." IEEE Pervasive computing: 48-56.

Neisse, R., M. Wegdam, et al. (2007). Trust Management Model and Architecture for Context-Aware Service Platforms. The 2nd International Symposium on Information Security (IS'07). Vilamoura, Portugal.

Niskanen, I., J. Kalaoja, et al. (2007). "An Interactive Ontology Visualization Approach for the Networked Home Environment." International Journal of Computer and Information Science and Engineering 1(4).

Oh, H., C. Rizo, et al. (2005). "What is eHealth (3): A Systematic Review of Published Definitions." Journal of Medical Internet Research 7(1).

OMG. (2002). "Corba Component Model version 3.0." from http://www.omg.org/technology/documents/formal/components.htm.

OMG. (2004). "Common Object Request Broker Architecture: Core Specification." http://www.omg.org/docs/formal/04-03-01.pdf, from http://www.omg.org/docs/formal/04-03-01.pdf.

Oscar.org. (2005). "Oscar - An OSGi framework implementation." from http://oscar.objectweb.org/.

OSGi Alliance. (2004). "About the OSGi Service Platform, whitepaper." from http://osgi.org/documents/osgi_technology/osgi-sp-overview.pdf.

OSGi Alliance. (2005). "The OSGi Service Platform - Dynamic services for networked devices." from http://osgi.org.

OSGi Alliance (2005). OSGi Service Platform Core Specification: Release 4.

Osxa. (2006). "Osxa -- the OSGi Framework that boldly goes where others can't." from http://www.osxa.org/wiki.

Oulasvirta, A. (2005). "Grounding the innovation of future technology." Human Technology 1(1): 58-75.

Pascoe, J. (2001). Context-aware software. Canterbury, University of Kent. PhD.

Pattichis, C., E. Kyriacou, et al. (2002). "Wireless Telemedicine Systems: An overview." IEEE Antenna 's and Propagation Magazine 44(2): 143-153.

Pawar, P., A. v. Halteren, et al. (2007). Enabling Context-Aware Computing for the Nomadic Mobile User: A Service Oriented and Quality Driven Approach. IEEE Wireless Communications & Networking Conference (WCNC 2007). Hong Kong.

Perich, F., S. Avancha, et al. (2002). Profile Driven Data Management for Pervasive Environments. 13th International Conference on Database and Expert Systems Applications (DEXA 2002). Aix-en-Provence, France.

Peters, J. and G. Hall (1999). "Assessment of ambulance response performance using a geographic information system." Social Science & Medicine 49(11): 1551-1566.

Philips Medical Systems (2003). TEN-HMS Study Demonstrates Clinical and Financial Efficacy of Home Telemonitoring. whitepaper.

Pressman, R. (2000). Software Engineering: A Practitioner's Approach. Berkshire, McGraw-Hill.

Ramparany, F., R. Poortinga, et al. (2007). An Open Context Management Information Management Infrastructure. Intelligent Environments (IE'07), Ulm, Germany.

Robinson, P., H. Vogt, et al. (2004). Some research challenges in Pervasive Computing. Workshop on Security and Privacy at the Pervasive 2004 Conference. Munchen, Germany.

Robinson, R. and K. Henricksen (2007). XCML: A runtime representation for the Context Modelling Language. Pervasive Computing (PerCom'07). White Plains, USA.

Ross, P. (2004). "Managing Care through the Air." IEEE Spectrum: 26-31.

Roussaki, I., M. Strimpakou, et al. (2006). Privacy-Aware Modelling and Distribution of Context Information in Pervasive Service Provision. IEEE International Conference on Pervasive Services (ICPS 2006). Lyon, France.

Rubin, A. D. and D. E. Geer, Jr. (1998). "Mobile Code Security." IEEE Internet Computing 2(6): 30-34.

Saif, U. and M. Palusak (2003). Service-oriented Network Sockets. International conference on mobile systems, applications and services (MobiSys'03), San Francisco, USA.

Satyanarayanan, M. (1996). Fundamental Challenges in Mobile Computing. Fifteenth ACM Symposium on Principles of Distributed Computing. Philadelphia, USA.

Schilit, B., N. Adams, et al. (1994). Context-Aware Computing Applications. IEEE Workshop on Mobile Computing Systems and Applications. Santa Cruz, CA, USA.

