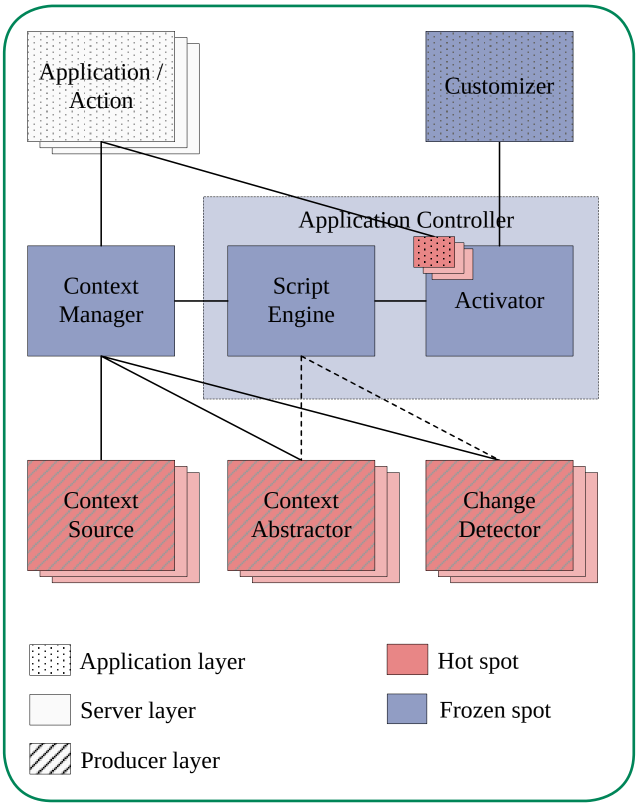
ESPOO 2005 VTT PUBLICATIONS 579



Panu Korpipää

Blackboard-based software framework and tool for mobile device context awareness

VTT PUBLICATIONS 579

Blackboard-based software framework and tool for mobile device context awareness

Panu Korpipää

VTT Electronics

Academic dissertation for the degree of Doctor of Science in Technology, to be presented with the assent of the Faculty of Technology, University of Oulu, for public discussion in Auditorium IT115, Linnanmaa, on November 25th, 2005, at 12 noon.

ISBN 951–38–6669–6 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6670–X (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FI–02044 VTT, Finland
phone internat. +358 20 722 111, fax + 358 20 722 4374

VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde 020 722 111, faksi 020 722 2320

VTT Elektronik, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel 020 722 111, fax 020 722 2320

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FI–90571 OULU, Finland
phone internat. +358 20 722 111, fax +358 20 722 2320

Supervised by
Professor Tapio Seppänen, University of Oulu

Reviewed by
Professor Martti Mäntylä, University of Helsinki
Professor Tommi Mikkonen, University of Tampere

Opponents
Professor Martti Mäntylä, University of Helsinki
Professor Albrecht Schmidt, University of Münich

Technical editing Leena Ukskoski

Otamedia Oy, Espoo 2005

Korpipää, Panu. Blackboard-based software framework and tool for mobile device context awareness. Espoo 2005. VTT Publications 579. 225 p.

Keywords mobile computing, context-aware computing, mobile interaction, mobile context awareness, application control, blackboard-based architecture, software framework, context management, information model, application programming interface, sensor-based interaction, customization, end-user development, personalization

Abstract

The usage needs of a mobile device vary according to context. Mobile context awareness research aims at providing the device user with a way of usage that suits the situation. Interaction based on context requires acquiring, abstracting and delivering information from multiple sources, such as sensors, to the application or application control. A generic software framework and tool for facilitating the rapid development of mobile device context-aware applications were developed in this work. The blackboard-based framework supports all tasks that are required for context-based application control, where contexts can be any events that are relevant to user interaction with the application, including explicit inputs. The core component of the framework, Context Manager, provides a publish and subscribe mechanism and a database for the applications and application control. The framework provides an application programming interface (API) for developers. As a higher abstraction-level programming interface, a customization tool enables easy end-user development of context-aware features into existing applications without changing them.

An extensible ontology is used as a uniform context representation within the framework. The purpose of the ontology, together with the API, is to enable easy access, use and reuse of human-understandable context information. Context information sources, such as sensors, often produce a continuous stream of low abstraction-level data. The framework supports the transformation of a continuous data stream into abstracted context events, described in the ontology. Context information is delivered to applications or application control as abstracted events. The main result of the dissertation is a software framework, ontology and tool, which facilitate the customization of sensor-based human-computer interaction in mobile devices. The practical applicability, scope, and computational efficiency of the implemented framework and customization tool are evaluated with performance measurements and multiple applications implemented in a mobile phone with real sensor sources.

Preface

This is an academic dissertation for the degree Doctor of Science in Technology from the University of Oulu, Faculty of Technology. The dissertation supervisor was Professor Tapio Seppänen from the University of Oulu. The appointed reviewers of the dissertation were Professor Tommi Mikkonen from the Technical University of Tampere and Martti Mäntylä from the University of Helsinki, Helsinki Institute of Information Technology. The received professional scientific feedback is gratefully acknowledged.

The work for the dissertation was carried out at VTT Electronics, Oulu, Finland during the years 2000–2005. The work was funded for the most part by Nokia through VTT project funding.

Science is teamwork. In the team of contributors for this dissertation, the first man to acknowledge, especially regarding scientific visibility, is Dr. Jani Mäntyjärvi. He frequently suggested publishing and forums for the scientific contributions, and co-authored multiple articles related to this dissertation. I wish to thank Juha Kela for significant participation in the architecture design and development process and for co-authoring articles; co-authors Heikki Keränen, Esko-Juhani Malm, Tapani Rantakokko and Vesa Kyllönen all had a professional iron grip on implementing and developing the designs; Professor Heikki Ailisto for commenting on the first version of the dissertation; and other colleagues at VTT and University of Oulu for their support. Thanks to the supervisor of the dissertation work, Professor Tapio Seppänen, are in order for insightful guidance and constructive scientific criticism in formulating the dissertation, and for co-authoring an article.

I am much obliged to Dr. Pertti Huuskonen, Urpo Tuomela, Ilkka Känsälä and Jonna Häkkilä for advocating the Nokia funding for the work over many years, outlining the leading directions and co-authoring articles – and many other colleagues from Nokia for their support. Thanks to Heikki Huomo from Nokia NVO for long-term backing of context-awareness research and valuable co-operation in the process of deploying the results into the industry. Furthermore, I would like to acknowledge the contributions of Johannes Peltola, Miika Koskinen, Satu-Marja Mäkelä, Sami Ronkainen, Johan Himberg, Heikki

Mannila, and Ilkka Salminen as co-authors of the articles related to this dissertation.

The scholarships granted by the HPY foundation and the Seppo Säynäjäkangas science foundation are gratefully acknowledged.

To end the preface, I would like to quote a thought from a great scientist:

"If we knew what it was we were doing, it would not be called research, would it?"

– Albert Einstein, 1879–1955

Oulu, October 2005 Panu Korpipää

Contents

List of symbols

2D	2-Dimensional, an entity having two dimensions
3D	3-Dimensional, an entity having three dimensions
AI	Artificial Intelligence, a research direction aiming at human-like intelligence in machines
API	Application Programming Interface, a set of functions provided by a software component
CBR	Case-Based Reasoning, a classification method
CC/PP	Composite Capabilities/Preference Profiles, an RDF-based data model for static device properties
CD	Compact Disc, a data storage
CEP	Context Exchange Protocol, a specification for exchanging context information
CORBA	Common Object Request Broker Architecture, a model for distributed communication of data
CYC	Common sense knowledge model, a model for representing common sense knowledge
CycL	CYC Language, a formal language for describing common sense
DAML+OIL	DARPA Agent Markup Language + Ontology Inference Layer, a Semantic Web-based information representation language
ER	Entity-Relationship, a model for representing entities and their relationships

EUD	End user development, software development performed by the end user
GHz	Gigaherz, a measure of frequency
GUI	Graphical User Interface, graphical interface for human-computer interaction
HCI	Human-Computer Interaction, a process of interaction between a human and a computer
HMM	Hidden Markov Model, a probabilistic model for representing a sequence of events
HTML	HyperText Markup Language, a language for representing hypertext
HTTP	Hypertext Transfer Protocol, a data transfer protocol
Hz	Herz, a measure of frequency
ICA	Independent Component Analysis, a method for finding independent patterns in data
IDL	Interface Description Language, a language for describing data structures
IP	Internet Protocol, a data transfer protocol
KB	Knowledge Base, a collection of logic sentences
KNN	K Nearest Neighbours, a classification method
LDA	Linear Discriminant Analysis, a classification method
MPEG-7	Moving Picture Expert Group standard 7, a standard for describing audio and media content

ORB	Object Request Broker, a software component in CORBA
OWL	Web Ontology Language, a Semantic Web-based information representation language
OWL DL	Web Ontology Language Description Logics version, a Semantic Web-based information representation language, a version of OWL
PC	Personal Computer, a universal machine for performing operations with any series of bits
PCA	Principal Component Analysis, a method for representing the information content of multidimensional data with lower dimensional projections
PDA	Personal Digital Assistant, a small portable personal computer
PSM	Problem Solving Method, a method that performs inference based on data described with ontologies
RDF	Resource Description Framework, a Semantic Web-based information representation framework
RDF-S	Resource Description Framework Schema, a Semantic Web-based information representation framework for RDF meta-data
RFID	Radio Frequency Identification, an electronic circuit for storing information that can be read from close range
SMS	Short Message Service, a protocol for communicating short text messages with mobile phones
SNR	Signal-to-Noise Ratio, the ratio between signal and noise
SQL	Structured Query Language, a standard language for querying a relational database

UAProf	User Agent Profile, a model for defining static device properties
UDDI	Universal Description, Discovery and Integration, a model for Web service discovery
UI	User Interface, an interface for human-computer interaction
UML	Unified Modeling Language, an abstract visual language for modelling entities, relations and processes
UPnP	Universal Plug and Play, a model for exchanging data between devices
URI	Universal Resource Identifier, a Semantic Web-based identifier for any web resource
URL	Universal Resource Locator, identifier of an entity in WWW
W3C	World Wide Web Consortium, a consortium developing the Semantic Web
WLAN	Wireless Local Area Network, a wireless short-range IP-based network
WWW	World Wide Web, a network of computers that communicate based on IP and HTTP
XML	Extensible Markup Language, a language for information representation

1. Introduction

1.1 Background and motivation

The common factor in mobile, nomadic, pervasive, ubiquitous and wearable computing is that the user is mobile. The usage needs of a mobile device vary in different places and situations. Mobile context awareness research aims at providing the device user with a way of usage that suits the situation, to increase the usability of the device.

The concept of context itself is older than mobile computing. This is reflected, perhaps unintentionally, by the fact that the popular definition of context and context awareness (Dey & Abowd 2000) actually implies that computer applications have always been context aware. However, mobility brings a new dimension to context. The information about the mobile device and user activity, environment, other devices, location, and time can be utilised in different situations to enhance the interaction between the user and the device. This is the base assumption of context awareness research.

New input sources, such as embedded sensors producing interaction-related information, are becoming available for mobile devices. These sources enable novel ways of interacting with the device, and even open possibilities to create entirely new types of mobile applications. To facilitate the full potential of utilising such new input sources, a software framework is required with a uniform means of acquiring and processing useful context-related information, and providing it for mobile device applications. A crucial task is the ability to produce reliable information in the presence of uncertain and rapidly changing data from multiple sources. The capability of systematically managing a wide variety of interaction inputs – i.e., contexts – is needed to facilitate quick development of context-aware features in mobile devices.

The users decide the usefulness of the context-aware features. Therefore, the development of context-aware applications needs to be tightly connected to the end-user demands. Traditionally, context-aware features have been hard-coded into applications, which makes development slow and inflexible to varying user requirements. Adding context-aware features into existing applications has required changing the existing application code. The preferences of how a mobile device is used for interacting with its applications vary among users, and the preferences of one user may change over time. At the design time it is thus difficult to define the behaviour of the device so that it meets the varying user demands in varying situations. The end-user should have the possibility of customizing the way of interacting with mobile device applications.

1.2 Research problems and hypothesis

There are many research problems still to solve in mobile context awareness. The range of context types applied in mobile computing is limited. Dey (2000) suggests that the main reason why applications have not covered more context types and context-aware features is because context is difficult to use. The author proceeds to state that the reasons why context is difficult to use include that context must be abstracted to make sense to the application, context may be acquired from multiple distributed and heterogeneous sources, context is dynamic; and that context is acquired from non-traditional devices, with which there is limited experience. Furthermore, Schmidt (2002) identifies a number of challenges in context-aware computing, of which the following issues are particularly relevant to this dissertation:

- It is still unclear how context relates to real world situations, how it can be represented in a universal way, and how it can be used to enhance applications.

- What is context useful for, and what kind of applications it can be used to enhance? The relationship between context and other inputs into the system has to be addressed.

- How to acquire context is still a central question in context-aware systems.

- Connecting context acquisition to context use is essential, and, for the utilisation of context by various components, agreement must be found on a representation of context useful to a multitude of components.

- Support is needed for building context-aware applications. Providing support for context acquisition, context provision and context use is necessary to make the development of context-aware applications simpler.

These questions are still relevant regarding mobile context awareness. Furthermore, Schmidt (2002) hypothesises that: “For all situations that belong to the same context, the sensory input of the characterising features is similar.” The hypothesis is in the source of a fundamental problem, as addressed by the author. In the real world it is very likely that different contexts produce similar characterising features, since all the aspects of the real world context cannot be sensed. For evaluating the true detectability of the real world context, extensive tests with a lot of repetitions from different situations are required, and even then the results may not be completely reliable. This is one of the reasons for the fact that only relatively straightforward and unambiguous context inputs have been used for context-aware applications to date. Moreover, it brings forward the necessity of user participation in defining the context-based features that are relevant for them.

The questions asked by the authors are still relevant in part: context is difficult to apply, and the kind of applications it is useful for is not clear. This is especially true with mobile phones, where applied context awareness has still largely remained as a future promise. Where mobile phones are concerned, a key issue is that context awareness should be constantly available to the mobile applications for enhancing user interaction, independent of external infrastructure and different networks. To have any chance of user acceptance, the mobile phone context-based user interaction must never be interrupted by a lack of infrastructure or possible delays in network communication. The aim is to enhance the interaction, not impair it. Therefore, a fundamental requirement in sensor-based context awareness for mobile phones is that context must be managed by the terminal itself. From a mobile terminal-centric viewpoint a relevant question to ask is:

How far can context sensitivity be pushed by focusing on technologies that can work on the mobile device itself, largely without the help of external infrastructure?

This setup immediately leads to finding an answer to:

What is needed to properly facilitate and support standalone infrastructure-independent context sensitivity in mobile phones?

This is a central question to this dissertation. To approach this question from all relevant aspects with detail sufficient to proceed, the research area is probed by posing additional and more detailed questions and conducting a preliminary analysis. The specificity of the following questions reflects the starting point to the research in this dissertation with respect to the related work, which is to be reviewed in section 2.

- How to represent context using a common structure, which is the same for all context-utilising applications, instead of laboriously defining a new representation for every new application? The content of context information varies according to the application domain, but the representation structure should remain unchanged. What kind of representation structure is needed when it should be simple enough to enable easy application development but expressive enough to be suitable for utilisation by as many types of applications as possible?

- What kind of application programming interface should be provided when, for simplicity and configurability, context needs to be used through the same functions for all context-utilising applications, independent of what contexts are involved?

- What kind of context representation is suitable when contexts must be flexibly available for the applications as data objects, instead of having to make new application code that connects to a new context source for a new context?

- What kind of framework structure and process is required when context should be received by the applications as events that occur when relevant changes in the situation of the user occur? A continuous, low abstraction-level data stream up to the application is not efficient and lacks interpretation.

- What kind of representation and application programming interface are needed when application developers should be provided with an application programming interface and ontology that lets them use abstracted context data elements defined and provided by other developers?

- How to detect context reliably in real world situations? More experiments are clearly needed, but is even that enough? Extensive data sets and quantitative measures are required for the evaluation. To what extent is applying multi-sensor context recognition in mobile devices feasible, or is it feasible at all?

- How to design context-aware applications that meet the needs of users when the preferences of application users are personal, and those preferences may change over time? End-user support for customizing context-aware features is needed, but how to practically facilitate a wide involvement of end users in defining context-aware functionality?

- How to connect the right contexts to the right actions, and what kind of tools are needed for this? Mapping of contexts to their usage is difficult, and the usefulness of the new features is unclear at the design time. Only the end-users can decide which features are useful.

- How can it be verified that context framework, representation, abstracting and recognition, customization, and applications work properly in a mobile device? For gaining real world usage experience, context-aware applications should be developed for those target devices that are truly mobile; carried with, and used by the user in changing situations, such as mobile phones, instead of making experiments with networked PCs.

The research problems for this work are integrated from the above-mentioned issues. Hence, to be able to gain an insight into the utility and implications of mobile terminal-centric context awareness, the following specific research problems need to be addressed first in order to find out how to best enable standalone context sensitivity in mobile terminals.

1. What is required to flexibly and efficiently handle all relevant aspects of sensor-based mobile terminal-centric management of context-related information?

2. How to represent context information so that it can be systematically processed, stored and used by the applications, and understood by the application developers, while maintaining representation extensibility?

3. How can context be recognised and abstracted online into a common representation from many different sources, especially device sensors, producing possibly incomplete and imprecise information?

4. What kind of application programming interface is required for the simplified development of context-aware applications, and further, what kind of tool is required for end-user development in mobile handheld devices?

Context-aware computing is a multidisciplinary field of research. The progress requires broad-viewed development of multiple topics. The summarised research hypothesis is the following.

By solving the research problems 1–4, it will be possible to create a functional software framework and tool that will enable end-users to quickly customize versatile context-aware applications in a mobile device.

1.3 Scope of the research

The research problems for this dissertation have been specified. This section further focuses the research area by restricting the scope.

This dissertation contributes solutions for advancing the development and application of context awareness in mobile devices, and, especially, handheld mobile devices. Handheld mobile device, such as a mobile phone, refers to a small lightweight multifunction device that is often carried with the user and contains at least one or more processors, operating system, several applications, a number of input devices, and a number of output devices including a display. In this dissertation, laptop PCs are not considered mobile devices, they are portable devices.

The primary sources of context are embedded in the device. Environment infrastructure and distributed computing-related issues are only discussed as extensions. The chosen device-centric approach also refers to performing most of the processing in the device, as opposed to performing the processing in the environment infrastructure. Hence the pervasive computing branch called “smart spaces” is not within the focus of this dissertation, although it is discussed in the literature review. The principal difference in the approach of this dissertation is that the mobile device is required to sense the environment and react accordingly, instead of having the environment detect the situation and react to the device. Moreover, the smart space approach requires the environment to contain a heavy computational infrastructure. In the device-centric approach computation is performed in the mobile device, and no external infrastructure is necessary for the complete operation of a context-aware application.

The primary source of context information is sensors. The framework and the context representation are designed to utilise other sources as well, but the focus is on sensors attached to the device, other device internal sources, or local wireless sensors. Concerning representation, the structure for representing context information should be common across domains, and the domain dictates the vocabulary of the context types. The emphasis in this dissertation is on the domain of sensor-based contexts. The framework and representation support managing location information, but otherwise location context is not discussed in detail. The focus is on dynamic context types which may have very rapidly changing values, and the applications may have tight response requirements. Static contexts, such as device properties, are supported but not discussed in detail.

The context-aware computing view is adopted for context definition. Context, as defined by the linguistic or common sense reasoning communities, is beyond the scope of this research. Concerning context representation syntax, the dissertation will review several alternatives of Semantic Web-related markup languages, but a detailed comparison is beyond the scope of this research. The dissertation does not focus on the markup languages.

Sharing of context information through the network is beyond the scope of this research. A couple of networked PC-based context frameworks will be reviewed, but the viewpoint is mobile device-centric.The question of evaluating the usefulness and usability of the context-aware features that are created in the applications by utilising the framework is beyond the scope of this research and requires further work. However, the early literature, e.g. Pascoe et al. (1999), already states that context-aware features have been experienced as useful.

The security issues regarding context-aware computing are beyond the scope of this research and require further work.

1.4 Research methods

The overall research strategy is the following. Identify what is required for answering the research problems and fulfilling the hypothesis, based on the literature review and use cases from the application viewpoint. Based on the identified requirements, develop the context framework. Finally, evaluate the framework by analysing the realisation of the requirements, which answer the research problems, in the implemented framework and in the applications that apply the framework.

Based on the research problems, the discussion is divided into three main sub-topics. Each sub-topic is studied starting from a literature review, identifying the requirements, and proceeding towards design, implementation and evaluation. The fourth sub-topic, which partly combines the other topics, according to the fourth research problem, is discussed together with the first sub-topic in the review and design part of the dissertation, and separately in the application and evaluation part.

Blackboard-based context framework and API

Literature review: Compare different software framework models, and form a basis for specifying the requirements for context framework.

Development: Analyse and specify the framework requirements based on use cases, and design and implement the framework according to the requirements.

Evaluation: Evaluate the framework and selected applications that apply the framework against the requirements.

Context representation and ontology

Literature review: Compare information representation methods in the literature and analyse their suitability for context information representation, form a basis for specifying requirements for the representation.

Development: Specify the requirements for context ontology, and design and implement the ontology for mobile device context awareness.

Evaluation: Evaluate the ontology and selected applications that apply the ontology against the requirements.

Context abstracting and recognition

Literature review: Compare the widely applied machine learning and inference methods, and analyse their suitability for context recognition in a mobile device.

Development: Choose suitable method(s) for context recognition and evaluate them with a case study. Implement context abstractors and recognisers in the context framework.

Evaluation: Present quantitative measures for context recognition accuracy in the case study. Evaluate the implemented framework elements and applications that operate based on the elements against the requirements.

1.5 Author's involvement and contribution to the results

This dissertation binds together the results of work in multiple projects during the years 2000–2005. The contribution of the dissertation is the result of teamwork. The contributing projects were the following: Episode3, Episode4, Proteus, Narsil, Anduril, Glamdring, Ambience, Nomadic Media, and Silmaril. Ambience and Nomadic Media were ITEA projects, and others were contract research funded by Nokia. The contribution of each project to this dissertation is briefly summarised.

Before the start of the mentioned project continuum, the author participated in a project in which a fault diagnosis system was created for a hot strip mill of a steel plant during the years 1996–1998. The author was responsible for designing and implementing an architecture and methods for recognising abnormal situations in the process, where measurements were acquired from numerous very different sources. Two publications resulted from that work (Kurki et al. 1998, Korpipää 2001). The approved architectural practices applied to the hot strip mill diagnosis system were later utilised in the design of the context architecture. Initiated by Dr. Pertti Huuskonen, Urpo Tuomela and Dr. Esa Tuulari, the first task of the author in the Episode3 project was to design and implement a context recognition architecture, which was ready at the end of January 2000. The design had the principal concept of a central context information server, which received data from heterogeneous sources but in a uniform representation. The author inherited the concept from the hot strip mill diagnosis system architecture.

Following the architecture design, the author implemented the first context recognition system for PC environment. The system, which used multiple sensor sources embedded in a sensor box (Tuulari 2000) that could be attached to a mobile phone, processed measurements with multiple concurrent abstractors – many of them initially designed by Dr. Jani Mäntyjärvi – and was ready and functional in the first quarter of the year 2000. The context recognition architecture already utilised a uniform structure for representing the abstracted data from multiple heterogeneous sources. This work, although not published at the time, was the basis for the development of the context framework in this dissertation, and the basis for producing abstracted data for multiple studies of explorative data analysis, e.g. (Mäntyjärvi et al. 2001, Himberg et al. 2001). During the following project, Episode4, the author further studied the uniform representation for context information. The representation was later further developed and first published by Korpipää and Mäntyjärvi (2003) in June. The following project, Proteus, executed in the year 2001, focused on context recognition, and the results were published by Korpipää et al. (2003a). In Narsil, during the first quarter of the year 2002, the author designed the context architecture for mobile handheld devices together with Juha Kela. The concept of a central context information server from the earlier design was utilised and further developed into the blackboard-based Context Manager. The Anduril project, later in 2002, continued the design and implementation of the framework, and the results were published by Korpipää et al. (2003b). The Glamdring project, during the latter half of 2003 and the beginning of 2004, concentrated on developing gesture recognition for mobile devices. The results were published by Mäntyjärvi et al. (2004) and Kela et al. (2005). In Nomadic Media, the context vocabulary model was applied to multiple domains. Finally, the Silmaril project in the year 2004 concentrated on developing and applying an end-user tool for the development of context-aware applications. The results were published by Korpipää et al. (2004a, 2005a, 2005b).

As mentioned, parts of this dissertation have been published in international scientific conferences and journals. Each publication is referenced in the corresponding chapter in which the issue is discussed. Some publications are referenced in more than one chapter. Hence the author's involvement in the most relevant published results is summarised here, in the order of appearance of the publications.

In Mäntyjärvi et al. (2001) and Himberg et al. (2001) the author was responsible for defining the representation of context features (context atoms) and for producing the features for the experiments, together with co-author Dr. Jani Mäntyjärvi. Other aspects of these two articles are not discussed in this dissertation.

In Korpipää and Mäntyjärvi (2003) the author defined the context ontology structure, i.e., the common properties of context information, context object. The author designed a sensor-based context ontology vocabulary together with co-author Dr. Jani Mäntyjärvi.

In Korpipää et al. (2003a) the author had the main responsibility in designing the recognition experiments and the representation and visualization of the features and classification results. The author had the main responsibility in analysing the results. Co-author Miika Koskinen implemented the Bayesian networks, the visualization, and executed the classification, and classification accuracy calculations. Co-authors Johannes Peltola and Satu-Marja Mäkelä produced the audio-related features. Co-author Professor Tapio Seppänen was responsible for selecting the Bayesian classifier framework and participated in designing the experiments.

In Korpipää et al. (2003b) the author designed the blackboard-based context framework and API together with co-author Juha Kela. The author designed how context ontology, and different levels of abstraction, are utilised within the framework. Co-author Dr. Jani Mäntyjärvi was responsible for the context-based fuzzy application control experiment. Co-authors Heikki Keränen and Esko-Juhani Malm participated in the development process and implemented the context framework and API.

In Korpipää et al. (2004a), the author invented how context ontology is utilised for automatically generating the user interface views in the customization tool. The author designed the model for specifying new context vocabularies. The author participated in the design of the user interface of the customization tool for small-screen mobile devices together with the co-authors Jonna Häkkilä, Juha Kela, Sami Ronkainen and Ilkka Känsälä. Co-author Jonna Häkkilä had the main responsibility in user interaction design and usability testing. Ilkka Salminen and Harri Lakkala contributed with the idea of describing context-action rules formally by using CEP scripts (Lakkala 2003a). Harri Lakkala and Ilkka Salminen provided the CEP syntax, designed as compatible with the context framework and the context ontology structure published by Korpipää and Mäntyjärvi (2003) and Korpipää et al. (2003b).

In Korpipää et al. (2005a) the author designed how context ontology and context framework, together with the customization tool, can be used for end-user development of context-aware applications, and designed the framework extension for application control. Co-authors Esko-Juhani Malm, Tapani Rantakokko and Vesa Kyllönen participated in the design process and implemented the system. Co-author Ilkka Salminen and Harri Lakkala provided the Rule Script Engine, which is used as a rule-based inference engine in application control, and the CEP format for describing rules. Co-author Ilkka Känsälä participated in the development and innovation process.

In Korpipää et al. (2005b) the author designed how the blackboard-based framework is extended to enable human-computer interaction customization, with enhancements for facilitating explicit novel input modalities such as gestures and physical selection. The author had the main responsibility for designing the integration of the framework elements, including various context sources, such as an HMM-based gesture recogniser. Co-author Jonna Häkkilä had the main responsibility for usability evaluation with the implemented system and an important role in the user interface design process. Co-authors Esko-Juhani Malm, Tapani Rantakokko and Vesa Kyllönen were responsible for the implementation. Co-authors Juha Kela, Ilkka Känsälä, and Dr. Jani Mäntyjärvi participated in the development and innovation process. Ilkka Salminen and Harri Lakkala provided the Rule Script Engine, the CEP format, and part of the context sources.

1.6 Outline of the dissertation

This dissertation has been written as a monograph. Even though there are multiple closely related scientific publications that contain most of the contributions in the dissertation and part of the text, the dissertation adds extensive requirements analyses and evaluation, updates the results, and binds the material into a consequential ensemble. As such, it is much clearer to understand than a bundle of articles would be.

The dissertation consists of studying three main entities – context framework, representation and ontology, and abstracting and recognition – according to the three first research problems. The fourth topic discusses the application programming interface and customization tool, which combine and utilise the results of the other sub-topics.

Correspondingly, the literature review is divided into three parts, after a short general introduction into the concept of context awareness. After the review, each sub-topic is studied separately by first deriving the requirements, which answer the research problems. Requirement analyses are followed by the designs, and, in the case of context recognition, an experiment. The application programming interface forms a separate chapter since it contains elements from each of the previous designs. Similarly, the customization tool, which binds together all the sub-topics and answers the fourth problem, is discussed in a separate chapter after the API.

The three sub-topics are evaluated separately. The implementation of each sub-topic is evaluated against the requirements. Moreover, a set of example applications is presented for each sub-topic. Since the applications use the features defined in the requirements, which have been set to answer the research problems, the results will be validated. The evaluation is followed by discussion, where the research problems and hypothesis are answered directly, the contributions and comparison with the related work are summarised, and the significance of the results is discussed. Finally, pointers for future work are given.

2. Review of technologies for mobile context awareness

The literature review gives an overview of context and context awareness-related work and examines the three related main sub-topics in more detail. Since each of the sub-topics has extensive background literature as a separate research direction, it is necessary to briefly introduce them and ground the terminology with definitions. Moreover, each sub-topic has a set of relevant enabling technologies to introduce and review.

2.1 Context and context awareness

2.1.1 Definitions

What are context and context awareness? The literature gives a multitude of answers to the question. Common definitions of context are close to synonyms, such as situation, state, setting, surroundings, etc., concerning user, application or environment (Hull et al. 1997; Pascoe 1998; Rodden et al. 1998). Context awareness is usually defined as the ability of an application to dynamically change or adapt its behavior according to the context (Brown et al. 1997; Schilit et al. 1994; Ward et al. 1997). The most general definitions by Dey and Abowd (2000) are the most widely adopted, perhaps because a general definition covers more research of the multidisciplinary science. The definition of context by Dey and Abowd (2000) is the following.

Context is any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between user and an application, including the user and the application themselves.

Furthermore, the authors suggest that context typically belongs to four categories: location, time, activity and identity. Every event has a place and time, which are fairly straightforward to sense and describe, and thus they are the most commonly used in context-aware applications to date. Representing and managing identity information is similarly straightforward and has been used to some extent in applications. However, when identity information is shared, privacy issues are of concern. Activity is a far more complex context category. It can be divided into the user, the device, and the environment activity. Sensing and recognising all but the simplest user activities requires a multitude of sensors, sophisticated recognition methods, background knowledge modelling, etc.

Context awareness is defined by Dey and Abowd (2000) as follows:

A system is context aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

Context can be used in applications in many ways. Schilit et al. (1994) propose the following categories for exploiting context: proximate selection, automatic contextual reconfiguration, contextual information and commands, and context-triggered actions. Dey et al. (2001) generalise the categories into the following three uses of context:

1. presentation of information and services to a user
2. automatic execution of a service
3. tagging of context to information for later use.

Context awareness research has inherited grandiloquent ambitions similar to artificial intelligence research (Russel & Norvig 1995). Schmidt (2000) has the following vision of future context-aware devices:

We will be able to create (mobile) devices that can see, hear and feel. Based on their perception, these devices will be able to act and react according to the situational context in which they are used.

Furthermore, Schmidt (2000) introduces the notion of implicit interaction. The availability of sensing technology is seen as a factor enabling a shift in HCI from explicit interaction, such as direct control by the user, to a more implicit interaction based on situational context. Hence context-aware computing is often understood as the use of implicit information for performing actions. However, according to the definition of context by Dey and Abowd (2000), explicit interaction events can also be regarded as context information.

Analysis of context information reveals common characteristics. Context information can be divided into two categories based on temporal characteristics. Static context information does not change over time, and includes settings such as the user device properties. Dynamic context refers to the information, which does change over time, with varying frequencies, depending on the information source.

Context information is often imperfect. Context information may be incorrect if it fails to reflect the true state of the world, inconsistent if it contains contradictory information, or incomplete if some aspects of the context are not known (Henricksen et al. 2002). Context information can be partially true – i.e., fuzzy (Zadeh 1965, 1996). It can also be true with a certain probability, based on earlier evidence. These characteristics reflect the uncertainty of context information, which must be considered when choosing the methods for representing and processing context information.

Even though many definitions have evolved, there still seems to be no consensus on what context should include. If the most generic definition is used (Dey and Abowd 2000), the concept of context becomes very general, and includes, among many other things, the explicit input given by the user to control an application, which is very relevant to, and part of, the situation of the user. The concept of context information as implicit information about the usage situation is more specific, but excludes important aspects. Should it then be concluded that all computer applications that are used by a human are context aware? Winograd (2001) points out that context-aware computing might be better described as the design of computing mechanisms that can use characterisations of some specified aspects of the user's setting as a context for interaction. In the case of mobile computing, this setting can change rapidly.

2.1.2 Critique

Context awareness research has received constructive critique. Greenberg (2001) emphasises the difficulties originating from the dynamic nature of context. The author questions the feasibility of context-aware computing in the following three major problem areas.

1. *Determining an appropriate set of canonical contextual states may be difficult or impossible.* It is not always possible to enumerate *a priori* a limited set of contexts that match the real world context. Moreover, if such a set is found, and is valid today, it may be inappropriate at any other time because of "internal and external changes in the social and physical circumstances".

2. *Determining what information is necessary to infer a contextual state may be difficult.* Many things contribute to context, and the relevance of these parts of context depends on the situation (context). User internal context information, such as interests, history, mood, objectives, is very difficult to capture. The system can only provide an approximation of the real context.

3. *Determining an appropriate action from a given context may be difficult.* Even if two contextual states appear to be same, one desired action may differ from the other. This can be due to the different history of events leading to the current, or the undetectable, internal states.

The claims of Greenberg (2001) are strengthened by Bellotti and Edwards (2001), who state that: "There are human aspects of context that cannot be sensed or even inferred, so context-aware systems cannot be designed simply to act on our behalf." As ways of avoiding the inappropriate design of context-aware applications, Greenberg (2001) discusses three ideas. Context-of-use should be studied carefully, and risky automatic actions should only be taken when there is strong evidence of correctness. The systems should be flexible; e.g., the user should be able to adjust the collected information as well as the inferred actions. Feedback from the inferred contexts is considered important, so that the users can view contexts and system behaviour, and make adjustments when necessary.

Erickson (2002) notes that the goal of context-aware computing is desirable: developing devices that are able to sense the situation and adapt their actions appropriately. However, the author points out a foundational problem: the context awareness exhibited by people is radically different from that of computational systems. People notice and understand a vast number of different kinds of cues, and interpret them according to their experience, while devices are only able to measure and recognise a very small set of simple cues, and act according to the predefined rules. The author proceeds that the primary motive of context awareness is to allow the systems to take action autonomously, leaving people out of the control loop, which requires considerable intelligence and common sense. Common sense is difficult to implement. Erickson (2002) suggests that humans should be kept in the control loop by, e.g., presenting them with measured context cues, letting them recognise context instead of devices, and letting them make decisions about actions. In fact, Schmidt et al. (2000) give an example of such a system, where the caller is informed about the situation of the person he is calling, and further actions can be made by the caller based on that information. Erickson (2002) concludes that the concepts of context and awareness are too powerful notions for describing such systems, and that "context-aware computing would do better to emulate the approach taken in scientific visualization, rather than in trying to re-enact AI's attempts at natural language understanding and problem solving."

2.1.3 Discussion on critique and definitions

Despite the critique, no new definitions for context and context awareness are given in this dissertation. The definitions of Dey and Abowd (2000) are adopted here, but with two adjustments. First, context should be human-understandable for easy use. For example, a plain voltage measured from a temperature sensor is not context in this dissertation, it becomes such when it has been given an abstraction, which is understandable to a human, such as temperature in Celsius degrees. Second, context should primarily relate to the mobility, and hence describe dynamic situations of the user and the device. Mobility is the main characteristic in the context awareness research.

Context is not regarded in this dissertation strictly as implicit information for application control. Explicit control information produced by the user is also context information, but in general it should only be considered a small part of the overall context. The goal should be to interpret context abstractions that more accurately reflect the situation of the user. A wide scope in context-aware application development requires acquiring, recognising and representing as many potentially useful constituents of the context as possible.

Correspondingly, a software framework is needed that offers the possibility to acquire, manage and deliver as events to applications *any* information derived from sensors attached to the device, including sensor input from, e.g., hand movements, used for a direct device control by the user. Hence a context framework should be viewed as a platform for managing and abstracting any sensor-based or other interaction-related information in order to enable event-based actions and efficient application control.

As was noted in the critique, it is evident that context-aware computing involves certain goals that are not feasible. For instance, fully automatic actions based on context, implemented as non-customizable at design time, are rarely useful, and wrong automatic actions can be very frustrating, as was pointed out by Erickson (2002). It is also clear that all aspects of context cannot be sensed, but that does not exclude the possibility of being able to sense some useful aspects of context. Furthermore, it is not necessary to aim at fully automated actions as the only goal of context awareness. Customization partially overcomes the third problem stated by Greenberg (2001): the problem of determining an appropriate action based on context. If the event-action behaviour is defined by the end-user instead of the application developer, a greater degree of personalisation and flexibility can be achieved. Moreover, the first problem by Greenberg (2001) is partly solved by letting the user change the event-action configurations if it is required when the social and physical circumstances change over time.

2.1.4 Related dissertations

In previous years context awareness research has been quite intensive. Five earlier Ph.D dissertations are considered related to this dissertation. Before a detailed review, to give a quick overview summary in advance, the earlier dissertations are listed here in order of appearance:

1. ‘A System Architecture for Context-Aware Mobile Computing’ (Schilit 1995)
2. ‘Providing Architectural Support for Building Context-Aware Applications‘ (Dey 2000)
3. ‘Supporting The Development of Mobile Context-Aware Systems’ (Mitchell 2002)
4. ‘Ubiquitous Computing – Computing in Context’ (Schmidt 2002)

5. ‘Sensor-Based Context Recognition for Mobile Applications’ (Mäntyjärvi 2003).

In this dissertation the related work has been categorised into three domains according to the emphasis. These categories are context frameworks, context representation and ontology, and context abstracting and recognition. The rest of this chapter follows the categorisation. Concerning the related dissertations, the first three are categorised as context architecture-oriented works, while Mäntyjärvi (2003) has a data mining view. Schmidt (2002) discusses aspects from all the categories, but has an emphasis on prototyping context-related applications. The relevant differences between this dissertation and the others are revisited later in the literature review in chapters corresponding to the topic.

2.2 Context frameworks

2.2.1 Definitions

The IEEE Standard Glossary of Software Engineering Terms (IEEE Std 610.12-1990, 1990) defines architecture and architectural design as follows:

Architecture. The organisational structure of a system or component.

Architectural design. (1) The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system. (2) The result of the process in (1).

A more detailed definition for architecture is given in the IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Std 1471-2000, 2000).

Architecture: The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

Software framework is more than an architecture, it is a reusable architecture. Wikipedia (2005) abstracts the essence of a software framework.

A software framework is a reusable design for a software system (or subsystem).

These definitions apply in this dissertation. To summarise, the term software framework is used to refer to an implemented architecture, which provides a reusable support structure for application development. Support structure refers to an organization of software elements, such as code libraries and API, which can be reused when new applications of the certain type are built. In this case, the software framework aims at providing an organization of reusable elements for building context-aware applications, which is why it is called a context framework.

2.2.2 Widget-based architecture model and Context Toolkit

The most referenced work in context architectures is presented in the Ph.D dissertation of Dey (2000), where the author gives a detailed description of the Context Toolkit architecture. Three main problem areas are identified in developing context-aware applications. First, the notion of context is not well defined. Second, there are no systematic common models and methodology for developing context-aware applications. Third, no tools exist for supporting application development (Dey et al. 2001). The authors notice that most context-based applications are location-based services, and, apart from them, there has been relatively little progress over the past few years. Lack of understanding of what constitutes a context, and how context should be represented are seen as problems that restrain advancement. Empirical investigations on the effect of context awareness on interaction and social issues suffer from the lack of versatile applications. Providing the means for more systematic application development is seen as the next step in facing these challenges. Hence the author proposes a conceptual framework and a toolkit, called Context Toolkit, for supporting prototyping context-aware applications.

The basis for Context Toolkit is adopted from the GUI (Graphical User Interface) paradigm, and GUI toolkits were used as an underlying model. The guiding principle in the design was to separate the acquisition of context from the use of it. Other primary requirements were to support interpretation of context, distributed communication, constant availability of context, context storage, and resource discovery. The requirements, defined for a networked PC-based system, are necessary, but incomplete. More detailed requirements need to be specified to address the restrictions and additional characteristics of mobile computing. Context Toolkit consists of five components that provide applications with the functionality for handling context. These are Widget, Aggregator, Interpreter, Widget service, and Discoverer (Figure 1).

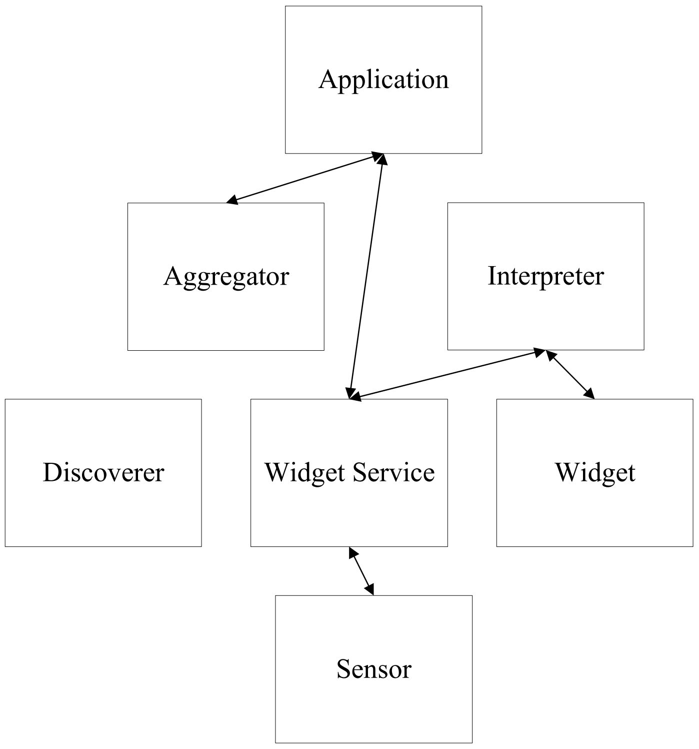


Figure 1. Context Toolkit architecture. The arrows represent typical interaction between components.

Widgets are attached to sensors and provide context for applications, thus separating context acquisition from its use. Aggregator collects several pieces of context into the same place, so that the application will not have to fetch them from several components. Interpreter takes context(s) as input and transforms them into another context. Widget service provides services the application may execute, taking context as input. Widgets, Aggregators and Interpreters register themselves with Discoverer. When an application is started, it contacts Discoverer to locate components that are relevant for it (Dey et al. 2001). Different elements are required in the context framework for mobile computing.

2.2.3 Client-server architecture model

The architecture of Dey et al. (2001) represents one of many ways to approach the context architecture problem. A number of models have been proposed for coordinating multiple inter-operating components. Winograd (2001) divides these models into three groups: widget model, client-server model, and blackboard model. Widget model is adopted from the architecture of GUIs. On a GUI, for example, a scroll bar is a widget, a high-level abstraction that can be used by the application, hiding the details of controlling hardware peripherals. A more flexible model is a client-server architecture, where high-level components are separate communicating entities. There is no central manager to keep track of services locally, and each component contains code to manage its connections, adding to the complexity of the component. The cost of finding independent services and communicating with them is higher than in a centrally managed process, but, in turn, components are more independent. An application that needs a certain service can either use a direct address (configured) or run a discovery process with the description of the service.

Hong and Landay (2001) give an example of a (networked) client-server-based context architecture model. The authors describe a service infrastructure for context awareness and the key technical challenges that must be addressed before such an infrastructure can be built. Three benefits of the service infrastructure approach are identified. First, the platform independency of the infrastructure allows a wide variety of devices and applications to access the services. Second, sensors and services that provide context are decoupled from one another, allowing both to be upgraded dynamically while the system is still running. Third, devices can be simpler since they can use infrastructure resources. However, in their discussion the authors completely ignore the existence of applications that require context information, possibly rapidly changing, directly from the sensors embedded in a mobile device. A complete architecture is required to have the ability to handle context from sources in the device and from sources in the infrastructure. As challenges, Hong and Landay (2001) see defining context representation and proper network protocols, creating basic services such as automatic path creation and proximity-based discovery, finding a balance between device and infrastructure responsibilities, security and privacy, and scaling up the infrastructure.

It should be noted that it is common for features from multiple architecture models to be combined. For example, the system may have a central blackboard (server) in the network, and clients that communicate with the server. On the other hand, the mobile device may have an internal client-server architecture. Clients (applications) can use services offered by, e.g., a device platform system server, which can be blackboard based.

2.2.4 Blackboard-based architecture model

The blackboard architecture model (Engelmore & Morgan 1988) is a heritage from AI research. The blackboard model is the third model discussed by Winograd (2001), and suggested as an alternative context management architecture. In contrast to the client-server model, service discovery is not necessary in the blackboard model. The viewpoint is data-centric rather than process-centric as in the other two models. Instead of sending requests to distributed components and receiving responses from them, a process sends messages to a common message board, the blackboard, and can subscribe to receive new messages matching a specified pattern. All communications go through the blackboard and are managed by a blackboard manager, and thus the communicating components can be less complex than in client-server architecture (Winograd 2001).

Winograd (2001) sets criteria for comparing the suitability of the three different models (widget, client-server, blackboard) for context management. The underlying architectural model affects each of the following criteria.

- Efficiency: The time efficiency is the most important efficiency criteria for interaction applications utilising context information.

- Configurability: Configurability is difficult to measure, but often critical in complex systems. It should, e.g., be possible to plug-in and modify components to the system without rebooting.

- Robustness: Robustness measures the ability of the system to handle and recover from error situations. A robust system must continue to function if components malfunction, are disconnected, send inappropriate data, or are restarted.

- Simplicity: Since humans build the systems, the key criterion is simplicity. For example, in the World Wide Web, the HTML and HTTP protocols are less powerful than their predecessors, but their simplicity enabled a large base of programmers to utilise them.

Winograd (2001) proceeds that a tightly coupled widget model, where a component and the application that uses it are compiled together, is most efficient but requires complex configuration and is not robust to failures. The blackboard model is less efficient in communication, since every communication requires two hops and uses a general message structure that is not optimised for any particular data or protocol. However, the blackboard model requires little configuring for the components, is more robust to failures, and offers simplicity provided by a uniform communications path. The client-server model has its strengths in simplicity and robustness, but requires heavier configuring – i.e., protocols for finding ports or resources and establishing connections.

Furthermore, Winograd (2001) discusses a blackboard-based architecture called Interactive Workspaces (Fox et al. 2000), and compares it with the Context Toolkit architecture with examples, one shown in Figure 2. The blackboard has two levels of data, Event Heap for short-term events and Context Memory for storing XML encoded data, which is relevant in the longer term and across applications. In Figure 2 a variety of components by Dey et al. (2001) (Interpreter, Discoverer, Aggregator, etc.) have been replaced by the shared blackboard (Event Heap, Context Memory). In the active badge example application events are generated upon a badge entering or leaving the space. Events are sent to Event Heap by a process associated with each location sensor. The active badge application is subscribed to these events. The application does not need to deal with a collection of widgets or aggregators. The resulting system is simplified compared with the widget-based approach, with no need to set up connections to multiple components.