Schilit, B. and N. Theimer (1994). "Disseminating Active Map Information to Mobile Hosts." IEEE Network 8(5): 22-32.

Schilit, N. (1995). A System Architecture for Context-Aware Mobile Computing, Colombia University. PhD.

Schmidt, A., M. Beigl, et al. (1999). "There is more to context than location." Computer Graphics 23(6): 893-901.

Sen, R. and G. Roman (2003). Context-Sensitive Binding, Flexible Programming Using Transparant Context Maintenance. Technical Report WUCSE-2003-72, Technical Report WUCSE-2003-72, Washington University.

Shadbolt, N. (2003). "Ambient Intelligence." IEEE Intelligenct Systems 18(4): 2-3.

Sheikh, K., M. Wegdam, et al. (2007). Middleware Support for Quality of Context in Pervasive Context-Aware Systems. IEEE International Workshop on Middleware Support for Pervasive Computing (PerWare'07), New York, USA.

Shirehjini, A. and F. Klar (2005). 3DSim: Rapid Prototyping Ambient Intelligence. sOc - EUSAI. Grenoble, France.

Sinderen, M. v., A. Halteren, et al. (2006). "Supporting Context-aware Mobile Applications: an Infrastructure Approach." IEEE Communications Magazine 44(9): 96-104.

Sinderen, M. v., M. Verheijen, et al. (2007). Context Modelling and Reasoning in a Context-aware Infrastructure. Freeband Awareness D1.1v2.

Skadron, K., M. Martonosi, et al. (2003). "Challenges in Computer Architecture Evaluation." IEEE Computer(August).

Sowa, J. (2003). Laws, facts, and contexts: Foundations for multimodal reasoning. Dordrecht, Kluwer Academic Publishers.

Standford, V. (2002). "Using Pervasive Computing to Elder Care." IEEE Pervasive computing 1(1).

Sun. (2003). "Java Remote Method Invocation Specification." from http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html.

Sun. (2005). "Java 2 Platform Micro edition (J2ME) datasheet." from http://java.sun.com/j2me/index.jsp.

Sun. (2005). "JAVA 2 Platform, Enterprise Edition (J2EE) specification 5.0." from http://java.sun.com/j2ee/5.0/index.jsp.

Svahn, F. (2003). "Ubiquitous Computing?! Does it apply to my Navigation System?" from http://www.viktoria.se/~fresva/documents/UC_essay.pdf.

Szyperski, C. (1998). Component Software, Addison-Wesley.

Szyperski, C., D. Gruntz, et al. (2002). Component Software: Beyond Object-Oriented Programming, Addison-Wesley.

Tachakra, S., X. Wang, et al. (2003). "Mobile e-Health: the Unwired Evolution of Telemedicine." Telemedicine Journal and e-Health 9(3): 247-257.

Tanriverdi, H. and C. Iacono (1998). Knowledge Barriers to Diffusion of Telemedicine. International Conference on Information Systems. Helsinki, Finland.

The Telemedicine Alliance (2004). Telemedicine 2010: Visions for a Personal Medical Network.

Vissers, C., L. Ferreira Pires, et al. (2002). The Archtitectural Design of Distributed Systems, Lecture notes. Enschede, University of Twente.

Wac, K., T. Broens, et al. (2008). "Survey: Understanding the context-awareness domain", *in preparation*.

Wang, X., T. Gu, et al. (2004). Ontology Based Context Modeling and Reasoning using OWL. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications. Singapore.

Wegdam, M. (2003). Dynamic Reconfiguration and Load Distribution in Component Middleware. Enschede, University of Twente. PhD: 241.

Wegdam, M. (2005). AWARENESS: A project on Context AWARE mobile NEtworks and ServiceS. 14th Mobile & Wireless Communication Summit. Dresden, Germany.

Wegdam, M., T. Broens, et al. (2008). "Lessons learned on context awareness in mobile applications", *in preparation*.

Wegdam, M., J. Brok, et al. (2007). An Architecture for Privacy and Trust in Context AWARENESS Infrastructures. Freeband AWARENESS Dn3.11.

Wegdam, M., M. v. Sinderen, et al. (2008). Overall Awareness Architecture. Freeband Awareness D0.3v4. M. Wegdam. Enschede.