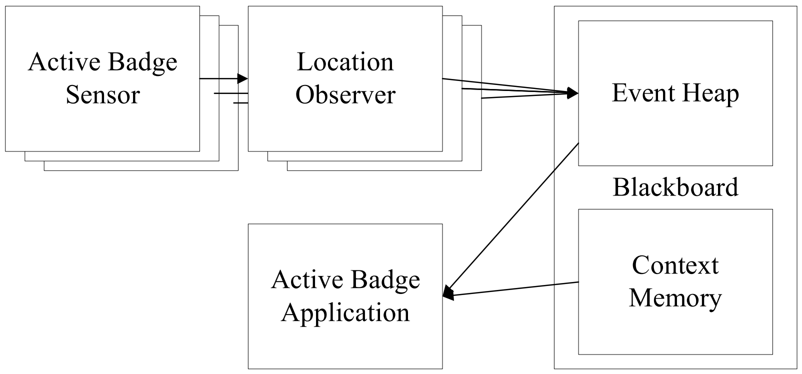


Figure 2. An example of an application that uses two-level blackboard architecture.

The main metrics for choosing the blackboard architecture were robustness, configurability and simplicity. Winograd (2001) proceeds that efficient communication is important, but given the increasing processor speeds, an architecture that avoids the complexity of configuring point-to-point communication paths can serve all but a few specialized uses that require tight action-perception coupling. Simplicity is achieved by avoiding having protocols for finding ports or resources and establishing connections. Robustness is two-sided. On the one hand, the functioning of the system depends on the functioning of the blackboard component, which must be built as reliably as any operating system component. On the other hand, the failure of components that produce information or use the blackboard has no critical effect on the overall functioning of the system (Winograd 2001).

Additionally, the centralized nature of the blackboard provides significant advantages in, e.g., context history management. For example, if many distributed widgets were responsible for producing an information entity, they would all have to be separately queried if the client required the past instance of that information entity, which would require all the widgets be available, finding them, establishing connections and collecting responses from many sources. Hence Winograd (2001) argues that the blackboard architecture is the most suitable model for context management, but does not proceed to develop and evaluate a context framework. Moreover, the author does not address context recognition, common structure for context abstractions, context management API, application control, and customization.

2.2.5 Architectures related to context management

Schilit (1995) presents a system architecture for context-aware mobile computing in his Ph.D dissertation. This early work at Xerox PARC has been influential for the further development of architectures for context awareness. The architecture was utilised for implementing applications based mainly on the use of location context, such as locating the nearest printer, and displaying a message on a display, which is close to the user. The architecture had three main components: device agents, user agents and active maps. Device agents maintained the status and capabilities of the devices, user agents maintained the user preferences, and active maps maintained the location information of devices and users. Context information was tightly coupled into the architecture components – i.e., adding new types of context information would have required implementing new device and user agents. Mechanisms for querying and notification of context information were supported. However, issues such as context recognition, storage, a uniform structure for context abstractions, context management API, application control, and customization were not addressed. Schilit's work focused on demonstrating that it is possible to build context-aware applications, whereas this dissertation focuses on providing and evaluating a general framework for developing mobile context-aware applications.

Context-service-based architecture is described in the Ph.D dissertation named 'Supporting the Development of Mobile Context-Aware Systems' (Mitchell 2002). The dissertation describes a context-aware tourist guide system named GUIDE, which can be viewed as a central Web server accessible by clients via a network. Additionally, the system contains geographically local caches, cell servers, which are used instead of the central server where possible. The system uses broadcasting, so that users entering a cell automatically receive (to their device) the most frequently accessed pages for that cell, to reduce response times. The scalability and applicability of this approach is not analysed in detail, although it is mentioned that a high uniformity of page requests is required for this approach to be useful.

Furthermore, based on the critique on GUIDE and the literature, the author derives a set of requirements and design for a more generic service-based architecture model, where context services provide an interface for accessing context information over the network and enable sharing the context among the applications. Services define an interface, which potentially provides the possibility of reuse. Figure 3 presents an overview of the architecture.

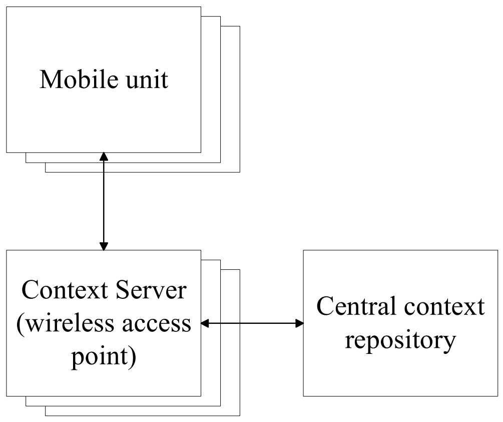

Mobile unit
Context Server (wireless access point)
Central context repository

Figure 3. Context-service architecture overview.

Context service providers reside in the user device or in the network server, depending on the case. The repository acts as a context data persistent storage. The architecture includes a concept of discovery of context services, and a concept of context abstractions (under the context service provider entity). The context service provider acts as an application agent, managing context for the application device and the discovery of new context services. Context services are networked abstractions that provide context information from physical or virtual entities, for use through the service provider.

Mitchell (2002) has a networked PC-oriented approach, as opposed to the mobile device-centric in this dissertation. Methods for abstracting context from multiple (sensor) sources are not addressed. The focus on context usage is on location, and other sensor-based context types receive no detailed attention. The general representation defined for context is rough, consisting of a hash table of name-value pairs, and the representation and the use of context have no clearly defined connection (e.g. as in ontology and API). The author's main focus is on networked context services and service discovery. Customization and context-based application control are not addressed.

The literature presents multiple other frameworks with a networked PC approach. Context-Aware Sub-Structure (CASS) middleware is a server-based framework for networked computers containing context-aware applications (Fahy and Clarke 2004). Sensor nodes are computers with sensors, connected to a server through the network. The CASS approach hence resembles the Multi-

User Publishing Environment (MUPE) (Suomela et al. 2003). MUPE is an open-source software framework for building multi-user context-aware applications. Context information can be shared through a network between devices by using Context Exchange Protocol (CEP) (Lakkala 2003a). MUPE has a networked blackboard-based context engine with a few features reminiscent of the features published by Korpipää et al. (2003b), such as support for context requests. There are many differences however. The MUPE context engine is designed to handle context information from networked sources, where context information is bound into entities, e.g. users. The response time requirements are much lower than in terminal context management. Contexts are produced in compound structures having multiple context types, whereas in terminal context management they are handled as single instances with one context type in each. The frequency for most sensor-based context types can be very high, and for others, such as location, it can be very low. Producing and handling context instances in large compound structures in a mobile device would thus create unnecessary data traffic. Database functionality is not provided in the MUPE context engine, but the latest context is stored for each entity. Context recognition, customization and context-based application control are not addressed.

Yau and Karim (2001) have built context-sensitive middleware on CORBA (Component Object Request Broker Architecture). They emphasise real-time establishing and terminating of ad hoc communication between distributed and mobile objects. Communication between objects is managed based on simple contextual data from network, device, and user interaction. Context information of objects can be defined using a logic-style rule language. Matching context, for example the range between devices, activates a method that establishes or terminates communication and starts data transfer between objects. CORBA defines an interface definition language (IDL) and application programming interfaces (API) that enable client/server object interaction within object request brokers (ORB). The focus of the study is in ad hoc networking and communication between a mobile device and the environment, with the aim that the environment, with its services, is context aware, whereas this dissertation aims at enabling context-aware mobile devices. CORBA is designed for distributed client/server communication, whereas in this dissertation the client and server are both local to a mobile terminal.

Mandato et al. (2002) describe a concept of a context-aware Internet portal, which attempts to combine Internet portal technology with personal mobility, terminal adaptation and context awareness concepts. Their central idea is to provide users with access to a variety of services, which are automatically adapted to the user's context. User preferences are considered part of the context. Physical variables, user activity, quality of service, and user status (such as mood) are envisioned as other sources of context. However, no accurate means for acquiring that kind of context is suggested. As a solution to the problem of nearby service discovery, they propose a local service portal, which would be connected to an Internet portal. Internet service discovery mechanisms, such as Jini, could then be applied. This solution would require, in addition to the existence of local portals, that location information be transferred as a part of the Internet protocol. The concept-level system is network infrastructure-oriented as opposed to mobile device-oriented.

Gaia is an infrastructure for context awareness based on first-order logic (Ranganathan & Campbell, 2003). It is distributed by a client server architecture, which has similarities to the Context Toolkit architecture. Context providers (Widgets) collect various types of contexts and can be queried by context consumers (Applications). A context synthesiser (Aggregator) contains logic rules that form new contexts from existing ones. A context provider lookup service (Discoverer) is used by the consumer for finding the context provider able to produce contexts of an appropriate type. In the blackboard model, such a lookup service is not necessary. Context history is stored in a database, except for context synthesisers. The blackboard model, having a central data storage, makes managing a context database more straightforward. Communication between distributed entities is done using CORBA. Components of the system can be distributed and discovered using the CORBA naming service and CORBA trading service. The authors do not report on processing actual sensor measurement data with the infrastructure. The choice of logic as the only modelling and inference language has the advantages of expressiveness and formality, but disadvantages of inference inflexibility and uncertainty handling. Furthermore, a method based on logic is provided for specifying rules for application control based on contexts. The system has a smart-space infrastructure-oriented approach as opposed to mobile device-centric.

Chen et al. (2003) discuss architecture for supporting context-aware systems. The authors propose a Context Broker Architecture (CoBrA), which uses Semantic Web languages and tools for managing and sharing context information. The architecture has a core server entity called Context Broker, which has the following responsibilities: provide a centralised model of context, acquire contextual information, reason about contextual information that cannot be directly acquired from the sensors, detect and resolve inconsistent knowledge that is stored in the shared model of context, and protect user privacy. The design of CoBrA is aimed to support context-aware systems in smart spaces, and each smart space is assumed to have a designated central context broker. By the choice of design, CoBrA is infrastructure-centric, whereas the framework proposed in this dissertation is mobile device-centric, where no additional equipment for a mobile device itself is required for system operation.

Wang et al. (2004) present an infrastructure for managing context information related to a smart space, e.g., a room, where the processing is performed by computers distributed in the environment. Each smart space requires its own infrastructure, which can be connected to each other. Context objects are produced by context wrappers, which use Universal Plug and Play (UPnP 2005) to publish context changes as events to which clients can subscribe, and an API is provided for context information access. Korpipää et al. (2003b) describe a context change subscription mechanism for the blackboard-based context framework of a mobile device, and a simplified API for accessing context information. Wang et al. (2004) apply Semantic Web–based tools for context reasoning, the Jena2 generic rule engine (Carroll et al. 2003), and, for querying, RDF data query language (Miller et al. 2002) from a context knowledge base in a PC. The authors have evaluated the system performance with a 2.4 GHz Pentium 4 workstation with 1.0 Gbyte of RAM and report that with the prototype Java-based application, the reasoning delays (about one second) sometimes matter to users. It can hence be extrapolated that the mentioned Semantic Web-based methods are not yet feasible for mobile computing – i.e., for use in a mobile terminal software framework. The authors identify as future work providing the ability to manage context information uncertainty with reasoning methods such as probabilistic logic, Bayesian networks, and fuzzy logic, since sensor-based contexts are not always precise. Sensor-based context recognition and customization are not addressed.

Genie of the Net (Riekki et al. 2003) is an agent-based architecture that is used as a component of an intelligent environment. An intelligent environment is defined as an environment that serves the user by providing, or automatically using, services that are useful in the situation at hand. The environment contains sensors, actuators, user interfaces, devices for information storage and computation, and other information services. The authors identify as an important problem a service overload, a situation where the number of services hinders their feasible utilisation. Genie is proposed as a component of an intelligent environment, which manages the services on behalf of the user by requesting services from the environment. As environment-oriented, the focus of the architecture differs substantially from the mobile device-centric approach in this dissertation. The prototype context recognition subsystem in Genie is based on a CORBA notification service. The system contains Producers, Filters, and Consumers. Producers send sensor data into a system channel, and Filters read the data from the channel, process it and send the results back into the channel. Context management, recognition, API, and context information representation are not discussed by Riekki et al. (2003). The authors identify the need for a common representation and exchange format for context information, which are seen as targets for future work.

Moreover, blackboard is a widely used model in middleware for communicating data between sensors and applications in ubiquitous computing. Some approaches use the notion of Agent. Agent is one name for an abstraction used to represent objects such as, for example, sensors and services. Adaptive Agent Architecture (Kumar et al. 2000) and Hive (Minar et al. 2000) are agent architectures that use a central blackboard to deliver data from agents representing sensors to agents representing applications. Both have a smart space-oriented approach.

2.2.6 Customization

Context frameworks can offer programming abstractions for the programmer of context-aware applications. The programming interface can be further abstracted, so that it becomes possible for users to define context-aware features. This kind of software development is referred to as end user development (EUD).

According to Fischer et al. (2004), EUD aims at a low cost of learning in software development while maintaining as wide scope as possible. To reach the low cost of learning, the aim is to decrease the conceptual distance between actions in the real world and programming. Figure 4 illustrates the relationship between the existing means of software development and EUD. The figure has been modified from an illustration given by Fischer et al. (2004). Hence customization can be viewed as a form of end user development.

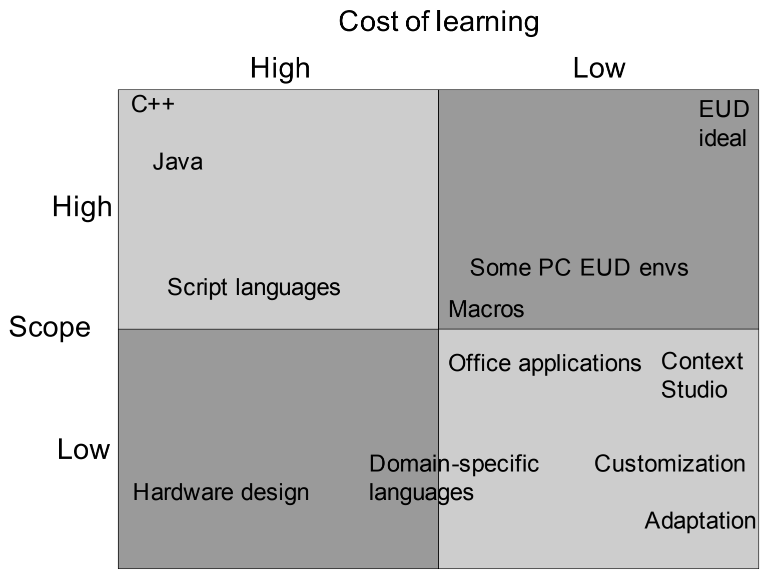


Figure 4. End user development aims at a wide scope and low cost of learning.

The idea of specifying context-based actions has been initially discussed from the user viewpoint in the literature. A concept of personalising context-aware mobile device applications was first introduced by Mäntyjärvi et al. (2003). According to the original concept, a set of context types and values were presented to the user, who marked a context value for all available context types as a description of a certain situation as a trigger for a certain action. The concept was tested with a plain UI (not designed for small-screen devices) in a PC environment, and no software framework was used to enable the actual execution of the defined context-action rules. Defining a context as a union of all context types led to complex rules, and the users had difficulties in specifying a set of correct rule triggers. In this dissertation, the concept is modified, further developed, applied with the enabling software framework, and the new concept is evaluated.

Other related work of customizing context-aware behaviour mainly discusses prototypes developed for networked PC environments. Ranganathan and Campbell (2003) acknowledge the need for a graphical user interface for defining context-aware features that would enable the user to specify context-action rules instead of writing first order logic. Sohn and Dey (2003) discuss an informal pen-based prototype tool that, in a PC environment, lets users configure input devices that collect context information and output devices that support response. Inputs and outputs can be combined into rules and tested with the tool. The goal of enabling both designers and end-users to create and modify context-aware applications is identified as a topic for further research.

Dey et al. (2004) experimented with an approach of programming-by-demonstration for prototyping context-aware applications. The authors have developed a tool for a PC environment that allows the user to train and label models of context, which can be mapped to actions. Context models are represented as examples. Modelling based on examples is feasible when it is performed for a single chosen type of context. In the case of multiple input sources, the programming by demonstration approach may lead into functionality that the user did not intend to have, if the user cannot control exactly which inputs define the situation.

Truong et al. (2004) present an approach for end-user programming of applications involving automated capture and playback of home activities. The authors describe a system for end-user programming for smart environments based on a magnetic poetry metaphor. The user interface for the system, developed for large-screen PC environments, contains a predefined domain-specific set of words, which the user can arrange to define system behaviour. Each user-defined application must include a setting for time, duration, frequency, location, and people to be functional in the system. This may result in complex definitions. Moreover, since the set of parameters used by the system is different from the available words, the descriptions the users make do not always correspond to the behaviour the user intended to have. The available context information in the system is time and people at a certain location. The authors report that the current version of the system allows the description of a single application.

2.2.7 Conclusions

The state of the art in context frameworks and customization was reviewed. The related work primarily discusses prototypes and experiments performed with PCs or laptop PCs connected in distributed environments. The viewpoint of the related work is mostly environment-centric, i.e. the frameworks are designed so that context information is processed in the environment infrastructure instead of the terminal. This is the fundamental difference to the focus of this dissertation. Therefore, the related work provides no solution to several issues central to this dissertation concerning a software framework designed for processing sensor-based data in a mobile device itself. Such mobile device-centric issues are, e.g., the lack of framework support for fast-changing event-based abstracted contexts, context abstracting and recognition process, an application programming interface for using rapidly changing sensor data, blackboard-based management of context information, and application control and interaction customization in a mobile device. Concerning customization and personalisation, the related work does not provide an easy-to-use tool for, nor enable, end-user development of context-aware applications in a mobile device, human-computer interaction customization in a mobile device, and customizing multimodal interaction of sensor-based input modalities in a mobile device. Moreover, the related work does not provide implementation and evaluation of the mentioned issues with a set of real applications implemented in a real handheld mobile device, such as a mobile phone. These are among the novel issues to be addressed in this dissertation.

Despite the mentioned different focus the related work discusses many relevant requirements for context frameworks and context processing in general, and introduces the concept of personalising context-based applications. The related work also provides an insightful comparison of architectures with a suggestion for a superior architecture model for context management, the blackboard model.

2.3 Context representation and ontologies

2.3.1 Definitions

In order to provide context information through a context framework for the applications, a uniform way of representing and sharing context needs to be designed. Ontologies have been widely studied in knowledge engineering, artificial intelligence, and computer science literature. Depending on the science, ontologies have been very differently defined and applied. Gomez-Perez et al. (2003) gives a throrough discussion of the categorisation and main types of ontologies.

Motivation for using ontologies at design time is given by Guarino (1998): "It enables the developer to practice a 'higher' level of reuse than is usually the case in software engineering (i.e. knowledge reuse instead of software reuse). Moreover, it enables the developer to reuse and share application domain knowledge using a common vocabulary across heterogeneous software platforms."

Ontologies have been defined from different viewpoints. Ontology has been described as defining basic terms and relations comprising a vocabulary, but also rules for combining the terms and relations. Moreover, ontology has been defined as a logical construct, and as a set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base (Gomez-Perez et al. 2003).

In this dissertation, the ontology is not built by using logic, nor does it contain rules, nor is it used as a part of a knowledge base in the traditional sense. Rules or inference mechanisms are considered as separate entities that use the ontology but are not defined in it. A general widely cited definition for an ontology is given by Gruber (1993).

An ontology is an explicit specification of a conceptualization.

The definition is further elaborated by Studer et al. (1998).

An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world *by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group.*

Moreover, Gomez-Perez et al. (2003) discuss the division of ontologies into lightweight and heavyweight, based on the degree of "depth" and restrictions (axioms, constraints) on domain semantics. The ontology in this dissertation is considered a lightweight ontology, including concepts, concept taxonomies, and properties (structure) that describe concepts, but no explicit constraints. According to Gomez-Perez et al. (2003), ontologies can be characterised according to the level of formality to highly informal (natural language), semi-informal, semi-formal, and rigorously formal. According to the definition of Studer et al. (1998), highly informal ontology is not ontology since it is not machine-readable. The ontology in this dissertation can be considered semi-informal since it is expressed in a "restricted and structured form of natural language" (Gomez-Perez et al. 2003). Furthermore, if a formal language is chosen for the ontology, it becomes formal to a degree.

In general, declarative knowledge is modelled by ontologies while problem-solving methods (PSM) (Gomez-Perez et al. 2003) specify generic reasoning mechanisms. Generic reasoning mechanisms are not addressed in this dissertation. The aim is to build extensible models of reusable knowledge that can be used by any reasoning mechanism. Any reasoning or recognition entity should provide the information it contributes according to the structure defined by the ontology, but the reasoning mechanisms themselves are not restricted. In this dissertation the representation research problem is focused on finding a sufficiently expressive level of description that is specific enough for some subset of applications to enable inferring appropriate action, but, at the same time, simple and clear enough for adequate genericity and understandability.

Context ontology refers to ontology for describing context information. The ontology in this dissertation has two parts: structure and vocabulary. The structure defines the common properties that are used to describe concepts across domains. Vocabularies define concepts and concept taxonomies. Vocabularies are domain-specific. When context-aware applications are built with the same ontology, the underlying structure is shared across different applications. Vocabularies can be extended, or new vocabularies can be created, covering new domains.

2.3.2 The Semantic Web

The Semantic Web (Berners-Lee et al. 2001) is a Web that utilises methods and languages for representing the structure and semantics of information in order to enable more efficient and reliable processing of content with machines. Semantic Web is under development, and related technologies are being developed by the World Wide Web Consortium (W3C). A central concept of the Semantic Web is that distributed computers have access to structured collections of information and sets of inference rules that they can use to conduct reasoning. This concept is familiar from the AI research. The W3C Semantic Web (2001) definition is the following.

The Semantic Web is the representation of data on the World Wide Web. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming.

A central component of the Semantic Web is ontologies, collections of information. An ontology for the Web is a document or a file that formally defines relations among terms. A typical ontology for the Web contains a taxonomy and a set of inference rules. The taxonomy defines classes of objects and relations among them, and properties of classes. Inference rules can be used for deducing actions based on objects and their properties (Berners-Lee et al. 2001).

The relevance of the Semantic Web to context ontologies in mobile computing is that, primarily, it potentially offers formal description formats for describing and sharing context ontologies. Secondarily, it offers potential methods for reasoning based on context information. In the Semantic Web, ontologies represent static domain knowledge and PSMs are used in Semantic Web Services that deal with that domain knowledge.

In this dissertation the discussion on reasoning is focused on context recognition. Application control reasoning mechanisms are not discussed in detail. Pattern recognition methods are applied for recognising context from multiple sensor sources. The focus of the discussion concerning the Semantic Web in this dissertation is on the methods for representation, not reasoning.

The rest of this chapter briefly reviews the central languages and methods related to the Semantic Web effort that are the potential for utilising in describing, sharing, and reusing context information between mobile devices and developers.

HTML

The current Web is mostly encoded in HTML (HyperText Markup Language) (Raggett et al. 1999), which is mainly designed for describing how content is displayed for human viewing in browsers. HTML is not suitable as a context information representation language.

XML

HTML has been followed by an increase in the use of XML (Extensible Markup Language) (Bray et al. 2000) as an alternative and additional encoding. XML, and the many extensions and enhancements that have come with it, offers a structured way of describing metadata related to any content. XML lets the content creators define their own tags, which can be used to annotate the content. Tags can be utilised by programs for multiple purposes, but each program developer has to know what each XML document developer has used the tags for. However, XML is a universal multipurpose representation syntax that can be utilised for describing sensor-based context information as well as web content.

XML Schema

W3C (XML Schema repository 2001) offers the following definition for XML Schema: "XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide means for defining the structure, content and semantics of XML documents". The purpose of a schema is to define a class of XML documents, and an XML document that uses the definitions in the schema is called an instance document. A schema can be viewed as a collection, or a vocabulary, of type definitions and element declarations, whose names belong to a particular namespace. Namespaces are used to distinguish the definitions from different vocabularies. Hence two definitions having the same name but different structures can be distinguished with namespaces (Fallside 2001). XML Schema offers an extensive set of predefined data types and facilitates detailed description of complex structures. XML Schema is a potential language for representing the structure and properties of sensor-based context information. XML can be used to represent the instances corresponding to the schema.

RDF

RDF (Resource Description Framework) (Manola et al. 2004) enables encoding information about a structure in the form of a triplet, where each triple is similar to the subject, verb and object of a simple sentence. The triples can be used to describe concepts of related things. These triples can be written using, for example, XML tags. Each triple is identified by a Universal Resource Identifier (URI), a type of URL (Universal Resource Locator). A different URI is used for each specific concept. URIs ensure that the defined concepts are not just words in a document but are tied into a unique definition that is available for everyone in the Web (Berners-Lee et al. 2001, Ahmed et al. 2001).

RDF is particularly designed for representing metadata about Web resources, such as the title, author and modification date of a Web page, etc. However, by generalizing the concept of “Web resource”, RDF can also be used to represent information about things described on the Web, even when they cannot be directly retrieved on the Web. RDF provides a common framework for expressing information, so it can be exchanged between applications without loss of meaning. The ability to exchange information between applications facilitates the utilisation of the information by applications other than those for which it was originally created (Manola et al. 2004). Therefore, RDF is also a potential basic description framework for formally representing and sharing the sensor-based context information of a mobile device. RDF can be seen as a basic assembly language on top of which other information modelling methods can be overlaid.

RDF Schema (RDF-S)

RDF provides a way of expressing statements about resources, using named properties and values. RDF Schema (Brickley et al. 2004) provides the ability to define the vocabularies used in the statements. In other words, RDF Schema enables defining the application-specific classes and properties for resources described in RDF. XML Schema datatype definitions can be used as part of the RDF Schema descriptions. The RDF Schema-type system is similar to the type systems in object-oriented languages, such as Java. However, there are also significant differences. One major difference is that instead of describing a class as having a collection of properties that have values of a certain types, RDF Schema property descriptions are independent of class definitions and have a global scope. RDF uses domain for specifying the class of a property, and range for specifying the type of values for the property. RDF Schema property definition, without a domain class specified, can be used as a property description for any class. For example, an RDF schema could describe a property of weight, without a domain, and it could be used to describe instances of any class that might be considered to have a weight. RDF Schema provides basic capabilities for describing RDF vocabularies. Additional capabilities are possible and are defined in the RDF-based ontology languages such as DAML+OIL (DARPA Agent Markup Language+Ontology Inference Layer) and OWL (Web Ontology Language), which are both based on RDF and RDF Schema. The aim of these languages is to provide additional machine processable semantics for resources, i.e. enable more detailed representations of resources (Manola et al. 2004). RDF and RDF Schema form a potential basis for describing and sharing mobile context information. The ontology languages offer capabilities for creating more detailed representations.

DAML+OIL

DAML+OIL (DARPA Agent Markup Language+Ontology Inference Layer) (Connolly et al. 2001) is a semantic markup language for Web resources. It is based on RDF and RDF Schema, and provides a more expressive set of modelling primitives, similar to frame-based languages (Minsky 1988) in AI. DAML+OIL has been followed by a further developed Web Ontology Language, OWL (Ouellet & Ogbuji 2002).

OWL

OWL (Web Ontology Language) (Bechhofer et al. 2003) is a semantic markup language for publishing and sharing ontologies on the World Wide Web. Ontology is defined here as "a formal explicit description of concepts in a domain of discourse (or classes), properties of each class describing various features and attributes of the class, and restrictions and properties" (Smith et al. 2003). OWL ontologies are usually stored on Web servers as documents, which can be referenced by other ontologies and downloaded by applications for use. OWL is developed as a vocabulary extension of RDF and is derived from the DAML+OIL (Bechhofer et al. 2003). OWL facilitates greater machine interpretability of content than that supported by XML, RDF and RDF Schema by providing additional vocabulary and formal semantics. OWL has three sublanguages, in the order of expressivity: OWL Lite, OWL DL, and OWL Full, of which OWL Full is the most expressive (McGuinness & Harmelen 2003).

The advantages and differences of OWL compared with XML and XML Schema, according to Smith et al. (2003), are the following.

- "An ontology differs from an XML Schema in that it is a knowledge representation, not a message format. Most industry-based Web standards consist of a combination of message formats and protocol specifications. These formats have been given an operational semantics, such as "Upon receipt of this PurchaseOrder message, transfer Amount of dollars from AccountFrom to AccountTo and ship Product." But the specification is not designed to support reasoning outside the transaction context. For example, we will not in general have a mechanism to conclude that because the Product is a type of Chardonnay it must also be a white wine."

- One advantage of OWL ontologies will be the availability of tools that can reason about them. Tools will provide *generic* support that is not specific to the particular subject domain, which would be the case if one were to build a system to reason about a specific industry-standard XML Schema. Building a sound and useful reasoning system is not a simple effort. Constructing an ontology is much more tractable. Third party tools based on the formal properties of OWL can be made that may be useful in processing the ontologies.

OWL is one of the potential formal languages for describing mobile context ontologies. However, as the authors acknowledge, building a generic reasoning system is not a simple effort, and as the more expressive languages such as OWL Full are concerned, it is unlikely that any reasoning software will be able to support every feature of OWL Full (Smith et al. 2003). Domain knowledge is usually only valid in a certain domain. If describing certain domain knowledge requires an expressive representation, and no generic tools can be built for reasoning based on such representation, the advantages of OWL over XML Schema and XML are limited. Moreover, the study by Wang et al. (2004) implied that the applied reasoning system operating on OWL was computationally heavy.

CC/PP

A CC/PP (Composite Capabilities/Preference Profiles) profile is a description of device capabilities and user preferences, also referred to as device's delivery context, that can be used to guide the adaptation of content presented in that device. In CC/PP, RDF is used to create profiles that describe user agent capabilities and preferences. Here, profile refers to the document(s) exchanged between devices that describe the capabilities of a device. A CC/PP profile contains a number of attribute names and associated values that are used by a server to determine the most suitable form for a resource to deliver to a client. CC/PP vocabularies consist of a set of attribute names, permissible values and associated meanings. If different applications that have different vocabularies need to work together, a common vocabulary or a conversion method between vocabularies is required. The CC/PP profile is structured as a two-level hierarchy, so that each profile contains one or more components and each component contains one or more attributes. Components could be, for example, hardware platform, software platform and an application, such as a browser. Attributes could be, for example, display height and width, operating system vendor, version, etc. (Klyne et al. 2003).

In addition to static device capabilities, the early application focus of CC/PP, it can also convey information about user preferences that should allow Web servers to improve the accessibility of Web sites (Klyne et al. 2003). In this dissertation, CC/PP is categorised as means of exchanging static context information, e.g. device settings. Static context information is not within the scope of this dissertation.

2.3.3 Information models related to context-aware computing

Fairly few studies concern models and ontologies for representing sensor-based context information from the mobile device viewpoint. Schmidt et al. (1999a) introduce a three-dimensional context space, with the dimensions Environment, Self and Activity. Environment is further divided into Physical and Social; Self is divided into Device state, Physiological and Cognitive; and Activity to Behavior and Task. A context is defined as a description of the current situation on an abstract level. Context is derived from cues, which are derived from sensor measurements. Context is described by a set of two-dimensional vectors, which consist of symbolic values for context and number values for indicating the certainty of the context. Hence the authors introduce two basic properties of context representation structure. In this dissertation, the representation is further developed and formalised to provide a more expressive set of properties as the base representation structure for context.

Schmidt et al. (1999b) propose a model of context in which each context is identified by a name, and for each context there are a set of features whose values describe context. Furthermore, the authors propose a hierarchically organised feature space, where at the top level two categories are identified: context related to human factors in the widest sense, and context related to the physical environment. Both of the categories are further divided into three categories: User, Social environment, Task; and Conditions, Infrastructure, Location. These six categories each contain a set of relevant features that are further divided into categories, so that the final category tree has a depth of four. The values of these features describe context. History of contexts is proposed as another context. In this dissertation the structure and model for context expressions is further developed and formalised; extensibility, vocabulary model, and naming conventions are discussed from a mobile computing viewpoint. Moreover, the representation is bound to the real use of context data through a software framework.

Furthermore, Schmidt (2002) presents a model in which contexts are bound to physical entities, such as Athlete, Hand, Coffee cup, and the entities have a type, such as Person, Body part, Artefact, Table 1. Such a model is natural for describing context related to different kinds of artefacts and physical objects. When the focus of modelling is centered on a mobile device and its user, a different model is appropriate. Moreover, for systematic use of context information within a software framework, vocabulary model and naming policy need to be addressed.

Table 1. Context categorization for physical objects.

Type of entity	**Entity**	**Examples of typical contexts**
Person	Athlete	Running, Walking, Sitting, Cycling
Body part	Hand	Moving, Moving fast, Still
Artefact	Coffee cup	Empty, Hot, Cold

Dey and Abowd (2000) model context with a categorization into four classes: location, time, activity and identity. Context vocabularies consist of lists of widgets, sources of context. The authors do not provide a uniform structure for presenting context as data objects, or a vocabulary model. Crowley et al. (2002) discuss a computing process-based approach of representing conceptual components. They bind context entities to processes, and by that choice their system is similar to the widget-based approach but differs in focusing on the transformation of information from measurements to contexts, or from contexts to other higher abstraction-level contexts.

Winograd (2001) emphasises the importance of creating ontologies for distributed environments that provide the application writer with a representation of the aspects of context that are relevant to program execution. Furthermore, the goal is seen as finding the right level of description, which abstracts away from implementation details but is still specific enough to enable inferring appropriate actions based on context. The author points out that the most difficult part of the design will be the conceptual structure, not the encoding, which is fairly straightforward once what to encode is understood. The author proceeds: “The hard part will be coming up with conceptual structures that are broad enough to handle all of the different kinds of context, sophisticated enough to make the needed distinctions, and simple enough to provide a practical base for programming." This statement forms one of the major design principles and requirements for the ontology developed in this dissertation.

In more recent related work on context ontologies, other than location, Semantic Web technologies are applied to the knowledge management of smart spaces (Wang et al. 2004). The authors present an infrastructure and ontology for describing context information related to a smart space, e.g. a room, in which the context is managed by computers distributed in the environment. The authors use OWL for describing a context ontology that consists of an upper-level context ontology and extended context ontologies. The upper-level ontology provides a set of basic concepts across different environments. The upper level ontology contains three classes of real-world objects (user, location, and computing entity, which has a subclass device) and one class of conceptual objects (activity) for characterising smart spaces. Korpipää and Mäntyjärvi (2003) propose a set of common basic properties for *all* context objects, with support for handling context information uncertainty. Wang et al. (2004) describe the class-specific properties required for describing each of the mentioned context subclasses.

Conceptual modelling approaches have also been used to represent context. For example, UML and ER modelling is applied by Henricksen et al. (2002) to model contexts and some of their features, such as temporal characteristics. The authors find that UML and ER are neither natural nor appropriate for describing context; instead, they propose using special constructs designed for the characteristics of context information.

Ranganathan and Campbell (2003) introduce a context model based on first order logic. The model describes the properties and structure of context information and the kinds of operations that can be performed on context. Contexts are represented as first order predicates, and ontology is used for specifying the structures of different kinds of predicates, as well as checking that the provided context expressions are valid. Furthermore, rules can be specified for inferring new contexts from existing ones, and inferring application functions based on context. Categorisation is not discussed in detail; the authors report on having used context providers such as location, weather, stock price, calendar, and authentication. The choice of first order logic as the modelling language brings certain advantages and disadvantages. The language is formal and expressive, but lacks properties such as uncertainty handling in inference, which is essential, particularly with sensor information. Furthermore, choosing only one method for modelling and inferencing is restrictive, since different methods are suitable for different tasks.

Context ontology described in RDF is discussed by Toivonen et al. (2003). The ontology is frame-based, consisting of classes and properties characterizing them. The ontology contains concepts for describing time, location, social aspects and device characteristics. The authors acknowledge that social contexts are very hard to acquire automatically, and the users may manually input that information. Static device characteristics are categorized as part of context information, but are not included in the ontology. Instead, UAProf specification (OMA 2003), which is based on CC/PP, is utilised for describing static device properties. A context aware portal is described, which uses context information for certain adaptive functions, such as message delivery based on time or location. The combining operators (and, or, not) used for adaptation purposes, and reasoning mechanisms, are not included as a part of the ontology to sustain modularity, a similar solution as in this dissertation. The authors state that the context ontology has not yet been utilised in practice but has been developed at the conceptual level. Upgrading the ontology from RDF to OWL is seen as further work.

Kofod-Petersen and Aamodt (2003) propose an open context model with a taxonomic structure of context types. The context model can be updated dynamically, and the model is linked to a domain model that enables semantic interpretation of situations. The model describes context as a hierarchy, where the uppermost node is the User Context. User Context has the subcategories Task, Social, Personal, Spatio-Temporal and Environmental. Personal Context is further divided into Physiological and Mental Contexts. The context model imposes a structure that all suppliers of context use. A specific domain model, in which the context model is integrated, is used to describe domain concepts that can be used for generalising situation cases. Furthermore, the authors present an architecture that facilitates the use of Case-based reasoning (CBR) (Aamodt & Plaza 1994) for identifying and acting upon situations that consist of instances of context information defined by the context model. Even though many categories are defined in the model, acquiring the actual context data is not discussed in the paper. For example, they do not mention how user Mental Context can be acquired. Furthermore, the number of potential situation cases in real-world scenarios is very high, which may cause problems with case management.

Extending UDDI (Universal Description, Discovery and Integration) with context-aware features is studied by Pokraev et al. (2003). The basic idea is to enhance Web services with semantic profiles that contain contextual information in order to be able to match the requested Web services to the context of the requesting party, which can be a mobile device. In other words, the aim is to be able to provide the user with services relevant to his context. The need for a common context model by means of ontologies for this purpose is acknowledged. The authors suggest an upper context ontology that facilitates context matching. In a high-level view, the ontology contains categories for Location, Time, Activity, Physical, Social, Network capabilities and Device capabilities. Existing external domain-specific ontologies are incorporated, e.g. CC/PP is utilised for Device capabilities. The category of Physical relates to sensor-based contexts, such as Temperature and Humidity. The authors recognise the highly dynamic nature of context in context-aware mobile applications, and the need to be able to match a relevant service based on more complex and less deterministic queries than in traditional Web service systems. A common structure of the context and vocabulary models is not discussed, nor is handling the information uncertainty.

Mitchell (2002) models context in the GUIDE system as an object, which includes a hash table of name-value pairs, timestamp, and an expiration time. Context domain semantics is discussed for modelling location, but not for other context types.

Vildjiounaite et al. (2003) study the context awareness of everyday objects augmented with sensing, communication and computation capabilities. The authors utilise the context structure by Korpipää and Mäntyjärvi (2003), and additionally model the properties of each object to which the context information is associated. The object types have domain-specific properties, e.g. the object type food has fat percentage, expiration date, etc. The viewpoint is smart space/object oriented.

Many models have been proposed for representing location information. Location ontologies are not the focus of this dissertation, and hence the discussion of the literature on the topic is omitted. However, the structure of the ontology described in this dissertation is designed to support location information.

2.3.4 Common sense context dimensions

The ideal context ontology for mobile computing would be a complete description of common sense, should it be possible to create, manage and infer such an ontology. The literature discusses attempts at formalising the general semantics of context, most importantly CYC (Lenat 1995, 1998). However, even though formalising all common sense formally would be a breakthrough for creating applications and devices with the ability of rational behaviour in the huge variety of real-life contexts, the challenges in achieving that goal remain very difficult.

Common sense research has produced a categorisation for context. Twelve dimensions, or context types, have been identified. Four of them concern time and place (Absolute Time, Type of Time, Absolute Place, Type of Place); others are more abstract, such as culture. Location and time are frequently used in mobile context awareness, but other dimensions are less exploited, largely due to the difficulty in detecting them automatically.

Moreover, common sense research has contributed a descriptive language for knowledge representation. CYC language (CycL) is a first order logic-based symbolic descriptive language developed to encode all common sense into a single knowledge base. The main differences compared with logic are the context mechanism and expanded, more expressive syntax. In short, the context mechanism is designed to divide the otherwise hugely inconsistent single KB into many small consistent KBs, or contexts, and inference is performed in a certain context or near it, enhancing efficiency.

As a context representation, CycL has the same weaknesses as logic, and it has more complex syntax. CycL is not as widely adopted. Concerning uncertainty, there are certain improvements compared with logic, such as Bayesian structures and persistence functions. CycL is a long-term effort aiming at formalising common sense, the lack of which is one of the main obstacles in creating human-like intelligence in devices (Lenat 1995, 1998, 1999a, 1999b; Lenat et al. 1995; Minsky 1988, 2000).

2.3.5 Conclusions

The state of the art in context representation and ontologies was reviewed. The related work contains semi-informal and formal approaches to modelling domains of context information. The most common domains for context modelling were smart environments and smart objects. Both of these domains concentrate on modelling the environment, some specific environment, or specific objects in it. The viewpoint chosen in this dissertation requires modelling the environment as far as can be sensed by the terminal itself. Terminal-oriented approaches describe static contexts such as device properties and context taxonomies for categorising the different context types that are considered relevant. This related work has contributed to this dissertation with the working model of how to categorise context values, and partly with basic properties of context information.

The existing formal approaches utilise Semantic Web-based methods or logic for context representation and inference. As discussed earlier in the review of context frameworks, the results on experiments with Semantic Web-based inference and querying methods such as the Jena2 generic Rule engine and RDF data query language, suggest that these inference methods are currently computationally too expensive to apply in mobile phones. However, from the viewpoint of this dissertation, Semantic Web-based languages offer a potentially useful formal syntax for specifying the constraints and other attributes of context vocabularies.

Due to the difference in the addressed principles, the related work lacks several specific topics of interest to this dissertation, such as the structure and common properties for domain independent representation of context information as data objects, a generic vocabulary model for describing context instances and vocabularies, and how to utilise context ontology in customizing context-aware mobile applications. Moreover, the related work does not provide implementation and evaluation of the mentioned issues with a set of real applications implemented into a real handheld mobile device.

2.4 Context abstracting and recognition

Two main types of inference can be done based on context information, context abstracting and recognition, and application control. Application control refers to performing application actions based on context information. The emphasis concerning inference in this dissertation is on context recognition.

2.4.1 Definitions

Context recognition is analogous to pattern recognition. Pattern recognition is defined by Theodoridis and Koutroumbas (1999) as follows:

Pattern recognition is the scientific discipline whose goal is the classification of objects into a number of categories or classes. Depending on the application, these objects can be images or signal waveforms or any type of measurements that need to be classified. We will refer to these objects using the generic term patterns.

Accordingly, in pattern recognition terms, context can be seen as a pattern. Duda et al. (2001) present the process of a typical pattern recognition system (Figure 5).

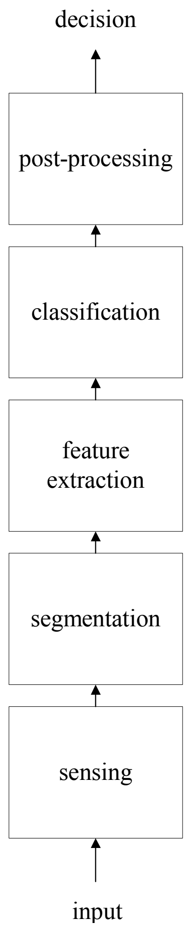


Figure 5. Typical flow of information in a pattern recognition system.

In Figure 5, the sensing phase converts physical inputs into signal data (e.g., sensor measurements). The segmentation phase isolates objects from the signal data (e.g., time intervals from continuous context data). The feature extraction phase calculates object properties (e.g., symbolic context features or atoms) that are useful for classification. The classification phase uses features to assign the object to a category (context). Post-processing concerns decision (action) making (e.g., application control).

In this dissertation context abstracting refers to the pattern recognition process up to and including either the feature extraction or classification phase. Context recognition refers to the pattern recognition process up to and including the classification phase. Hence context recognition is also context abstracting.

The aim in the context abstracting process is to represent sensor data with instances of context ontology, and provide them as events for an application through the framework. As in pattern recognition, the task of context recognition is to classify a set of objects (measurements or other signals) into a set of classes (context descriptions, as defined in the ontology). Usually, the functions that are used for classification are learned from the data resulting from the usage of a device or an application. Hence context recognition can also be seen as a machine learning problem (Mitchell 1997). A related research direction is data mining and knowledge discovery (Fayyad et al. 1996), which addresses the problem of finding from data the relevant explaining functions, which are not necessarily known in advance. This dissertation does not address context data mining. In machine learning terms, data mining focuses on unsupervised learning, whereas the case study in this dissertation is a case of supervised learning. In supervised learning a teacher provides a category label for each pattern in a training set, and the goal is to be able to classify each pattern into the given correct category. Since the correct categories are known, quantitative evaluation is possible. In unsupervised learning, or clustering, there is no explicit teacher, and the system forms clusters of the input patterns according to the parameters given to the clustering algorithm.

2.4.2 Related sensor-based context abstracting studies

This section addresses studies that aim at abstracting or recognising contexts like the user activity and state of the environment. One approach to context recognition is to add a set of sensors into a mobile device the user is carrying, or into several parts of the body. The other approach is to have the sensors in the environment. This dissertation focuses on the former approach, and on mobile context recognition in particular.

Schmidt et al. (1999a) experiment with recognising environment and usage contexts based on sensors embedded in a mobile device. The authors introduce a layered architecture for context recognition from sensors. In the architecture, the bottom layer consists of the sensors. The cues provide an abstraction of the sensors. Cues are similar to features in pattern recognition terms (context atoms). Context is recognised from the cues by using simple recognition rules in online recognition. Kohonen maps (Kohonen 2001) were experimented with in exploratory offline clustering of cues, where clusters were found to represent certain contexts. Context classification was not performed and the analysis was qualitative.

Schmidt (2002) discusses sensor-based context acquisition and use of context information in several applications related to aware artefacts and sensing environments. Concerning context information processing, the layered architecture (Schmidt et al. 1999a) is presented as a conceptual model for sensor data processing (Figure 6). Concerning information processing, Schmidt (2002) focuses on the sensing and feature extraction phases of pattern recognition process (Figure 5). The recognition experiments and results are not discussed in detail; the main focus is on prototyping multiple context-aware applications and discussing experiences from the prototypes.

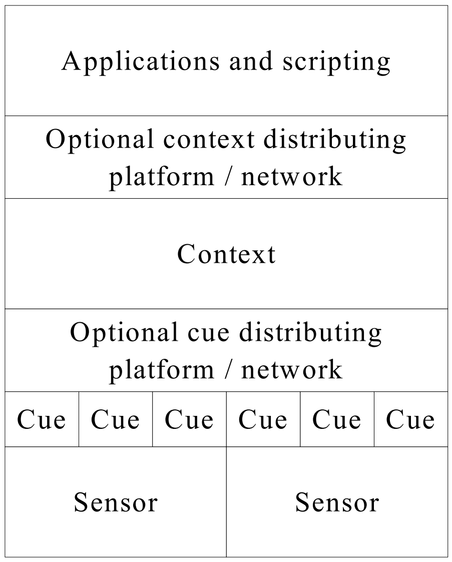

Applications and scripting
Optional context distributing platform / network
Context
Optional cue distributing platform / network
Cue
Cue
Cue
Cue
Cue
Cue
Sensor
Sensor

Figure 6. Layered perception architecture for processing sensor information.

Clarkson and Pentland (1998) apply Hidden Markov Models (HMM) to differentiate noise from speech by and around a wearable computer user. Clarkson et al. (2000) describe experiments in recognizing the user situation using a wearable camera and a microphone. HMMs are used to detect coarse locations (such as at work or in a subway) and coarse events (such as conversation or traffic). Camera-based systems are beyond the scope of this dissertation.

Laerhoven and Cakmakci (2000) report on using a layered structure consisting of feature extraction (cues), Kohonen maps, K Nearest Neighbor classification (KNN), and first order Markov chains to detect user activities such as walking, standing and bicycling (Figure 7). The sensor set includes a two-axis accelerometer, passive infrared, carbon monoxide, microphones, pressure, temperature, touch, and light sensors. Recognising multiple simultaneous contexts is not discussed. The results only show true positives as percentages.

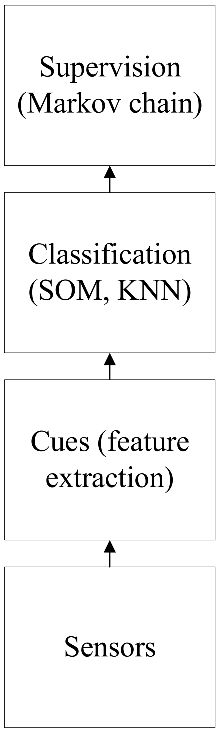

Supervision (Markov chain)
Classification (SOM, KNN)
Cues (feature extraction)
Sensors

Figure 7. The layered context abstracting system of Laerhoven and Cakmacki.