Wegdam, M., J. van Bemmel, et al. (2006). AWARENESS Trust and Privacy Architecture. Freeband AWARENESS Dn3.1. B. Hulsebosch. Enschede.

Wei, Q., K. Farkas, et al. (2003). Context-aware Handover Based on Active Network Technology. Active Networks, IFIP TC6 5th International Workshop (IWAN03), Kyoto, Japan, Springer-Verlag.

Weiser, M. (1991). "The Computer for the Twenty-First Century." Scientific American September: 94-110.

Weiser, M. and J. Brown (1998). The Coming Age of Calm Technology. Beyond Calculations: The Next Fifty Years of Computing. P. Denning, R. Metcalfe and J. Burke, Springer.

Wootton, R. (2005). "TeleMed and eHealth 2004: Citizen-centred care." Journal of Telemedicine and eHealth 11(supplement 1).

Yua, S., Y. Wang, et al. (2002). Development of Situation-Aware Application Software for Ubiquitous Computing Environments. International Software and Applications Conference (COMPSAC'02). Oxford, England.

Zelkowitz, M. and D. Wallace (1997). "Expirimental validation in software engineering." Information and Software Technology(39): 735-743.

Zhang, D., Z. Yu, et al. (2004). Context-Aware Infrastructure for Personalized Healthcare. International Workshop on Personalized Health. Belfast, Northern Ireland.

Samenvatting

De wereld wordt in toenemende mate uitgerust met hoge capaciteit, verbonden, mobiele en ingebedde computer systemen. Context-awareness biedt een attractieve manier om gepersonalizeerde applicaties te realiseren, die beter aansluiten bij de behoeften van de gebruiker in zulke rijke computer omgevingen.

Context-aware applicaties gebruiken de door context bronnen geleverde context informatie om hun gedrag aan te passen aan de zich voordoende situatie. De uitwisseling van context informatie benodigd een associatie tussen een context consumerende applicatie en een geschikte context producerende bron. Wij noemen een dergelijke associatie een 'context binding'.

Het maken van context-aware applicaties is door enkele intrinsieke karakteristieken van context bronnen erg complex. Ten eerste, context bronnen zijn gedistribueerd. Aldus is voor het creëren van een context binding een vorm van discovery en selectie nodig. Ten tweede, context bronnen zijn willekeurig beschikbaar tijdens de levensloop van de applicatie. Dit maakt het onderhouden van een context binding moeilijk. Ten slotte, context bronnen leveren context informatie met fluctuerende kwaliteit. Dit maakt een binding mogelijk ongeschikt voor gebruik door een applicatie. Op dit moment moeten ontwikkelaars aanzienlijke capaciteit steken in het maken van applicatie code, die met deze moeilijkheden om kan gaan, met als resultaat om context bindingen te creëren en te onderhouden.

Dit proefschrift geeft inzichten in de algemene karakteristieken van context-aware applicaties en hun ontwikkelingsproces. Wij stellen een abstractie voor die de Context Binding Transparantie wordt genoemd. Deze transparantie heeft als doel om de complexiteit van het creëren en onderhouden van context bindingen voor de applicatie ontwikkelaars te maskeren. Op deze manier faciliteren we het ontwikkelingsproces van context-aware applicaties. De verantwoordelijkheid voor het creëren en onderhouden van context bindingen wordt ontnomen van de applicatie ontwikkelaar en verschoven naar een context binding infrastructuur. Dit maakt het voor de applicatie ontwikkelaar mogelijk om zich primair te concentreren op de ontwikkeling van de applicatie logica in plaats van op de logica die nodig is om context bindingen te creëren en te onderhouden.

De ontwikkelaar interacteert met de context binding infrastructuur door gebruik te maken van context retrieval en publishing diensten, en een context requirement specificatie taal. Deze taal maakt het mogelijk voor de ontwikkelaar om zijn context eisen op een hoog niveau te specificeren in plaats van in programmeer code. In dit proefschrift stellen we een realisatie van een dergelijke taal voor, genaamd de Context Binding Description Language (CBDL). Deze taal is ontwikkeld om toepasbaar te zijn voor een breed spectrum aan context-aware applicaties.

We stellen ook een realisatie van de context binding infrastructuur voor, genaamd de Context-Aware Component Infrastructure (CACI). Deze infrastructuur realiseert een context binding transparantie en is opgebouwd uit een context binding mechanisme en een context discovery interoperabiliteits mechanisme.