Laerhoven and Aidoo (2001) further discuss the layered approach. Their goal is to make the system learn the context descriptions from its user while the user is performing the actions, with as little user interaction as possible. As a form of supervised learning, the concept requires the user to label the clusters that are formed during the action on the Kohonen map for later use. Moreover, since there are many inputs for the learning, the user may end up training those inputs to the cluster, which are not relevant for the actual context being trained.

Castro and Muntz (2000) experiment with an indoor location system, where the signal-to-noise ratio (SNR) information of WLAN base stations is used to locate a person. They use a hierarchical Bayesian network model that consists of a query variable (location) as a root node, SNRs from multiple base stations as leaf nodes, and an intermediate layer of nodes. Each SNR has a certain probability distribution for classifying locations correctly. Entropy is used to determine from the distribution the uncertainty of each base station SNR as a classifier input, and those that have the highest entropy are chosen. The aim is to minimise the number of sensors while maintaining a good accuracy. The approach is smart space oriented.

Peltonen et al. (2002) classify from the audio data 17 everyday environment situations, such as streets and restaurants, achieving an accuracy of 63.4% of true positives with an average of 30-second analysis duration. They choose and compare the two best performing feature sets out of the 11 possibles as an input for 1-nearest neighbor and Gaussian mixture model classifiers. The study focuses on audio. Multi-sensor multi-action context recognition is not discussed.

Mäntyjärvi (2003) focuses on applying statistical and machine learning methods for explorative data analysis of context data. In other words, the main contribution of the author is context data mining for discovering data patterns that correspond to real world situations for the purpose of searching for a suitable representation for context. Furthermore, as the partial contributions of the dissertation, Mäntyjärvi et al. (2001) and Himberg et al. (2001) perform data mining to discover mobile device user context from multidimensional sensor data (3 axis acceleration, light, temperature, humidity, skin conductivity). Minimum variance segmentation, k-means clustering, PCA and ICA are used as ways of analyzing the data, which reveal patterns referring to coarse usage situations. These studies first applied the early version of the context representation later published by Korpipää and Mäntyjärvi (2003). Moreover, the natural next step to the explorative data analysis approach, where the results are analysed qualitatively, is the traditional supervised learning label-train-classify approach used in this dissertation to quantitatively evaluate how well certain contexts can be recognised from a sensor data set.

Bao and Intille (2004) use five 2D accelerometer dataloggers attached at five locations in the human body (dominant arm wrist, dominant leg ankle, thigh, arm and hip) for recognising 20 different indoor activities (pre-segmented, one activity at a time) annotated by the 20 test subjects themselves. Mean energy, frequency domain entropy and correlation of acceleration data (between 2 axes of each board, and between all pairwise combinations of axes on different boards) are used as features. Four classifiers were tested, and the decision tree classifier C4.5 performed best, having 84% average accuracy. The application area of the study is wearable computing, whereas this dissertation has a mobile device-centric view. Moreover, Korpipää et al. (2003a) present multi-sensor recognition from a continuous (not pre-segmented) data stream, where multiple contexts appear at the same time instant.

Lukowicz et al. (2004) study recognizing wood shop activity by using microphones (2) and accelerometers (3) worn on the body. Nine different activities, such as sawing, drilling, etc., are recognised one at a time from continuous data, by first segmenting it with sound intensity analysis and then partitioning with linear discriminant analysis (LDA) classification with majority decision for several second windows, before the final classification of the identified segments. The identified partitions are not perfectly aligned to the true activities. LDA and hidden Markov Models (HMM) are used for the classification, where the final accuracy is calculated as the result of both classifiers having the same output for the segment. One classification result is given for each identified segment, and in one dataset there were 25–30 coarse partitions of different lengths. An average of 84% accuracy was reached. Korpipää et al. (2003a) classify multiple simultaneous contexts with one-second resolution – i.e., the classification result is given for each second in the continuous data, instead of partitioning the continuous data into segments of different lengths before the classification. In mobile device context-aware applications, such as performing an action in the device user interface based on context, it is necessary that the classification resolution is not too coarse.

Furthermore, recognising human emotions with sensors has been studied by Picard (1998). The human affective state is potentially a very relevant part of the overall context of a mobile device user. The problems in practical measurement and detection of emotion are, however, very challenging. Emotions are often subjective, overlapping, ambiguous and difficult to define. Measuring them often requires sensors with skin contact in various parts of the body. However, in laboratory conditions some success has been reported in detecting basic emotions, such as anger, sadness, joy and fear.

2.4.3 Methods for context-based inference

The aim of this section is to outline some major distinctions of a set of commonly used machine learning and inference methods by reviewing the representation and inference properties of each method. The relevant properties include efficiency, uncertainty handling, capability of handling multidimensional input data, flexibility of updating a model, and scalability. The review is used to introduce and compare a few methods with potential for application to context-aware mobile computing. However, since the scope of this topic is extensive and the number of potential methods is large, the review has been compacted and serves mainly as introductory material without deeper analysis.

As an example, a comparison of methods in AI is given by Minsky (2000) (Table 2). Part of the methods selected for introduction can be positioned in the Table 2; however, the argumentation for this positioning is left outside this dissertation. In Table 2, the horizontal axis represents numbers of causes, which increase in table cells from left to right, and the vertical axis represents scale of effect, which increases from up to down.

Table 2. Minsky (2000) compares AI methods in a causal-diversity matrix.

Number of causes →

Scale of effect ↓

Easy	Linear, Statistical	Connectionist, Neural network, Fuzzy logic
Ordinary qualitative reasoning	Classical AI	Analogy-based reasoning
Symbolic logic reasoning	Case-based reasoning	Intractable

In addition to the representation and inference properties, the third important characterising factor is learning. The methods under discussion have very different needs for training data, and the complexity of the corresponding learning algorithms vary radically. However, the supervised approach for learning adopted in this dissertation assumes that the model describing the transformation of data from measurements to contexts is trained offline, while the actual inference needs to be performed online. Hence the learning efficiency is not emphasised.

Bayesian networks

Bayesian modelling is named after the amateur mathematician Thomas Bayes (1702–1761). Bayesian network is a symbolic knowledge representation method, where nodes of the network describe events and transitions between events are conditional probabilities. The inference in Bayesian networks is based on probabilistic reasoning – more precisely, on the Bayes theorem. In a network where many nodes are dependent on the others, the inference becomes computationally heavy, even with relatively small amount of nodes. Updating the model (adding a node for example) requires updating all the dependent conditional probabilities. Bayesian modelling supports representing uncertainty, and including background knowledge in the model is intuitive. (Russel & Norvig 1995; Myllymäki & Tirri 1998; Mitchell 1997; Pearl 1988; Duda et al. 2001)

Naïve Bayes

The Naïve Bayes model is also based on the Bayes theorem, but in addition the model assumes that all the events in the model are independent of each other, leading to a tree structure instead of a network. Therefore, the inference in the Naïve Bayes model is very fast and straightforward, which enables the ability to handle more inputs than the Bayes model. Training a Naïve Bayes classifier has a computational complexity *O(en),* where e is the number of training examples and n is the number of features for each example. The Naïve Bayes inference has the computational complexity *O(cn),* where *c* is the amount of different classes. The strengths other than efficiency are the same as with Bayes networks, but updating the model is easier since, unlike the Bayes model, there are no dependent conditional probabilities in the network that have to be updated once a new node is added. In spite of the basic assumption of node independence, the method has been successfully applied, and for problems such as text classification it is among the most efficient methods known. (Myllymäki & Tirri 1998; Mitchell 1997, Pearl 1988, Duda et al. 2001)

Case-based reasoning

In case-based reasoning (CBR), knowledge is contained as examples, which can, for instance, be presented as symbolic expressions. In the simplest form of instance-based reasoning the examples are points in the n-dimensional Euclidean space, and classification of a query instance, i.e. inference, is conducted by calculating the Euclidean distances between the query instance and case examples. The class of the query instance is then the class of K Nearest Neighbours of the query instance (KNN). In CBR, however, the case descriptions can be more complex and inference may rely on search or knowledge-based reasoning. A model in CBR is flexible to update; adding a new case means adding an example. The amount of possible outputs can be large, since an example basically corresponds to an output. The efficiency depends on the chosen representation and inference framework. With complex case representation, finding a proper distance metric can be difficult. (Mitchell 1997, Aamodt & Plaza 1994)

Logic

In logic, information is encoded as symbolic rules and propositions. First order logic can describe objects, relations, functions and properties. Inference in logic is conducted according to certain inference rules, which take the sentences in the knowledge base as an input and produce new sentences as an output, which are added into the knowledge base. Logic syntax expresses meaning explicitly for humans, background knowledge can be described straightforwardly, and the knowledge model can be modified and expanded flexibly by adding new rules into the knowledge base. (Russel & Norvig 1995; Hirsh & Hearst 2000; Cresswell 1973)

Fuzzy logic and fuzzy sets

In fuzzy logic the knowledge is encoded into symbolic rules, but the variables involved are multiple valued instead of Boolean as in Logic, and the values are described by fuzzy sets. Instead of crisp true-false values, fuzzy variables can be true with a certain degree as defined in a fuzzy set. In other words, fuzzy sets and fuzzy logic support the handling of partial truth. (Zadeh 1965, 1996; Cox et al. 1998)

Neural networks

In neural networks, the knowledge is encoded into the weights between the nodes of the network, where each node may have many inputs. The structure of the network has no direct conceptual meaning for a human – i.e., a neural network has a non-symbolic knowledge representation. In inference, each node sums its real-valued inputs and, using a triggering function, produces one real-valued output, which may become a weighted input for many other nodes. Neural network inference is very simple, and very fast. Other positive features of neural networks include genericity in modelling functions, uncertainty handling, and a capability of massive parallelism – that is, the capability of simultaneously handling a mass of inputs effectively. The main weaknesses are that the network is a black box, hiding the explanation to a decision, and the background knowledge inclusion is difficult. (Russell & Norvig 1995; Mitchell 1997; Pyle 1999)

Hidden Markov Models

Hidden Markov models (HMM) represent the information as a stochastic state machine, where the nodes correspond to the states of the system and the transitions between the states indicate the probabilities of a state change. Inference in HMMs corresponds to calculating the probability of an observed sequence of states for each state model, and choosing the one with the best likelihood. The inference, when the Forward algorithm is applied, has a computational complexity of $O(c^2T)$, where c is the number of states and T is the number of observations. The strengths of HMM include uncertainty handling, the capability of modelling sequences of events, and flexibility. With the probabilistic framework, HMMs can handle noise in sensor data and imperfect training data. A HMM modelling a pattern, such as a context, is not dependent on other HMMs. This enables flexibility, and the HMMs in a set, each representing a class, can be deleted, added and modified without affecting the other HMMs. In case of multidimensional input data, the dimensionality has to be reduced to one before applying HMMs. HMMs have been successfully applied in speech, character, and gesture recognition. (Rabiner & Juang 1993, Duda et al. 2001)

2.4.4 Conclusions

The state of the art in context abstracting and recognition was reviewed. The related work can be roughly categorized into early experiments that exploratively examine the behaviour of the data, in other words data mining, and the more recent pattern recognition-oriented experiments, where the data is classified into a set of previously defined classes. Data mining-oriented studies produce qualitative evaluation results, while the pattern recognition-oriented studies are able to address the recognition accuracy quantitatively. This dissertation contributes to the latter category. The application domains of the related work can roughly be categorized into recognition in a specific environment, most commonly home, and to studies concerning the mobile user. The related work addressing the focus domain of this dissertation is mostly data mining-oriented and contributes qualitative results. This related work of explorative data analysis has contributed as a natural predecessor to the studies of supervised learning and quantitative evaluation. The related work mainly reports offline experiments as opposed to taking a step towards a functioning online system in a mobile device.

Due to these issues, the related work lacks several topics of interest to this dissertation, such as the recognition of multiple simultaneous contexts from multiple sensor sources, transformation of continuous sensor data flow into context change events within a context framework, evaluation of the feasibility of continuous multi-action context recognition quantitatively, and applying classification from sensor data involving uncertainty, within a context framework, for real mobile device applications. Moreover, the related work does not provide implementation and evaluation of context abstracting and classification within a context framework with a set of real applications implemented in a real handheld mobile device.

2.5 Summary

The literature review of technologies for mobile context awareness examined the state of the art in context frameworks, context representation and ontologies, context abstracting and recognition, and customization. In general, the existing literature does not give a unified and solid view of the reviewed topics from the mobile device-centric viewpoint.

Concerning context frameworks, the related work primarily discusses prototypes and experiments performed with PCs or laptop PCs connected in distributed environments. The viewpoint of the related work is mostly environment-centric, i.e. the frameworks are designed so that context information is processed in the environment instead of the terminal. The related work lacks several topics concerning a software framework designed for processing sensor-based data in a mobile device itself. However, despite the different focus the related work discusses many relevant requirements for context architectures, and provides a useful comparison of architecture models, suggesting the blackboard model as a superior model for context awareness.

Concerning context representation and ontologies, the related work contains semi-informal and formal approaches to modelling domains of context information. The most common domains for context modelling were smart environments and smart objects. Both of these domains concentrate on modelling the environment, some specific environment, or specific objects in it. Terminal-oriented approaches describe static contexts such as device properties and context taxonomies for categorising the different context types that are considered relevant. This related work has contributed to this dissertation with the basic idea of categorising context types. Due to the difference in the addressed principles, the related work lacks several specific topics of interest to this dissertation, especially concerning the context data structure that must be common across different application domains.

Concerning context abstracting and recognition, the related work can be roughly categorized into data mining and pattern recognition-oriented experiments. The application domains of the related work can roughly be categorized into recognition in a specific environment, most commonly home, and to studies concerning a mobile user. The related work addressing the mobile user application domain is mostly data mining-oriented. This related work of explorative data analysis has contributed as a natural predecessor to the studies of supervised learning and quantitative evaluation.

Another clear deficiency in the related work is the lack of implementation and evaluation of the issues pointed out in the review with a set of real applications in a handheld mobile device such as a mobile phone. Such an implementation and evaluation is significant since it is the only way to confirm that the proposed solutions really work and actually can be applied in a handheld mobile device. Such confirmation cannot be reached with prototypes and simulations in a PC environment, where the constraints and requirements are completely different.

Software architecture models, knowledge representation models, and machine learning methods were analysed for potential utilisation in mobile context-aware computing. The existing art and the development targets identified in the literature review form the basis for analysing the requirements for the development of context framework, representation and ontology, and abstracting and recognition, with the aim of solving the underlying research problems defined in the introduction.

3. Context framework requirements analysis

This chapter analyses the requirements for a software framework for enabling mobile device context awareness. Requirements for context representation and context abstracting and recognition methods will be separately addressed in sections 5.1 and 6.1 respectively.

3.1 Characteristics of mobile computing

The related work was found deficient in several aspects of developing context awareness for mobile devices, where the computing is performed in the mobile device itself instead of the environment infrastructure. The challenges differ significantly from stationary PC computing and infrastructure-centric computing. Mobile device-centric context-aware computing has more restrictions and additional characteristics and requirements. The restrictions of mobile computing, relevant for developing mobile device context-aware applications, compared with stationary and infrastructure-centric computing include the following issues:

- Less computing power
- Less memory capacity
- Restricted network access capability: lower bandwidth, higher price of data transfer, varying availability of network, and varying network transfer rates
- Smaller screen size
- Restricted input capabilities
- Limited battery capacity.

The additional characteristics and requirements of mobile computing, relevant for developing mobile device context-aware applications, compared with stationary and infrastructure centric computing include the following issues:

- Unrestricted mobility; beyond the location(s) having a local computing infrastructure

- Unlimited number of different usage situations due to the unrestricted mobility
- Fast changing usage situations, requiring fast response of the system to the user
- The mobile device is required to keep track of the context, instead of the environment keeping track of the device, due to the unrestricted mobility, fast changing usage situations and the network access capability restrictions
- The mobile device has to function as a standalone device anywhere – i.e., retain as much functionality as possible even when not connected to the network.

3.2 Arguments for device centralized context management

There are practical arguments for mobile device-centralised context management. As discussed previously, distributed computing with mobile devices has several deficiencies, such as low bandwidth and the cost of data transfer. Rapidly changing usage situations, measured with device sensors, require a quick response from the system, the faster the response the better user experience. When context data is measured by the terminal, and the data is used for the purposes of immediate interaction, it is not feasible to continuously send high sampling rate data to a distributed server over the air for analysis and then back to the terminal for making the interaction action. Including a distributed server in this kind of tight interaction control loop having strict response time requirements is not sensible or feasible; it would be unnecessary, slow, frustrating to users, battery consuming, and expensive, to mention a few arguments. As a simple example, the device could have accelerometers, which are used to measure whether the device is still or not, and the result is used to switch on the screen backlight. Therefore, sensor-based context management must be performed in the mobile terminal.

Practical experience from the process industry has shown that when there are multiple heterogeneous information producers that abstract data from a process, they should provide the produced information in a uniform structure to a central node to facilitate uniform processing of the abstracted information at the consumer side (Kurki et al. 1998, Korpipää 2001). In the mobile device industry, the process is the usage of a mobile device.

3.3 Arguments for selecting the blackboard model

As the blackboard model is central to this dissertation, some further discussion on the arguments for selection is in order. The literature review Winograd (2001) already gave well-established arguments for the blackboard model over the two compared models – i.e., the widget model and the client-server model – emphasising simplicity and configurability. Figure 8 illustrates the difference of the client-server and widget models, Figure 8 a, compared with the blackboard model, Figure 8 b.

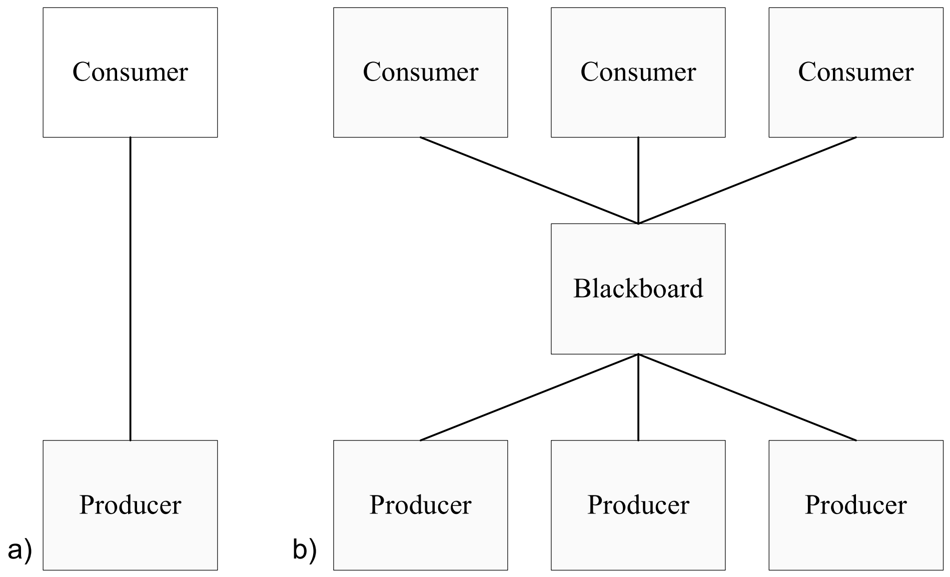


Figure 8. A conceptual comparison of the blackboard model with the client-server and widget models.

In the widget model and the client server model the information producer and consumer have direct communication, where either the consumer has to address each producer separately directly or vice-versa. In the blackboard model the information producers do not have to know about the consumers, and the consumers do not have to know about the producers. Data has a uniform structure for all the components. The blackboard model hence offers flexibility by providing any consumer information from any producer without separate search and connection establishment.

Having a uniform structure for all data, and a common centralised memory space, the blackboard model offers simplicity and configurability advantages for utilising inference engines that operate on the events from the data objects added into the memory space. The inference engine can operate as both consumer and producer. The consumers see the inference engine operation as transparent, whereas in the widget and client-server solutions the consumer would have to find each inference engine and separately connect to it.

Based on the practical requirements, experience and the literature, the blackboard architecture is considered to be the most suitable context management model, and is thus used in this dissertation. A blackboard-based context framework suitable for mobile device context management will be built.

3.4 Conceptual entities of the framework

In the requirements analysis, to further discuss the feasibility of a blackboard-based architecture model, Context Toolkit (Dey 2000), as a widely cited context architecture representing the widget model, is used for comparison. Another framework for comparison is the Web services architecture for distributed context management, representing the client-server architecture model (Mitchell 2002). Both frameworks have been published in academic dissertations, and include a requirements analysis.

The requirement specification was initiated by analysing use cases (UML 2005) for the application's use of context information. To maintain a proper focus, the use cases were defined to model how an application should use context information in a mobile terminal in general. In other words, the main actor was the application, not the user. Deriving the requirements was hence quite straightforward, and the use cases are not presented here in order to avoid too much repetition since the requirements contain much of the same information. Based on the use cases and the literature, conceptual entities of the framework were identified. These main entities are briefly introduced prior to the requirements.

- Context Manager is the central entity that contains the blackboard, and communicates context information with other entities.

- Client is used to refer to any entity that uses Context Manager. Applications, Context Sources, and Context Abstractors are all clients.
- Applications are entities that utilise context, but they may also produce contexts.
- Context Sources are entities that receive context information from the local and global infrastructure (abstract it) and write it into the blackboard.
- Context Abstractors transfrom and abstract raw data or contexts into other contexts.
- Change Detector determines when a context has changed.
- Application Controller controls applications based on context information.
- Context feature, context object, context instance and context atom refer to a single basic unit of context, independent of the abstraction level.

3.5 Overview of the requirements

This chapter gives an overview of the most relevant context architecture requirements in the literature, and in this dissertation. Dey (2000) defines a set of requirements for an architecture supporting the development of context-aware applications. The authors aim to provide a conceptual framework that supports all the tasks that are common across applications, leaving only application-specific tasks for the designer. The set of requirements by Dey (2000) is presented in Table 3.

Table 3. Context architecture requirements by Dey (2000).

	Requirement
1.	Context specification
2.	Separation of concerns
3.	Context interpretation
4.	Transparent, distributed communications
5.	Constant availability of context acquisition
6.	Context storage and history
7.	Resource discovery

Mitchell (2002) presents a set of requirements that aim at providing a Web services architecture for context management (Table 4). The approach is environment infrastructure-oriented, and, as such not, within the scope of this dissertation. However, the requirements contain common characteristics. The following set of requirements is identified.

Table 4. Context architecture requirements by Mitchell (2002).

	Requirement
1.	Supporting user and device mobility
2.	Support persistence of application and user state
3.	Support flexible interaction models
4.	Security and privacy of user data
5.	Extensibility
6.	Modelling the environment
7.	Management of shared and distributed data
8.	Configuration and interoperability
9.	Context capture
10.	Context interpretation
11.	Infrastructure transparency
12.	Context presentation, adaptation and persistence
13.	Ability to support awareness
14.	Ability to support context sharing
15.	Specification and representation of context

In this dissertation the requirement set is extended and deepened. Even though the requirements are partially overlapping with the related work (Dey 2000, Mitchell 2002), the focus is on mobile device-centric context awareness requirements. An overview of the requirements in this chapter is presented in Table 5.

Table 5. Requirements for mobile device context framework.

	Requirement
1.	Concurrent context management in mobile device
2.	Requirements for the application programming interface
3.	Flexibility in handling new contexts
4.	Context abstracting and recognition (detailed requirements in section 6.1)
5.	Event-based communication of context to application
6.	Context database
7.	Context caching
8.	Time resolution of context
9.	Change detection
10.	Context confidence
11.	Context representation (detailed requirements in section 5.1)
12.	Application control
13.	Customization

Each of these requirements is discussed separately in the following chapter from the mobile device-centric viewpoint. The requirements analysis aims at providing a basis for the design of the context framework for a mobile device. Some parts of the requirements analysis reach a level of detail level where the design solution is apparent or already partly determined. This is done on purpose and is a useful and widely applied engineering practice in order to enable a straightforward design phase.

3.6 The requirements

3.6.1 Concurrent context management in a mobile device

The main function of the context management system is to facilitate the use of context for applications that are located in the mobile handheld device. The context framework must be able to handle information acquired from the device's internal and external sources. External context information may come from both the local and global (Internet) infrastructure. The device's internal context includes information received from the sources that are in the device, such as sensors. Handling multiple sources concurrently is required. Concerning internal sensor sources, it is most efficient to process the information in the device itself. Hence Context Manager, and the blackboard, must reside in the handheld device carried by the user.

Concurrent processing of acquiring, abstracting, storing and delivering context from multiple sources is required. The context management system must be able to handle multiple contexts that appear at the same time. Concurrent use of multiple contexts by multiple applications is required.

The first requirement of Mitchell (2002) emphasises that the application must withstand periods of disconnection from networked context source. When the blackboard manager is in the device, disconnection does not prevent the functioning of context exploiting the application. Disconnection is seen by the application as having no changes in context from networked sources.

3.6.2 Requirements for the application programming interface

Context Manager is required to provide a set of services that can be used by any client through an API. Any client is allowed to add context into the blackboard, and any client is allowed to use it. The clients must be able to subscribe to be informed about changes in context. In other words, a 'publish and subscribe' mechanism is required. When a context event occurs, and there is a change in context, the client is informed, but otherwise no data is sent to the client. This is a major advantage over using a direct flow of raw data from the source (e.g., sensor) in the application. The application can process other tasks while no important changes occur in context. Message traffic up to the application decreases radically. In other words, the context framework should follow the Hollywood Principle: "Don't call us, we'll call you" (Larman 2002).

There are three types of basic subscriptions an application may require to use: Context change, Context start and Context end. Context change informs the client every time the context value of the subscribed context types change. In the Context start subscription, the client is informed of the start of a specific context value. For example, an application may be interested in the context value of being in a movie theatre, a location-type context. The application wishes to turn the device sound off when the user arrives in the movie theatre, and back on when the user leaves the theatre. If the application could only subscribe to be informed of changes in location-type contexts, it would receive a message every time a location context changes, thus generating needless message traffic. By subscribing to the start and end of a certain specified location context value, the application only gets exactly those messages it is interested in. Similarly, in Context end subscription, the client is informed of the end of a specific context. The client must be able to unsubscribe all the subscriptions it owns.

Another required way of using context information is directly requesting it from the Context Manager, which should be similar to making queries from a database. There are three basic types of queries that the client may require to use: Context set request, Latest contexts request, and Time interval request. Context is returned based on context type, source, or both. Context set request returns contexts of a given set of context types or sources. Latest contexts request returns a given amount of latest contexts, and for Time interval request contexts of a given time interval are returned.

3.6.3 Flexibility in handling new contexts

Adding new contexts and new elements that produce or process context should not require making changes to the Context Manager. Hence the Context Sources and Context Abstractors should be plug-ins. New contexts must be handled as data objects instead of widgets, as proposed by Winograd (2001). The context framework must be able to handle new context types and values without changes to the framework entities. For example, information from passive sources, such as tags that only broadcast, can be straightforwardly utilised in a blackboard-based architecture. Context information from such sources can be utilised without implementing a new widget for each tag, as required by the widget model.

Hence context should not always be bound to the source of the context, as the widget model proposes. It is possible that the source of a certain context will change while the user is on the move. The application does not necessarily need to know about the change of the source. The blackboard model is a feasible solution for providing this kind of transparency; the application always gets the context from the same blackboard, regardless of the source, and the sources have one place where context is written.

3.6.4 Context abstracting and recognition

The information acquired from sources such as sensors may require abstracting to be usable for applications. Abstracted contexts can be application-specific or applicable to many applications. In case the abstracted context can be utilised by many applications, Context Abstractor or Context Source should add the abstracted contexts to the blackboard for use by other applications.

As was defined earlier and illustrated in Figure 5, context abstracting refers to the pattern recognition process up to and including either feature extraction or the classification phase. Context recognition refers to the pattern recognition process up to and including the classification phase. Context Recogniser here refers to an entity that performs context recognition. Context Abstractor refers to an entity that performs context abstracting.

Earlier, Dey (2000) and Mitchell (2002) specified one requirement related to context recognition – context interpretation – and both architectures support interpretation at the concept level. However, a detailed analysis of the process and suitable methods for abstracting data, particularly from multiple sensors in a mobile handheld device, is required to facilitate systematic use and reuse of the device's local sensor resources.

It should be possible to add, modify and remove Context Abstractors (and Context Sources) from the framework without disturbing the system. In other words, it should be possible to plug in producers of context. The loose coupling of framework elements will ease the configurability of the system (Winograd 2001).

It should be possible to recognize context from multiple sources and time sequences. The framework should support sensor fusion. Recognition of higher level context from existing ones may be performed from the two basic cases of input:

1. Context recognition from a set of contexts, where the set can be the size of one or larger.

2. Context recognition from context history. Recognition may be based on either a specified number of latest contexts or a time interval.

The result of a previous recognition can be used as a further input for another recogniser. From the client viewpoint, the use of abstracted higher level contexts should be transparent. Client may subscribe to a higher level context as to any context. Every time a context in the recognizer subscribed input set changes, recognition is performed and new context is added to the blackboard, and client is informed of the possible change.

The abstraction level of the data received by the context manager may vary. Three cases can be identified:

1. Context Manager receives from the source event-based abstracted contexts that do not require further abstracting to be used by the application.

2. Context Manager receives from the source event-based abstracted information that requires further abstracting. In this case context recognisers can be used.

3. Context Manager receives raw measurement data that is updated continuously and possibly with a high frequency.

In the third case the frequency of the incoming data is decisive on how to handle the data. If the frequency is low, the source may add the data directly to the blackboard, and the abstractors receive the data by subscribing to it and perform further abstracting. If the frequency of continuous input is high, the source itself should abstract the incoming data before adding it to the blackboard. If possible, the source should also perform change detection, so that the incoming high-frequency data would be converted to event-based data.

3.6.5 Event-based communication of context to application

The primary advantages of the blackboard model are simplicity, sharing, light configuring and robustness. The blackboard-based context framework should primarily be used for delivering data for applications as events. In an ideal situation the advantage gained from this approach is efficiency the data is provided for the application in a directly usable abstracted form and only when it is needed. Another advantage is that those computing elements that refine the data are separate from the application, which ideally only needs to use data instead of first processing it into a usable form. If the incoming data from the sources is continuous, the framework should abstract it so that the data provided for the application with the subscription-indication mechanism would be event-based.

In continuous communication the model is not the most efficient one due to the central node, the blackboard, which does not exist in a point-to-point communication. Continuous communication is feasible with low frequencies. High frequencies of continuous communication, required by the application without abstracting, should be handled with a direct connection from the sensor to the application. The aim is to deliver events to the application, even though the source would measure or receive a continuous stream of data. There are two basic ways of dealing with continuous incoming data within the framework:

1. Context Sources that receive information from external sources simply forward it to the blackboard. Context abstracting and change detection will be performed after Context Manager. This approach is feasible if the frequency of incoming data is low. The implementation of Context Sources can be very simple in this case.

2. Context Source itself performs abstracting and change detection before adding context to the blackboard. This approach is preferred if the frequency of incoming data is high.

3.6.6 Context database

Context storage, or a database, is required to store context history. Time sequences of context can be directly utilised by clients to recognise the current context, establish trends, or predict a future context. Dey (2000) also sets the requirement of context storage, but the choice of model generates several difficulties, which were discussed in Chapter 2.2.4. This dissertation contributes a more detailed analysis and design of the required capabilities for a mobile device local context database.

In order to maintain the availability of device applications to context history in the event of a reboot, a permanent storage for context information is needed. The most feasible solution for the storage is a terminal database, since modern mobile devices contain sufficient capacity for permanent storage. If the database should exist in the network, disconnections would cause several problems with the unavailability of accessing and storing context, the delays in storing terminal-originated contexts would be longer, and bandwidth would be needlessly consumed in cases when only terminal contexts are used.

The memory size of the context table residing in the device should be configurable; the history length for each context type stored in the database should be configurable, to allow longer histories for more important contexts and short history for more temporary contexts; and the amount of context types in the database should be configurable.

Not all context data needs to be stored in a permanent database, even though it should be possible for applications such as location tracking. Sensor-based applications, such as those applying movement sensors, utilise short-term active-type of data, which does not require permanent storage. Furthermore, the performance requirements for managing sensor data are demanding, and if a relational database is used, inserts and compacts are not very efficient operations. Hence the use of the permanent storage database provided by the Context Manager should be optional. This option should improve the performance, especially with high sampling rate input data, such as accelerometer data. For short life span data, the Context Manager should contain a fast cache memory, which has the history length of one for every context type.

3.6.7 Context caching

Clients are not allowed to delete contexts from the context table. The most important reasons for this policy are the following:

- Other clients may possibly need the context that would be deleted, or the history of the context.

- Since context can be any information from any source, the owner of the context is not always known. The owner is thus not even a specified property of context, and deleting cannot be based on ownership.

Instead of individual clients performing deleting, it is handled centrally. The approach is similar to memory cache solutions. When the context table is full of context types (the cache is full), a number of types (including history) will be deleted when new ones appear. There should also be a memory limit for the history length for each context type. The following options can be used as criteria for deleting contexts:

- When the limit for the number of different context types in the context table is exceeded, the least frequently used context types are deleted.
- When the limit for context types is exceeded, those contexts that have not been used in a certain time frame are deleted.
- When the limit for context types is exceeded, those contexts that have no subscribers are deleted. If all have subscribers, use other criteria.

If the criteria for deleting a context and its history are not matched, context never expires globally. It is a task of the client to determine when a context has expired from its own application-specific viewpoint. Even though a client decides that a certain context is not valid for it anymore, it cannot delete it from the context table. The same context may still be valid for other clients. Context histories are not deleted by clients either.

Symbolic context names may require renaming. Renaming is not performed centrally. It is seen as a type of context abstracting, where input context is given a new name and put back to the blackboard. For Context Manager, renamed context is a new context.

3.6.8 Time resolution of context

Since every (context) message between the source and the client goes through the blackboard, it takes at minimum two hops to deliver the message. In addition, subscriptions require checking for every received message. Therefore, communication in the blackboard model is less efficient than in point-to-point communication, and high message frequency may cause difficulties in the case of continuous communication.

Depending on the size of the blackboard and available resources, the maximum communication frequency for continuous communication through the blackboard should be limited. Applications that require high frequency of continuous communication should not use the blackboard. Moreover, for sources that produce continuous data, the history limit should generally be configured very short to avoid quickly consuming a lot of memory.

3.6.9 Change detection

At each moment in time the context blackboard contains a set of context types that have at least one value in the context history. When a new context value is received, a matching operation must be performed to see if it has changed from the previous one. If there has been a change, the subscribers to that context are informed of the change.

The method for detecting a change between two contexts depends on the representation of the context. In the most simple case, the context is represented by a string of characters, for example, and change is detected using a simple string match operation. This should be the default case for any context, performed by the Context Manager. However, if the context is represented so that simple change detection is not possible, a separate Change Detector is required. The type of change detection method depends on the chosen context representation and the use of the context.

Change detection takes as input the current and the previous context(s), and indicates change or no change information. Even though separate change detection causes extra message traffic, it is necessary to have support for it, since it is not possible to hardcode into Context Manager all the change detection methods that may be required depending on the representation and use of context.

Context change detection should be optional. The input data for user interaction can be implicit or explicit, which has a difference from the context management viewpoint. For implicit inputs, the application is usually concerned about receiving events when a change occurs in the contextual state. For example, only changes in the user location are relevant and the application does not need to be repeatedly informed of the same location. Implicit inputs therefore require change detection. Explicit inputs must not have change detection. For example, the user may perform the same gesture one after the other, and the Context Manager is required to pass both events to the Application Controller.

3.6.10 Context confidence

Context information received by the Context Manager can be imperfect. Context information is said to be incorrect if it fails to reflect the true state of the world, inconsistent if it contains contradictory information, or incomplete if some aspects of the context are not known (Henricksen et al. 2002).

Moreover, the context information can be partially true when the boundary between two symbolic context values is not crisp. It can be true with a certain probability, based on evidence learned earlier. These characteristics reflect the uncertainty of the context information, which must be considered when defining the context representation and methods for processing it.

The data object representing context is required to have a confidence attribute. Hence instances of context information contain a property that describes the confidence of the instance. The confidence property can be used to express probability, fuzzy membership, or any other measure of uncertainty, depending on the case. The use of such confidence should be optional, since it may not always be available. The default value for confidence should be true (one).

Confidence for a context changes over time. Confidence always refers to the situation at the moment the context was stored, and as such is not always valid. Similarly, the context value itself may not be valid either, if, for example, the source that produced the context is not available for updating the value. Stored context is always a snapshot of the situation at the moment it was acquired, and the use of it must be decided by the application.

3.6.11 Context representation

Mitchell (2002) has two requirements for context representation: R6 modelling the environment, and R15 specification and representation of context, which the author evaluates as partly satisfied. R6 addresses modelling location, and R15 discusses simple operators for making decisions based on single contexts. Among other issues, the author suggests further work in context representation being to identify useful context types and data formats applicable to a wide range of context-aware applications, and to specify the granularity and the scope of events.

Adding new contexts and new elements that produce or process context should not require making changes to the Context Manager. The same applies to the syntax of the context. No changes to the Context Manager should be required, regardless of the syntax of the incoming context. However, common context properties should be defined to enable the use of contexts through an API. Context representation should facilitate the use of the context with the API. In addition to defining the common structure of the context, the representation should specify how to create vocabularies that describe useful context types with a sufficient level of detail for use.

Context provider developers, abstractor developers, and application developers must agree on a common representation for context when developing context-aware applications. The context framework must not strictly restrict the representation, but it must provide a template and instructions for producing contexts that allow the simplified use of, e.g., sensor-based data with the given API. More detailed requirements and discussion for context representation are given later.

3.6.12 Application control

The context framework is required to provide an API for programming context-aware applications. This requires creating new application code, or modifying existing applications, for defining the functions that will be performed based on the context information.

Changing existing applications to include actions based on the context is laborious and may not even be feasible. Context-based application control needs to be separated from the applications themselves. Existing applications can provide the context framework with those functions that can be used for controlling them. The context framework is required to have an entity that can be used for connecting context events to the control functions provided by the applications. In other words, an entity that enables the control of applications based on context events without modifying the applications is required.

3.6.13 Customization

The Application Controller entity connects context events, the inputs, to application control functions, the outputs. For reaching the maximal flexibility of the system, these input-output mappings must not be hard coded into the framework entities. A representation is required, which facilitates describing the connection between inputs and outputs without programming executable code. During the framework operation it should be possible to delete, change and modify the mapping between inputs and outputs.

Furthermore, the preferences of how a mobile device, e.g. a smart phone, is used for interacting with its applications and external appliances vary among users. The preferences of one user may change over time. At the design time it is thus difficult to configure the behaviour of the device so that it meets the varying user demands in varying situations and configurations, which can change over time. Therefore, for the maximal simplicity, flexibility, and potentially wide spread of developing context aware applications, the end-user should have the ability to customize the way of interacting with applications and external appliances.

When a device with a small screen is used for customizing multiple control tasks, and for the control tasks themselves, usability is of primary importance. There are several usability requirements for the Customizer tool: it should be easy to learn, effective, efficient, satisfying, and the user should feel in control of the system. These requirements are partly adopted from an ergonomic requirements standard's usability guidelines (ISO 9241-11:1998, 1998). However, extensive discussion of usability is not within the scope of this dissertation.

3.7 Summary

The literature review highlighted limitations in the existing context frameworks. Based on the literature and the use case analysis for the use of context information in mobile handheld devices, a set of requirements was identified. In particular, the viewpoint of mobile device sensor-based data processing revealed several new or updated requirements for the context framework. Hence the main contribution of this chapter is the introduction of new and updated requirements, which were defined considering the additional restrictions and additional characteristics of mobile devices.

The new and updated requirements are the following. The context framework should provide an application programming interface to the context data, where contexts are treated as data objects. The interface must remain unchanged, regardless of how and where the context is derived. Contexts should primarily be delivered for the applications as events, including sensor-based continuous data sources. The framework should support plugging in components during the system operation. The framework should support the abstracting of the context, and it must be possible to plug in abstractors. The framework should contain a central database residing in the device for easy access to the context history. The database should work as a configurable context memory. The framework should provide support for dealing with uncertain context information, and it should provide support for determining when the context has changed in order to deliver the context to applications as events. The context representation should provide a template for the context without too strict constraints, since there are many different types of context information. The representation should, however, be simple enough to allow easy usage of the context through a programming interface. The framework should separate context-based application control from the applications themselves. For maximal flexibility in developing context-aware applications, the end-user should be provided with a tool to define context-action mappings.

4. Context framework design

This chapter is partly based on the articles by Korpipää et al. (2003b, 2005b). The results of Korpipää et al. (2003b) are further developed and additional details are presented. The design is updated according to Korpipää et al. (2005b) and additional details are presented.

In the previous chapter, the requirements for the context framework were analysed. This chapter presents the architectural design of the framework. The main elements of the architecture are introduced and the flow of data between the elements is explained. Different choices for the implementation of Context Sources, Context Abstractors and Change Detectors, based on the type of incoming data, are discussed.

4.1 Overview of the design

The main entities of the framework are Context Manager, Application, Context Source, Context Abstractor, Change Detector, Application Controller and Customizer. Application Controller consists of a Script Engine (Lakkala 2003b) and an Activator. An overview of the architecture is given in Figure 9; the solid lines denote bi-directional communication, and the arrows one-directional. The dotted lines from Context Abstractor and Change Detector denote that these entities can also be implemented as scripts to be executed by the Script Engine.

Context Manager is the application-independent blackboard-based central node of the framework. Context Manager resides in the mobile terminal, and provides the same interface for all entities, including applications. The Context Manager stores context data, receives requests and subscriptions, and accordingly delivers responses and indications for the clients and Application Controller. In other words, Context Manager provides a publish and subscribe mechanism and a database. Applications can use Context Manager, or, alternatively, can be controlled by the Application Controller. Application Controller can be customized with the Customizer. Each of the other entities can be implemented as either remote or local. In this dissertation entities are implemented as mobile device local. The sources are either in the terminal device, or local sources otherwise connected to the terminal.

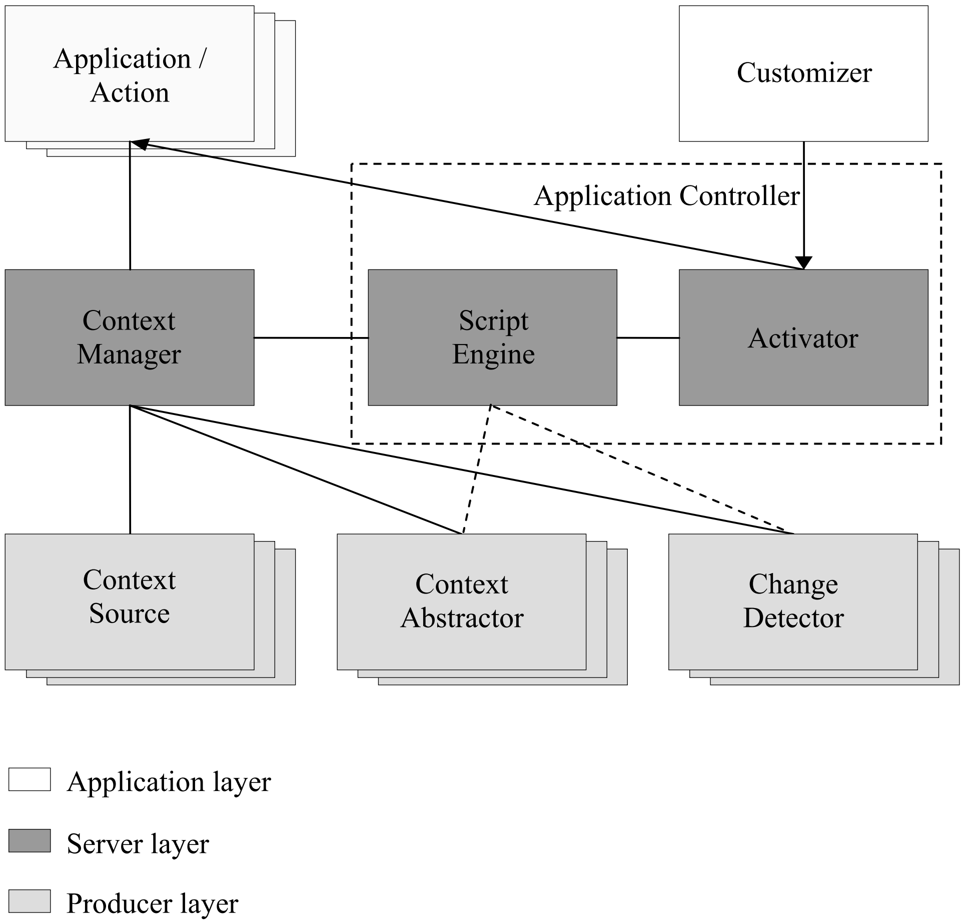


Figure 9. The overview of the context framework.

4.2 Frozen spots and hot spots

An important characteristic of a software framework is that it is reusable over different application domains. Reusability aspect of software frameworks has been addressed by Pree (1994), who introduces a notion that a software framework consists of frozen spots and hot spots. Frozen spots define the overall architecture of a software system, which remains unchanged (frozen) in any instantiation in different application domains. Hot spots represent those parts of the framework that can be specific to individual software systems.

According to the notion of Pree (1994), the context framework contains both frozen spots and hot spots. In Figure 9, the server layer components, marked with dark grey, are all frozen spots. In other words, they are application-independent elements that remain unchanged when the framework is reused across different application domains. The frozen spots are reusable without modification in any instantiation of the framework. The Activator server is also a frozen spot, but it uses a collection of function interfaces for calling code that can be application-specific. Application-specific function interfaces, which are hot spots, should ideally be plug-ins to the Activator. A Customizer is also a frozen spot, since it adapts to different domains through a text-based configuration, an ontology vocabulary, requiring no changes to the code when reused. However, Customizer is a frozen spot that can be replaced or modified, since it is an application level component.

Producer layer components are basically all hot spots i.e. new or modified components may be needed when new domains are addressed. For example, new Context Sources may need to be developed when new device sensors become available. Context Abstractors and Change Detectors are hot spots, although they are abstractions that are not mandatory in a working instantiation of the framework; their tasks can be executed alternatively by other framework elements. All hot spots in the framework can be plugged in without modifying the frozen spots.

4.3 Context Manager

Context Manager is the blackboard-based central node of the context framework in the handheld mobile terminal. This central node stores context information received from any source, and serves it to the clients. Multiple clients can produce and use the context information concurrently. The clients can directly query data by context type or source, and they can subscribe to various context change notification services. With the subscription-notification mechanism, applications can use abstracted event-based context information without needing to concentrate on the details of how to acquire and abstract the data. Furthermore, the Application Controller can use the subscription-indication mechanism in application control. The Context Manager provides a common application programming interface, which can be used by all clients, including the Application Controller. API will be described later in more detail.

The Context Manager blackboard serves as a common data space for communicating information between clients. In addition, it serves as a context database, which can be used for storing a given number of context values for each context type. These values can be queried similarly to a relational database, except that the database client interface is simplified by hiding SQL query details. For each context type, an amount of context values to be stored is specified. When the maximum amount is reached, a number of the oldest values are deleted from the database. Context type concepts can be regarded as virtual tables in the database. Context type property and querying will be discussed in more detail later.

4.4 Context Source

The purpose of the Context Source entity is to connect to a data source and deliver contexts to the Context Manager. The source of the context data can be terminal internal or external. Context information from any source can be added to the blackboard, as long as the information is correctly formatted for storing and using through the Context Manager API. This section describes the design issues for the most important types of sources for processing sensor data. The examples on data flow and processing concern sensor data.

As was mentioned in the requirements, the abstraction level of the delivered data should be high enough and the frequency low enough for good performance in a blackboard system. In the case of raw high-frequency (sensor) data, abstracting and change detection should always be performed before delivery. Otherwise, abstracting or recognition should be performed in the Context Source in the following cases:

- Raw data is not needed by any client
- Abstracting makes contexts easier or more efficient to use.