Het context binding mechanisme gebruikt een door de applicatie ontwikkelaar gespecificeerde CBDL document om context bindingen te creëren en te onderhouden, ten behoeve van de context-aware applicatie. Het proces om een context binding te creëren bestaat uit discovery van context bronnen bij beschikbare context discovery mechanismen, selectie van geschikte context bronnen, maken van een binding tussen de applicatie en de geselecteerde context bron, en onderhoud van deze binding. Het onderhouden van een context binding bestaat uit het mogelijk re-binden naar andere geschikte context bronnen in het geval dat de gebonden bron verdwijnt of dat de geleverde kwaliteit van de context informatie fluctueert. Dit proefschrift geeft een voorbeeld van een mogelijk re-binding algoritme.

Het context discovery interoperabiliteits mechanisme maakt het voor een context-aware applicatie mogelijk om transparant gebruik te maken van verschillende context discovery mechanismen die beschikbaar zijn in de applicatie omgeving. Het doel van het interoperabiliteit mechanisme is om de heterogeniteit en de fluctuerende beschikbaarheid van context discovery mechanismen te verbergen voor de context-aware applicatie. Het context discovery interoperabiliteits mechanisme is een ondersteunend mechanisme. Het kan ook onafhankelijk door context-aware applicaties gebruikt worden die niet gebruik maken van een context binding mechanisme.

We hebben een prototype van CACI gecreëerd door gebruik te maken van het OSGi componenten raamwerk. Dit prototype bestaat uit implementaties van het context binding mechanisme en het context discovery interoperabiliteits mechanisme.

De evaluatie van de voorgestelde context binding transparantie en infrastructuur bestaat uit een gebruikers enquête en een vergelijking van de ontwikkelingsinspanning en software kwaliteit van een implementatie van een telemedicine case met en zonder CACI. De resultaten van de enquête gaf een algemene interesse van mogelijke gebruikers weer in de eigenschappen van de context binding infrastructuur. De implementatie van de case gaf een mogelijke verbetering weer van het ontwikkelproces van hogere kwaliteit context-aware applicaties door gebruik te maken van een context binding infrastructuur.

Dit onderzoek wil benadrukken dat de beschikbaarheid van context informatie en de kwaliteit van deze informatie de ontwikkeling van context-aware applicaties zeer beïnvloed. Door een middleware-infrastructuur te gebruiken die de creatie en onderhoud van context bindingen ondersteund kan het makkelijker worden om hogere kwaliteit context-aware applicaties te ontwikkelen.

Publications by the Author

During the development of this thesis, the author has published various parts of this work in the following papers (listed in reverse chronological order):

- Hesselman, C., Benz, H., Pawar, P., Liu, F., Wegdam, M., Wibbels, M., Broens, T., Brok, J., Bridging Context Management Systems for Different Types of Pervasive Computing Environments, International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (MOBILWARE'08), Innsbruck, Austria, February 2008
- Broens, T., Quartel, D., Sinderen, M. van, Capturing Context Requirements, European European Conference on Smart Sensing and Context (EuroSSC'07), LNCS 4793, Kendal, England, 2007
- Broens, T., Poortinga, R., Aarts, J., Interoperating Context Discovery Mechanisms, Architecture, Concepts and Technologies for Service Oriented Computing workshop (ACT4SOFT '07), Barcelona, Spain, 2007
- Broens, T., Quartel, D., Sinderen, M. van, Towards a Context Binding Transparency, EUNICE Open European Summer School 2007 (EUNICE'07), LNCS 4606, Enschede, the Netherlands, 2007
- Broens, T., Huis in't Veld, R., Vollenbroek-Hutten, M., Hermens, H., Halteren, A. van, Nieuwenhuis, B., Determinants for successful telemedicine implementations: a literature study, Journal for Telemedicine and Telecare, 13(6), p303-309, 2007
- Broens, T., Sinderen, M. van, Halteren, A. van, Quartel, D.,Dynamic Context Bindings in Pervasive Middleware, IEEE Middleware for Pervasive Computing Workshop (PerWare'07), White Plains, USA, 2007
- Broens, T., Halteren, A. van, Wac, K., Towards an application framework for context-aware m-health applications, International Journal of Internet Protocol Technology, 2(2), 2007
- Eertink, H., Poortinga, R., Broens, T., Tobies, S., Tokmakoff, A., Halteren, A. van, Sharing Intelligent Services between Homes, European conference on Ambient Intelligence workshop (AMI'07), Darmstadt, 2007