Figure 10 shows the abstracting process in the Context Source. The process is similar to the pattern recognition process (Duda et al. 2001). The measurement (sensing) phase reads the sensors and outputs raw data. The preprocessing phase builds measurement data arrays that each contain a certain number of samples, and calculates general features for each time interval. The phase corresponds to the segmentation phase in the pattern recognition process, where the features for each time interval are considered a segmented object. The feature extraction phase calculates the actual abstracted features, which can be either numerical or symbolic. If symbolic features are required, feature extraction includes quantization and labeling the values with names corresponding to a real-world context. These names are defined in an ontology. The named features are also called context atoms, which can be used by applications directly or may require further refining by the context recognition. Context recognition may follow if required.

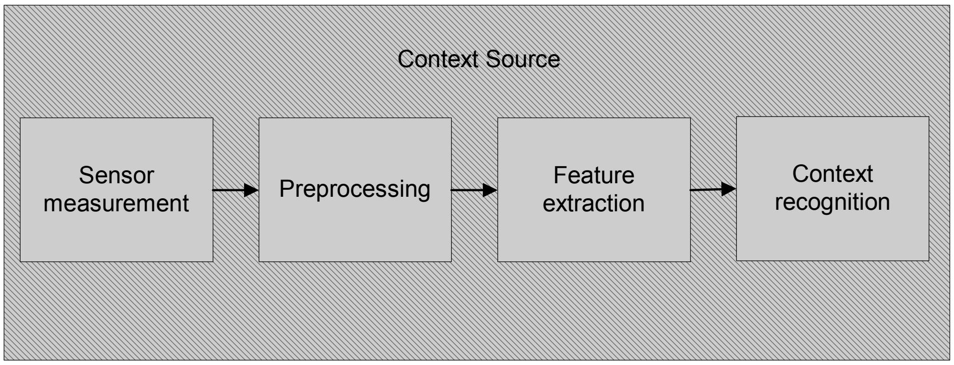


Figure 10. Phases of abstracting raw sensor data into human-interpretable context information.

Feature extraction or context recognition is not always needed prior to adding data to the blackboard. Numerical data values can be added after the preprocessing phase, if the application so requires data, and if the frequency of communication is not so high that it considerably reduces the system performance.

There are three most common types of Context Sources categorised according to the where the information is coming from, and whether the type of incoming information may change or not (Table 6).

Table 6. Three types of Context Sources and their properties.

Context Source origin	**Type(s) of context produced by the source**	**Examples**
1. Device internal	Fixed	Device sensor, device profile, device settings
2. External	Varying	Bluetooth, cellular network, RFID, other devices
3. External	Fixed	Bluetooth beacon (e.g., weather station)

In the case of external information, the source here refers to the terminal-side source implementation, which receives the information and adds it into the Context Manager blackboard. The source types have the following characteristic properties:

1. Device internal fixed information source. This type of source handles the information coming from the source that is embedded in the device, and always produces the same type(s) of context. Thus the source itself can contain context type-specific code for processing the data. This type of resource server is similar to the Context Toolkit widget. The difference is that the application is not required to find and contact the source itself but can access any context information from the Context Manager by context type through a common interface.

   For example, the device sensor source handles the information coming from the sensor that is fixed in the device. The data from the sensor is always of the same type, and the source can be implemented as specific to the type of sensor. One source can be used to process and add into the blackboard many different sensor signals.

2. Device external varying information source. Since the type of context information that is received from outside of the device can vary, this type of source must not contain any domain- or application-specific code for processing the incoming data. Instead, the task of this type of source is to route the incoming context information into the blackboard, or perform formatting of the context. The sources of this type can receive information either from local providers, such as a sensor, or remote providers, such as a networked server.

3. Device external fixed information source. In this source type the type of context information received from an external source is always constant, and the source can be domain-specific.

4.5 Context Abstractor

As was defined earlier and illustrated in Figure 5, context abstracting refers to the pattern recognition process up to and including either feature extraction or the classification phase. Context recognition refers to the pattern recognition process up to and including the classification phase. Hence context recognition is also context abstracting. The result of feature extraction can be a symbolic feature, context atom. The Context Abstractor entity here refers to both of these processes, which aim at producing human-understandable easily usable descriptions of raw data. Context abstracting and recognition can be implemented as part of Context Source, as a separate entity, Context Abstractor, or as a script that is executed by the Script Engine.

Context Abstractors can subscribe to any context type that is required for processing a higher level context, which is added back to the blackboard. Each time there is a change in the subscribed context type(s), the abstractor receives an indication and performs the abstracting or recognition. Plug-in Context Abstractors can be added to and removed from the system online. The application can operate by using the higher level contexts without needing to know about the underlying processing. Context Abstractors use as input either a set at a certain time instant or a time series of context atoms, and return single higher level contexts to the Context Manager.

The process of raising the abstraction level of the data may include transforming the features of recognition results into symbolic expressions that are human-understandable. Context Source, and when necessary, Context Abstractor can be used to perform the transformation of raw measurement data into a representation defined in the context ontology. When the context types have a definite set of human-understandable values, actions based on these values are easier to customize.

Different architectural options for implementing Context Abstractors need to be considered for the most performance-efficient solution. Based on the abstraction level of the incoming data, the type of communication and the frequency of incoming data, the abstracting should be performed separately or in Context Source. Table 7 presents a categorisation of the most relevant combinations of the mentioned properties with example sources, and proposes a feasible abstracting implementation.

Table 7. Context Abstractor implementation should be chosen according to the type of incoming data.

Category number	**Data abstrac-tion level**	**Data communi-cation type**	**Data frequency**	**Source examples**	**Abstracting element**
1.	Raw	Continuous	Low	Temperature, humidity	Abstractor or Source
2.	Raw	Continuous	Moderate	Acceleration, light	Context Source
3.	Raw	Continuous	High	Skin conductivity, acceleration	Context Source
4.	Symbolic	Continuous	Low	Network	Abstractor or Source
5.	Symbolic (or raw)	Event	Low, Moderate	User profile, UI events Bluetooth ID, RFID Tag	Abstractor or Source

The example categories are discussed separately:

1. Raw continuous sensor data sampled at low frequency can be directly added into the blackboard by the source, and the abstractor can subscribe to the data from the blackboard, receiving indications upon changes in the data. Both implementations are feasible.

2. When raw continuous data is sampled at a moderate or fast frequency, the abstractor should receive the data directly from the source, so that the Context Manager would not be overwhelmed with too much incoming data from many sources.

3. The same as category two.

4. Continuous low sampling frequency data can be added directly into the blackboard by the source. The abstractor can use the subscription mechanism. Both implementations are feasible.

5. Symbolic (or raw) event-based data should usually be added to the blackboard. Even if the frequency of events is occasionally moderate, the average performance should be sufficient. The abstractor can use the subscription mechanism. Both implementations are feasible.

The abstractors can subscribe to contexts recognised by other abstractors, forming a context abstraction hierarchy. The abstractors add the results to the blackboard and the application can use it through the Context Manager API, regardless of the way it was delivered.

4.6 Change Detector

Events in the real world measured by sensors reflecting context changes should be transformed from the measured signal to events that can be utilised by applications. The event-based interface to context information relieves the application from receiving data continuously, and hence from processing data continuously. The aim is to provide the application with only the relevant changes in the context, according to the changes in the real world situation, events that correspond to the usage needs of the application. The Change Detector entity is used for that purpose in the framework; there are five different options for implementing a Change Detector in the framework:

1. Perform change detection in the Context Source before adding data into the Context Manager blackboard. Context is added by the source only when change occurs. This is the preferred option if the frequency of the incoming data is high.

2. Perform change detection in the Context Abstractor. The abstractor may store, e.g., the previous context it has processed and check if there are changes. Context is only added to the blackboard by the abstractor if change has occurred.

3. Use Context Manager default change detection. The default function is to perform a string match for the incoming context values. If the incoming context value is not the same as the previous value of the same type, the context blackboard is updated.

4. Use a separate Change Detector, which makes a subscription to the Context Manager for a context type and monitors the values, indicating the specified changes. Changes can be indicated directly to an application or the Change Detector can add a new context to the blackboard, which is used by the application.

5. Implement Change Detector as a script that is executed by the Script Engine.

4.7 Application Controller

Context-aware features can be implemented with the context framework in two ways. Applications may use the Context Manager API directly for receiving context information, based on which actions are made. The other way is to use the Application Controller entity. The Application Controller handles activating application actions or system events based on the context events on behalf of the application. The Application Controller can subscribe to the Context Manager for receiving indications about changes in the subscribed context types. The Application Controller can thus operate as event-based – i.e., it performs control actions when it receives change indications about the subscribed context types.

The Application Controller encapsulates a selected application inference approach, or combines multiple approaches. The control inference can be, for example, rule-based, fuzzy or probabilistic. The context framework does not restrict the inference approach, nor does the context representation. A rule-based application control inference approach was selected in this dissertation. The rule-based inference approach does not directly facilitate handling uncertainty. The uncertainty in sensor signals is eliminated by the producer layer components, i.e. Context Sources or Context Abstarctors. Hence the Application Controller can produce discrete event-based application control commands.

4.7.1 Script engine

The Application Controller contains a script-based inference engine that can execute arithmetic and logical operations, which are described as text-based XML scripts in Context Exchange Protocol (CEP) syntax (Lakkala 2003a). The Script Engine element in the context framework was adopted in this dissertation. The Script Engine was developed and implemented by Ilkka Salminen and Harri Lakkala (Lakkala 2003b); the purpose of the Script Engine within the context framework has also been documented by Korpipää et al. (2005b).

The Script Engine enables describing context-based application actions as rule scripts. The Script Engine can subscribe to context types that appear in the script, evaluate the script when changes in the subscribed context types are indicated by Context Manager, and indicate the evaluation result forward to the Activator. Hence the Application Controller can operate as, for instance, a discrete rule-based controller.

Script Engine can further be applied to perform straightforward abstracting and change detection tasks as well, by indicating the evaluation results back to the Context Manager blackboard. This can also be done without implementing executable code.

4.7.2 Activator

For triggered rules, the Activator launches the designated application functions or system events. Each action expression consists of a human understandable part and machine executable function parameters. The function parameters are carried in the indication message of the CEP script action part. Each action expression is configured in the ontology action vocabulary read by the Customizer. Hence the human understandable part is visible to the user and the user can perform customization based on these action expressions.

The Activator contains function interfaces for executing the application actions and system events that are available for controlling. As was mentioned, each application action instance is configured as an expression where the human understandable part corresponds to parameters for the application functions. The inference engine indicates the machine executable part of the action expression forward to the Activator upon a triggered rule.

4.8 Customizer

Every development process has two basic phases: design time and use time (Fischer et al. 2004). At the design time, the needs and objectives of the user of context-aware application can only be anticipated. The users may find hard-coded features unsuitable at use time, requiring modification. Moreover, the user needs may change over time.

The idea of a Customizer is that instead of implementing the context-aware application features at design time, a set of contexts and actions are provided for the user. The task of the user is to decide whether and how to use them. The Customizer is a tool that lets the user define context-action behavior into existing mobile device applications at use time.

In other words, the Customizer is a tool for configuring the Application Controller. It can be used to connect context events to application actions – i.e., inputs to outputs. The customised context-action features can be described as rule scripts that can be read by the inference engine. The tool can generate such scripts based on the graphical descriptions given by the user with the tool. The Application Controller receives context events from the Context Manager, and activates application actions as specified with the Customizer. The Customizer is targeted at end-user usage.

4.9 Summary

An architectural design for the context framework for mobile device sensor-based context-aware applications was given in this chapter with implementation recommendations. The design realises the requirements. Hence the framework is designed with regard to the additional restrictions and characteristics of mobile devices. The related context frameworks have been designed for distributed PC environments, which have different requirements. Moreover, the blackboard-based architecture model was chosen in the requirement analysis, based on the practical requirements, the literature, and experience, as a most suitable model for mobile context management, compared with the widget and client-server models. The given design, first published in 2003, is the first blackboard-based software framework for managing context-related information in mobile phones.

More specifically, the design differs from the related work in facilitating the following aspects. The related work does not provide a software framework for developing mobile device sensor-based context-aware applications, mobile device framework support for providing fast event-based abstracted contexts defined in ontology, framework support for context abstracting and context recognition process in a mobile device, blackboard-based management of context information in a mobile device, a relational context database for mobile device context management, and framework support for application control and interaction customization in mobile devices.

5. Context representation and ontology

"Plurality should not be assumed without necessity."
- William of Ockham, ca. 1285–1349

This chapter is partly based on the articles by Korpipää and Mäntyjärvi (2003) and Korpipää et al. (2003b, 2004a). The results of Korpipää and Mäntyjärvi (2003) and Korpipää et al. (2003b) are further developed and additional details are presented. The results of Korpipää et al. (2004a) are partly reused and additional details are presented.

In the previous chapter the design of the framework, based on the requirements and aiming at implementation, was presented. The purpose of each framework element in processing data was explained. This chapter explains how to represent abstracted data within the framework.

In software engineering, data structures define data representation. Data structure definitions in program code are rigorously formal, and their main purpose is to be efficient for machine processing, while readability for humans is of less concern. When data representation is required to be easily human-understandable, semantics are of importance. Such understandability is required in customization for instance.

In this dissertation context ontology serves the purpose of representing context information so that it is easily human-readable in addition to the machine processability. The ontology consists of two parts: structure and vocabularies. Structure defines the common properties of context that are used across different domains and applications. Vocabularies are application- or domain-dependent expandable context conceptualisations, which aim at understandability and simplicity for the end-user and the application programmer. New vocabularies are developed for new domains according to a vocabulary model. The term ‘context representation’ here refers to the entity of representing context within the framework, which includes ontology, syntax and implementation.

5.1 Requirements for the ontology

The framework requirements were defined in the requirements analysis chapter. Additional requirements concerning the representation and ontology are presented in this chapter, where the issue is discussed in detail.

Sensors can be used to acquire information from the environment and the usage situation. Traditionally, sensor information is mostly utilised as raw numerical data in mobile computing. The information value and the usefulness of raw measurement data is low for the end-user or application developer. Raw sensor data can be abstracted for understandability, and abstractions reduce the amount of data traffic from the sensor to the application.

The processing of context information from several low-cost sensors integrated in mobile terminals is carried out using signal processing methods to extract suitable features. The suitability of the extracted features should reflect the concepts of the real world, and they should be useful for applications. Hence the purpose of the context ontology is to define how (sensor-based) context information should be represented with regard to its real-world use.

The following requirements were the main guiding principles in designing the ontology. The goals are listed in the order of emphasis that was put on them in design:

1. Simplicity. Choose the simplest necessary representation. The ontology structure and vocabulary model should be simple enough to be easily utilised by application developers. Vocabularies should be easily understandable to the end-user. Expressive and detailed ontology is not useful if it is too complex compared with the necessary level of detail required by most applications.

2. Practical access. The ontology should enable simple, practical and efficient queries and subscriptions to context information through the Context Manager API.

3. Flexibility, expandability. The context ontology should be expandable to cover new domains, and the existing vocabularies should be modifiable.

4. Domain. The ontology should support easy utilisation of abstracted mobile device sensor-based context information.

5. Facilitate inference. The representation should enable efficient inference by the Context Abstractors as well as the Application Controller. It should not restrict the inference to any single method, since the efficiency of a method is dependent on the type of task.

6. Genericity. The ontology should support different types of context information.

7. Efficiency. The representation should be memory-efficient.

8. Expressiveness. The possible amount of detail in describing any single context and the versatility of the expressions should be high.

5.2 Structure of the ontology

The ontology structure is defined as a set of properties. Each context (object) is described using six properties, shown in the list below. Each context instance is required to contain at least *Context type*, *Context value* and *Source* in order to facilitate the practical management, storing and usage of context information through the Context Manager.

- *Context type* is the category of the context, which operates as a variable name. All subscriptions and queries must have context type as the primary parameter. Context type is an identifier of context instance, together with source.

- *Context value* is the semantic or absolute value for a context type, which operates as variable value, used together with context type. Context value may alternatively or additionally contain an absolute numerical value or feature.

- *Source* is used to describe the semantic source of context. It can be used by a client interested only in contexts from a specific source. Source can describe the entity that the context instance represents.

- *Confidence* is an optional property of context for describing the uncertainty of the context instance. Confidence can describe, e.g., a probability or a fuzzy membership.

- *Timestamp* denotes the latest time the context occurred.

- *Attributes* can be used to specify the context expression freely, and contain a pointer to any additional properties of details that are not included in the other properties.

The obligatory properties of context can be used as a tuple (ContextType, ContextValue) or as a triple (ContextType, ContextValue, Source). API functions are based on using these properties as identifiers. The assumption is that most context information can be represented with a tuple – i.e., a name-value pair – or a triple.

Attributes can be used to specify the context instance freely, when the other properties are not specific enough. Attributes can be represented as name-value pairs. For example, attributes can contain the unit for context value in the case of an absolute context value, or the type of confidence. It is possible to implement the access of attributes as separate from primary properties, so that the application does not receive an unnecessarily large amount of detail when accessing the primary context information.

Relations between contexts are modelled in abstractors, which receive context values from the blackboard and monitor relations between the specified contexts or within one context type. If the specified relation or pattern is detected, new higher level context is added into the blackboard. Even though this higher level context may be a result of multiple context objects, it can be added into the blackboard as a new single context object. For example, for a rule "when A happens before B, then C", an abstractor is implemented, which subscribes to listen to A and B and corresponding timestamps, and when the rule is fired, C is added into the Context Manager blackboard. There is no need to dictate in the ontology how the relations must be represented. Hence the rationale for describing complex relations is to allow any kind of description, and any kind of inference operating on that description, as long as the resulting context is represented with the properties of the context ontology structure.

5.3 Ontology vocabulary model

Ontology vocabularies are designed according to the domain or application needs. Vocabulary design is a process of defining domain- or application-specific values for the *Context type* and *Context value* -properties that are understandable by humans. Those values should categorise and describe the possible real-world situations so that the information is useful for applications, and understandable to humans, according to the requirements.

Context types are defined to name and categorise context values. Context type resembles a variable name, and context values resemble the set of values for the variable. Figure 11 illustrates one context type and a set of three context values for the type. Context type can have one or more concepts. Each context type can have one or more context values. Context instance at a certain time can have one of the values for each context type. Different context types can have one or more common concepts. Hence it is possible and useful to form a tree hierarchy of context type concepts, where the leaf nodes represent the context values. The hierarchy can be utilised in querying and subscribing to branches of the context tree, rather than only one path.

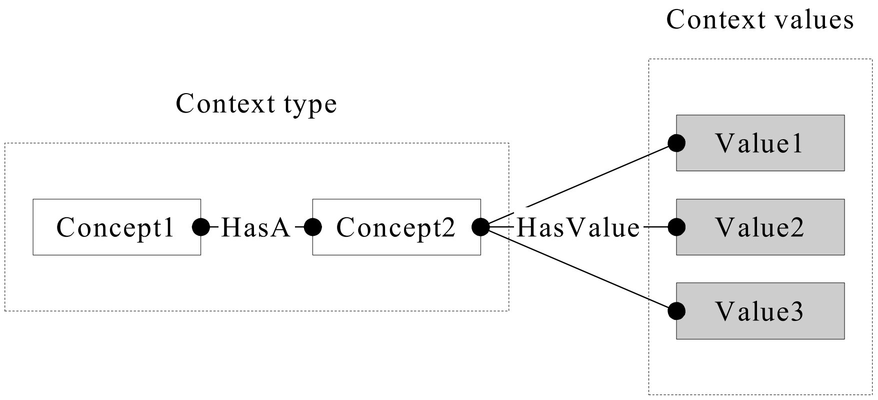


Figure 11. A model for creating a vocabulary consisting of context types and context values.

Context type concepts should be categorised from generic toward specific, so that the more generic concepts are to the left towards the root and the more specific concepts are to the right toward the leaves. In this manner, context types can have more common generic concepts but are separated by the more specific concepts.

For simplicity and ease of use, very long context types should be avoided. At maximum, context types should have three or four concepts. For example, if context type concepts are modelled as folders in a user interface, navigating deep folder hierarchies is slow. On the other hand, context types should be specific enough to allow having a small set of values. Too large a set of values is difficult to handle, at least in a user interface, if the user has to choose from a set of values. Each context value should have a potential relevance to some application.

In customization, actions to be performed based on contexts can also be represented by using the context vocabulary model. Actions are defined with two properties, action type and action value, which describe actions similarly as context type and context value describe contexts.

5.4 Naming conventions

Appropriate naming is essential, particularly in describing the obligatory properties, context type, context value and source, which are required for each context object added to the context manager blackboard. These properties are also used for accessing context information from the Context Manager.

There are two main aspects to be considered in naming context types. First, appropriate naming should reflect the meaning of the context to the user, which is either the application developer or customization tool user. Naming should reveal the use of the context. Second, a correct naming convention ensures that the user of the context information can fully utilise the features provided by the Context Manager. When the context types are built as paths consisting of elements from generic to specific concept, the context information can be accessed with partial context types that represent a larger context information subset (Korpipää & Mäntyjärvi 2003). This naming convention has also been utilised in CEP, where the reference to a subset of context type hierarchy is called a wildcard (Lakkala 2003a). Moreover, CEP recommends that vendor-specific context types are named starting with a prefix that names the vendor, e.g., “x-vendor_name:”, followed by the normal context type definition.

The set of context values can be either numerical or symbolic. If context values are described as numerical, for application use, they should be understandable by humans. If raw numerical measurement values are used, naming a context value set is not required. Context type can be used normally. Numerical values can be divided into named intervals – e.g., a temperature value can be "over 20". If context values are defined for the purpose of customization, they should be understandable by humans, and symbolic, and the number of values in the set should be low enough to allow choosing a value from the set to function as a condition in a rule.

The context source property does not need to be described in the vocabularies, but it is required if there are many context providers for the same context type and the client is interested in contexts originating from a specific source. A naming policy for the source is necessary to avoid naming conflicts between different context information providers. The categories of sources are terminal internal and external. Terminal internal sources can be named starting with the prefix "Phone", followed by a description of the source, such as sensor name. If the terminal source consists of multiple elements, it should be named similarly to the context type, where the name forms a path, e.g., "Phone:AccelerationX". Terminal external sources can be named as, e.g., the URI or IP address of the source.

5.5 Example vocabularies

Ontology vocabularies were designed for mobile device contexts and application actions for use in customization. The vocabularies were designed as shared conceptualisations. A sensor-based context vocabulary was presented by Korpipää and Mäntyjärvi (2003), and an audio-based context vocabulary was given by Korpipää et al (2003a). Parts of a vocabulary for customizing context-aware applications were given by Korpipää et al. (2004a, 2005a, 2005b). Furthermore, in the Nomadic Media project vocabularies were designed for describing the context information of several domains. A few parts of the vocabularies are presented as examples in this chapter.

In a sensor-based context vocabulary (Korpipää & Mäntyjärvi 2003), the Environment category vocabulary consists of contexts that represent the state of the environment. Sensors measuring the environment include temperature, humidity, sound, and light, embedded into a sensor box for a mobile device (Tuulari 2000). Figure 12 illustrates a part of the example environment context vocabulary in a tree form.