- Jones, V. Mei, H., Broens, T., Widya, I, Peuscher, J., Context Aware Body Area Networks for Telemedicine, Pacific Rim Conference on Multimedia (PCM'07), Hong, Kong, 2007
- Mei, H., Widya, I, Broens, T., van Halteren, A., Shishkov, B., van Sinderen, M., A framework for smart distribution of bio-signal processing units in m-health, International Conference on Software and Data Technology (ICSOFT'07), Spain, 2007
- Olavo Bonino da Silva Santos, L., Ramparany, F., Dockhorn Costa, P., Vink, P., Etter, R., Broens T., A Service Architecture for Context Awareness and Reaction Provisioning, Modeling, Design, and Analysis for Service-oriented Architecture Workshop (mda4soa'06), Chicago, 2007
- Wac, K., Halteren van, A., Broens T., Context-aware QoS provisioning for an M-health service platform, International Journal of Internet Protocol Technology, 2(2), 2007
- Broens, T., Halteren, A. van, Sinderen, M. van, Infrastructural support for dynamic context bindings, European Conference on Smart Sensing and Context 2006 (EuroSSC'06), LNCS 4272, Enschede, the Netherlands, 2006
- Broens, T., Halteren, A. van, SimuContext: Simply Simulating Context, Proceedings of the IEEE International Conference on Autonomous Systems 2006 (ICAS'06), Santa Clara, USA, 2006
- Dockhorn Costa, P., Pires, L., Sinderen, M., Broens, T., Controlling Service in a Mobile Context-Aware Infrastructure, Context-Aware Pro-active Systems workshop 2006 (CAPS'06), Kassel, Germany, 2006
- Etter, R., Dockhorn Costa, P., Broens, T., A Rule-Based Approach Towards Context-Aware User Notification Services, IEEE Conference on Pervasive Services 2006 (ICPS'06), Lyon, France
- Ramparany, F., Euzenat, J., Broens, T., Bottaro, A., Poortinga, R., Context Management and Semantic Modelling for Ambient Intelligence, Future Research Challenges for Software and Services workshop (FRCSS'06), Vienna, Austria, 2006
- Broens, T., Supporting the developers of context-aware mobile telemedicine applications, On The Move conference 2005, LNCS 3762/2005, PhD Symposium, October, Napa, Cyprus, 2005
- Broens, T., Halteren, A. van, Wac, K., Towards an application framework for context-aware m-health applications, EUNICE Open European Summer School 2005 "Networked Applications" (EUNICE'05), Colmenarejo, Spain, 2005
- Wac, K., Halteren van, A., Broens T., Context-aware QoS provisioning for an M-health service platform, EUNICE Open European Summer School 2005 "Networked Applications" (EUNICE'05), Colmenarejo, Spain, 2005

- Kranenburg, H., Salden, A., Broens, T., Koolwaaij, j.,Grounded Contextual Reasoning enabling Innovative Mobile Services, Applications and Services in Wireless Networks workshop 2005 (ASWN'05), Paris, France, 2005
- Pokraev, S., Koolwaaij, J., Setten, M. van, Broens T., Dockhorn Costa, P., Wibbels, M., Ebben, P., Strating, P., Service Platform for Rapid Development and Deployment of Context-Aware, Mobile Applications, IEEE International Conference on Webservices (ICWS'05), Industry track, Orlando, Florida, USA, 2005
- Broens, T., Pokraev, S., Sinderen, M. van, Koolwaaij, J., Dockhorn Costa, P., Context-Aware, ontology based, service discovery, European Symposium on Ambient Intelligence 2004 (EUSAI'04), LNCS 3295, Eindhoven, the Netherlands, 2004
- Wegdam, M., Broens, T., Hulsebosch, B., Hesselman, C., Sinderen, M. van, Tönis, T., Lessons learned on context awareness in mobile applications, *in preparation*
- Wac, K., Broens, T., Pawar, P., Understanding the context-awareness research domain, *in preparation*

Notes