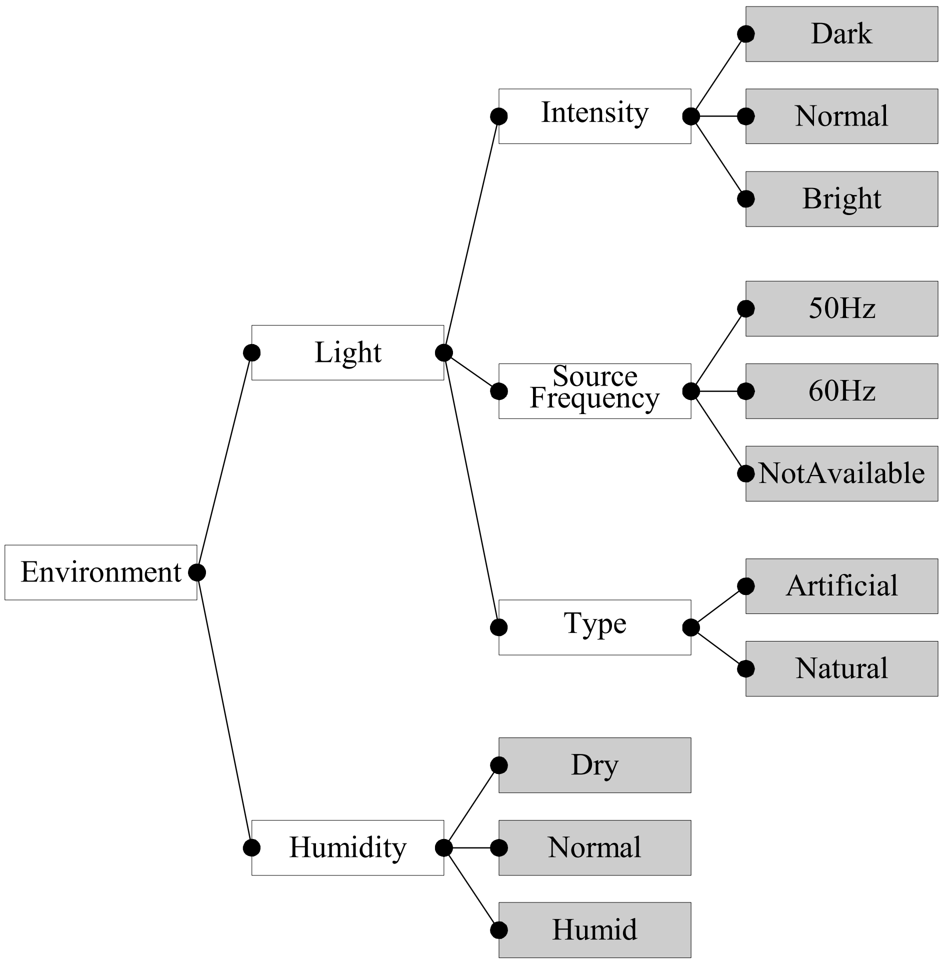


Figure 12. Part of the example environment context vocabulary.

The leaf nodes (grey boxes) represent context values, and the path to the context value (white boxes) represents the context type. The vocabulary can be expanded if new contexts become available. Table 8 presents the same environment vocabulary part in a list form, with other context types included as well.

Table 8. Environment context vocabulary as a list.

Context type	**Context values**
Environment:Light:Intensity	Dark, Normal, Bright
Environment:Light:SourceFrequency	50Hz, 60Hz, NotAvailable
Environment:Light:Type	Artificial, Natural
Environment:Humidity	Dry, Normal, Humid
Environment:AirPressure	Low, Normal, High
Environment:Temperature	Below -20, -20 to 0, 0 to 20, over 20
Environment:Sound:Intensity	Silent, Normal, Loud
Environment:Sound:Type	Car, Elevator, Speech, RockMusic, ClassicalMusic, TapWater, OtherSound

Vocabularies can represent contexts at many levels of abstraction. The application that uses the context does not need to know about the underlying abstracting processes needed to produce the context values. For example, Environment:Sound:Type contexts require sound classification from multiple lower level features (Korpipää et al. 2003a), whereas Environment:Humidity context values are the result of a feature extractor utilising fuzzy quantization, which produces symbolic features, context atoms (Mäntyjärvi et al. 2001). The reliability of the context values depends on the reliability of the measurement and abstracting process.

There are two categories of context types based on their values. The first category has a value that can be produced from measurements in all situations. Such context types are, for example, temperature, light and sound intensity. All possible measurements at any moment can be represented as one of the values defined for the type, unless the sensor output is erroneous. In the other category, NotAvailable -context is required to define the state when context value cannot be produced from measurements, and the current value is not the previously detected one. For example, when the light type is natural, either of the defined SourceFrequency values is valid, so the value for that context type is NotAvailable. NotAvailable is similar to the OtherSound value for sound type, which can be interpreted as having none of the recognisable values available, since they could not be recognised by the abstractor.

The device category vocabulary describes concepts that relate to the device itself. This category is large, since it includes such device activity context types as Movement, Orientation, Keypad, CallStatus, ForegroundApplication, BatteryStrength, NetworkStrength, Charger, NetworkName, etc. The example in Table 9 presents some of the acceleration sensor-based abstractions of the device category vocabulary in a list form.

Table 9. Device context vocabulary.

Context type	**Context values**
Device:Orientation	UpsideUp, UpsideDown, DisplayUp, DisplayDown, DisplayRight, DisplayLeft
Device:Movement:Swing	Push, Pull, SwingUp, SwingDown, SwingLeft, SwingRight
Device:Movement:Activity	Still, Activity
Device:Movement: AccelerationPeak	Low, Moderate, High

Abstracting contexts from sensor data that describe the user activity is very challenging since the user activity can be only indirectly inferred based on the device sensors. Moreover, the activities that produce features referring to, and recognised as, user activities can ambiguously be produced by other real-world situations. For example, measurements can indicate the typical acceleration frequency of walking or running, but it cannot be proved that the feature is the result of the user walking.

In the Nomadic Media project context vocabularies were specified for the domains of airport, home and hospital, according to the vocabulary model. Table 10 presents an example of the airport domain location vocabulary.

Table 10. Airport domain location vocabulary.

Context type	**Context values**
Location:Facility	Airport
Location:Airport:Name	Helsinki–Vantaa
Location:Airport:Code	HEL
Location:Airport:Area	Arrivals, BaggageClaim, Departures, Parking
Location:Airport:Terminal	Domestic, International
Location:Airport:HotSpot:Name	Registration, Arrival, Check-In, FreeTime, Departure
Location:Airport:HotSpot:NearbyDevices	BT IDs in range (array)

5.6 Context instances

The ontology structure defines the primary properties of the context information, and the vocabularies categorise and name the domain-specific types and values of the context. In other words, the vocabulary defines the set of contexts that may occur in certain domain. When the context management system is online, multiple context instances can, and usually will, exist at a certain time instant. Each instance is represented by the context type, context value, source, confidence and other properties defined in the structure. Depending on the implementation, the context instance can be represented – for example, as a context object. The Context Source and/or abstractor transforms the incoming data into a context instance by assigning values to the properties, after which the instance can be added into the Context Manager blackboard. Table 11 lists a few examples of possible context instances according to the ontology. Timestamp, source, and attributes properties are omitted.

Table 11. Examples of context instances from the ontology.

Example number	Context type	Context value	Confidence
1.	Environment:Light:Type	Artificial	1
2.	Environment:Temperature	Over 20	1
3.	Environment:Humidity	Dry	.7
4.	Device:Placement	AtHand	1
5.	Gesture	Throw	1
6.	Environment:Sound:Type	RockMusic	.8
7.	Location:Facility	Airport	1

Independent of the source, underlying processing and abstraction level, all context instances are represented with the same structure, and can be used by the application through a common interface. Moreover, raw data values can be represented as instances on the blackboard as well, if required by the application. The optional confidence value represents fuzzy membership of the context in example three, and probability in example six. Other values are crisp and hence use the default confidence one. If necessary, the interpretation of the confidence can be identified in the attributes property.

5.7 Interpretation of symbolic values in vocabularies

Context values defined in the vocabularies can be divided into two categories: subjective and objective. Symbolic context values abstracted from measurements such as light, temperature and humidity can have different interpretations in different situations or by people from different cultures. Hence subjective contexts, such as bright light, suffer from a degree of ambiguity, which reduces their applicability. Generic use of subjective contexts is difficult. Vocabularies with subjective symbolic values should be designed according to the domain or application needs. Absolute values can be used in addition to or instead of the symbolic value when necessary. In customization, subjective values should be either omitted or their meaning should be presented explicitly, since otherwise the users are puzzled about their meaning (Korpipää et al. 2004a).

For example, if values for temperature context are defined as a fuzzy set of Cold, Normal, Hot, the values are strongly situation- and observer-dependent. Another way to express temperature would be having the temperature in Celsius degrees as a context value. The context change could be indicated when the integer changes. The solution is feasible in this case: first, the context is directly meaningful to humans; and, second, the value is an unambiguous and objective fact. The absolute and semantic contexts could be separated by naming them in the context type – e.g., Environment:Temperature:Semantic.

However, absolute values are not necessarily as feasible to use for all context types. For instance, a light-intensity context could be measured in lux. If the same change detection principle of monitoring absolute integer values is used, the light context would be more change-sensitive. Ideally, the message traffic from changes should not be more frequent than that required by the application. One solution for reducing the change sensitivity is to define larger steps for the intensity value – for example, ten lux – and a change indication would occur upon a step change. More difficult to handle as raw values are features that are not directly interpretable by humans. For example, Device:Movement:Activity context values are abstracted from a feature indicating the maximum standard deviation for a certain time interval from each acceleration channel. Using such a feature as a numerical value would require further explanation for both an application developer and a personalisation tool user. Abstracting before use is highly recommendable for such features. Furthermore, having a large number of values for context should be avoided if the goal is to enable user personalisation of context-aware behaviour.

Objective contexts are situation- and observer-independent. Examples of such contexts are numerical temperature, light type, and device orientation. These contexts can be considered generic, and generally should not be interpretable differently in different situations or by different people.

5.8 Syntax

Using context in mobile devices requires representing the context in a machine-readable form, syntax. The suitability of the choice of syntax is task-dependent. For example, if the aim were to create a logic inference engine that operates based on the syntax, it would be reasonable to use e.g. OWL, assuming that performance issues were not an obstacle. If human readability and wide adoption were the primary criteria, XML-based syntax would be well founded. If memory and time efficiency of processing context information were the most important criteria, an efficient syntax, such as object-oriented language structure, would be the optimal machine-readable syntax.

Since the focus in this dissertation is on terminal context management, the optimal syntax choice is an object-oriented language data structure. Context instances can be represented as context objects that encode the context properties defined in the ontology structure. If the external context sources provide context instances in a different syntax, messages have to be transformed before adding them to the terminal Context Manager blackboard. XML-based CEP (Lakkala 2003a) is directly compatible with the context framework described in this dissertation, and can be applied as the formal syntax when context data is sent from external sources to the terminal or from the terminal to external entities.

The context object is used to encapsulate a context instance within the terminal context framework. For the purpose of use in customization and sharing, the Context Exchange Protocol syntax (CEP) is utilised (Lakkala 2003a). CEP is compatible with the ontology structure defined in this dissertation and published by Korpipää and Mäntyjärvi (2003). CEP is an XML-based format that uses the XML Schema for specifying the meta-structure of the format. XML was chosen in CEP due to its extendability, easy debugging, and message legality checking, and XML was chosen over RDF based on its better understandability by human readers (Lakkala 2003a). A simplified example of a single-value context instance described in CEP compatible with the ontology is as follows:

```
<atom name="Location:Airport"
source=""
userId=""
timestamp="">
<string name="Terminal">Domestic</string>
</atom>
```

Name corresponds to the context type, Location:Airport:Terminal, and the context type has a context value, Domestic. Other attributes are omitted. Furthermore, CEP can be applied to describe context-action rules, to be applied for application control. An example of a CEP rule script (for which a graphical representation is later shown in Figure 18) compatible with the ontology is as follows:

```
<script xmlns="http://www.nokia.com/ns/cep/script/1.0/"
xmlns:cep="http://www.nokia.com/ns/cep/1.0/">
<if>
<and>
<equal>
<atomRef name="Device:Charger" />
<cep:string>Charging</cep:string>
</equal>
<equal>
<atomRef name="Location" />
<cep:string>Home</cep:string>
</equal>
</and>
<actions>
<notify message="Application function parameters" />
<cep:atom name="ExternalDevice:ImageAlbum">
<cep:string>SaveNewImages</cep:string>
</cep:atom>
</actions>
</if>
</script>
```

Each CEP rule, applied for application control, has one or more condition and one or more actions. The action part contains the human understandable action expression to be applied in customization, and machine executable application function parameters. In the previous example the machine executable application function parameters carried by the 'notify message' are omitted for clarity. The notify message has a separate syntax.

In other words, context rules and context instances can be formally represented and shared by applying CEP. The syntax for formal representation of the vocabulary itself is not required in the application areas discussed in this dissertation. The vocabularies, i.e. the list of available context types and their values, are represented in a semi-informal way for the best human readability, and for rapid utilisation by the application developers and customization tool users. Alternatively, context and action vocabularies can be described as a collection of CEP instances.

5.9 Discussion

Requirement one suggests that the encoding of the ontology vocabulary should be towards lightweight and semi-informal. This choice does not make the vocabulary maximally expressive, as is aimed at by requirement eight. A balance has to be found between the usability of the representation and the expressiveness. The representation should not be overly specific for the task it is designed for. For instance, a formal ontology could represent that the Environment: Light: SourceFrequency only makes sense if Environment: Light: Type is Artificial. In the current design this knowledge does not need to be separately modelled since the Environment: Light: SourceFrequency has the value NotAvailable, which straightforwardly expresses the same information for the application.

Nevertheless, the choice of a semi-informal vocabulary syntax does not exclude the possibility of incorporating a more formal ontology vocabulary representation, based on, e.g., the requirements four, three and eight. It is to be noted that all requirements cannot be fulfilled maximally by one design choice.

A more formal representation of the vocabulary could facilitate expressing the constraints of the domain. For instance, the ontology vocabulary could state the limits of allowed context values, which could then be checked by the Context Manager. In a terminal-centric system the main use for such a feature would be system self diagnostics, i.e. an incorrect input value could be detected and indicated or rejected. The formal representation of the vocabulary semantics becomes more relevant for context instances received from outside the terminal.

Concerning the validity of context values as a function of time, a formal vocabulary representation could better express the validity time limits or the allowed history length for each context type. The validity time window or history length for persistent values can be application-dependent. In sensor-based terminal-centric context management applied for enhancing interaction, most context values are likely to be useful as non-persistent at the time instant of an event occurrence. However, to conclude, incorporating a more formal and expressive vocabulary syntax is further work.

5.10 Summary

Semantic descriptions can be used for representing abstractions of sensor-based context data. An ontology was introduced for representing context information. The ontology consists of two parts: structure and vocabulary. Structure defines the common properties of context that are used across different domains and applications. Vocabularies are application- or domain-dependent expandable context conceptualisations, which aim at understandability and simplicity for the application programmer and user. A generic vocabulary model was presented for creating new vocabularies for new domains. Examples of context vocabularies for multiple different domains were represented.

The main contributions of this chapter are a requirements analysis for context representation and an ontology for a mobile device, structure and common properties for domain-independent representation of context information as data objects, and a generic vocabulary model for describing context instances and vocabularies.

6. Context abstracting and recognition

This chapter is partly based on articles by Korpipää et al. (2003a, 2003b). The discussion on context abstracting and recognition by Korpipää et al. (2003b) is extended at length. The case study part is, for the major part, the same as that discussed by Korpipää et al. (2003a).

Representation and ontology for describing abstracted context data was given in the previous chapter. This chapter will explain how context abstractions can be formed from a continuous raw data stream. The abstracted data is then represented as described by the ontology.

The framework supports inference based on context information. As was discussed earlier, inference here refers to context recognition or application control, and the former is discussed in more detail in this chapter. In the framework, context abstracting and recognition takes place in the plug-in Context Abstractors or Context Sources.

6.1 Requirements for context abstracting methods

The requirements for Context Abstractor entities from the architecture viewpoint were analysed earlier. The additional requirements discussed here concern the capabilities of context abstracting methods for utilisation in mobile context-aware computing. Context abstracting methods are used for the phases of feature extraction and classification, referring to the process of pattern recognition (Duda et al. 2001).

1. Efficiency. Efficiency is a primary requirement for recognition methods in mobile computing. The processing, battery and memory capacities in mobile devices are limited. The abstracting methods should have as low time and memory complexity as possible. The amount of processor time and memory accesses is proportional to the battery energy consumption.

2. Handle multidimensional input data. Many sources of context are potentially available for a mobile device. The classification methods should be able to

efficiently handle multiple input features for inferring higher abstraction level contexts. Moreover, the situations of mobile device usage are dynamic and can change rapidly, requiring fast response times as well.

3. Handle uncertainty. The input data used for context abstracting is not always perfect. Context sources may produce incomplete, incorrect or inconsistent information. The abstraction methods should be robust to uncertainty.

4. Updating flexibility. Other criteria include the flexibility of updating the (learned) models. If the system is designed to learn online, flexibility becomes an important criterion, as well as the learning efficiency and the amount of training data needed to create the model. Furthermore, model extensibility and modifiability is required when new contexts become available.

5. Scalability. When the system is extended to contain large models or a large number of them, it should sustain its performance.

The process of context (pattern) recognition consists of phases that process data at different levels of abstraction. Different context abstracting methods have strengths at different phases of the pattern recognition process. Hence it is justifiable to combine these strengths by applying different methods for the feature extraction and classification phases of the recognition process.

6.2 Inference for context abstracting within the framework

The context framework and the ontology have not been designed for any specific inference method. The framework allows the use of any inference method. The Context Manager hides the details of the context abstracting from the client. A collection of agents takes information from the common data space, processes it, and returns more abstract information to the blackboard. The approach is commonly known from blackboard systems (Engelmore & Morgan 1988). There is no need to restrict how the agents represent and process the information as long as it is returned as defined in the ontology structure. Figure 13 offers a simplified view of the inference approach.

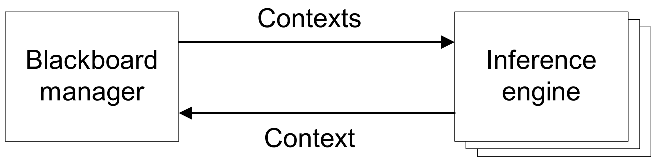


Figure 13. Inference engine(s) receive contexts from the blackboard and return abstracted contexts back to the blackboard in a form defined in the ontology.

Abstracting context from multiple sources may require methods that can manage incomplete information. Context uncertainty is represented with the confidence attribute, which can be utilised in several inference methods, such as probabilistic networks and fuzzy logic.

Contexts of different levels of abstraction can be added to the blackboard by the inference engines. High-level context refers to a context inferred from an assembly of lower level contexts. Both abstractions can be described with the same ontology structure. Higher level contexts can be derived from either a set of context values of different types at a certain time instant or from a context history. Context history is a series of successive instances of contexts, which are stored in the blackboard manager database. The ontology does not need to model time sequences. Time-dependent relations can be modelled in the inference engines, which receive a sequence of contexts from the blackboard and return contexts as defined in the ontology.

The Client does not need to know the abstractor input context types when it is using an abstracted high-level context. The Context Abstractor subscribes to the input context types, and gets indications from the Context Manager upon value change for the subscribed types, and adds the new abstracted context into the blackboard. Hence the abstractors only process data when there have been changes in the inputs, instead of continuous processing. The Client only needs to subscribe to the abstracted context type to be informed about the changes in it.

It is possible to implement domain-specific inference engines of any kind, which are "hard-coded" for certain abstracting tasks. Another solution is an inference engine that can operate on context data with a set of previously defined operators, such as condition, logical, and comparison operations (Lakkala 2003b). By combining these operators, straightforward abstracting and change detection tasks can be performed. Moreover, inference tasks can be described as XML scripts, without implementing executable code.

It is notable that the semi-informal lightweight ontology described in this dissertation is directly compatible with and does facilitate straightforward context abstracting and change detection tasks based on the CEP XML scripts and the inference engine (Lakkala 2003b). Context Abstractors in the framework can be implemented without programming executable code, as scripts that can be plugged in. The Script Engine is adopted in this dissertation and it is not the contribution of the author; hence a more detailed discussion can be found elsewhere (Lakkala 2003a, 2003b, Korpipää et al. 2005b).

6.3 Multidimensional contexts

At each moment of time a set of context instances describes the overall situation. The set of instances can be represented as a context vector, where each vector element is a context instance. The context vector can represent a higher abstraction level context. Similarly, a time sequence of context instances can represent a higher level context.

Hence vector representation is one of the possible representations that can be used by the Context Abstractors for inferring abstractions from vectors of context instances. However, all contexts on the blackboard are treated as atomary. When an application subscribes to a context that is abstracted from multiple sub-contexts, the Context Abstractor can be used to recognise the desired context from the sub-contexts. The input for the abstractor is a vector consisting of context atoms, and an output is an atomary higher level context.

High-level context described as a vector needs to be labeled if human understandability of the context is required. In unsupervised learning of contexts from multidimensional data, generating proper labels understandable by humans is a challenge. Another challenge is learning context-action patterns. For example, the learning system could discover a cluster corresponding to a context (Himberg et al. 2001; Flanagan et al. 2002), and the user regularly performs a certain action in that context cluster. Based on this knowledge, the system could automatically generate a rule that connects the context to the action. However, practical challenges remain, such as how to automatically discover those context-action patterns from the data that are relevant for the user.

6.4 Context recognition case study

The process of context abstracting and the implementation choices of the process phases within the framework have been explained. The representation for the abstracted contexts was given. An example of the process of context recognition from multi-sensor data is given in this section. A study of recognising sensor-based contexts from continuous data measured from a real-world scenario is presented. The chosen abstracting methods are assessed against the requirements. This section is partly based on the article by Korpipää et al. (2003a).

The purpose of the context recognition case study is to initially examine the potential feasibility of multi-sensor context recognition. The study takes the form of an offline pattern recognition experiment, where the data is collected with a measurement system in which the positioning of the sensor box corresponds to positioning a mobile phone in the front pocket of a jacket or shirt. In case context recognition of this type is found feasible and applicable, the future prospect is to have the sensors integrated in a mobile phone, where the recognition is processed by the context framework, i.e. Context Sources and Context Abstractors.

6.4.1 Feature extraction

In the pattern recognition process the phases of sensing and segmentation take place before the feature extraction. In the study, data was collected from nine channels – three for acceleration, two for light, and one for humidity, temperature, touch and audio. Segmentation was substituted by dividing the data into one-second intervals, which were each analysed separately. In terms of pattern recognition, each one-second snapshot from all the channels represents an object or pattern, the target of classification.

Feature extraction is the next phase in the recognition process. The task of feature extracting is to raise the abstraction level of the data and express the information contained by the data more compactly. Many alternative methods exist for extracting features. In the study, symbolic features – context atoms – were extracted. In terms of AI, the processing step of generating context atoms is a signal-to-symbol conversion (Engelmore & Morgan 1988). Numerical context feature representation is additional to or an alternative expression of symbolic values (Korpipää et al. 2004b). The decision on which kind of features are extracted depends on their use. If the extracted features are designed to be directly used in personalisation, they should be understandable by humans and symbolic.

The features to be extracted were chosen according to how well they describe aspects of the real-world situation of the mobile device user (Mäntyjärvi et al. 2001, Himberg et al. 2001). The use of symbolic features was justified on the following basis:

- Symbolic human-understandable values facilitate the user's task of customizing context-based actions.
- Application control methods, such as fuzzy control, require the semantic expression of features (Mäntyjärvi & Seppänen 2002).
- Context atoms can be used directly by application developers or users without further processing.

The following two methods were used for producing context atoms:

1. Set crisp limits for the chosen feature. The result is a Boolean expression of each context atom value. For example, in the case of light intensity (ontology context type Environment:Light:Intensity), the context values are Dark, Normal and Bright. If one of these is true, the others are false. The line intersections represent the crisp limits (Figure 14).
2. Apply a fuzzy set for the chosen feature. The result is a fuzzy set in which the instance has values according to a membership function. The intensity of light can be, e.g., Normal with a membership of 0.3 and Bright with 0.7 (Figure 14).

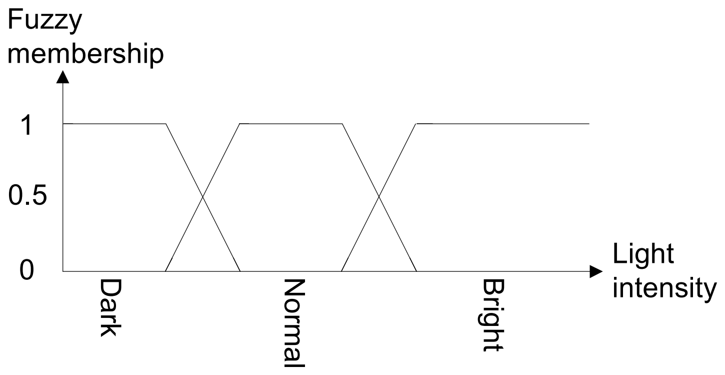


Figure 14. Quantization example.

The primary motivation for having used fuzzy sets is that many events in the world are fuzzy. Fuzzy quantization can be viewed as a granulation of information, which makes it possible to exploit the tolerance for imprecision by focusing on the information that is decision-relevant (Zadeh 1996). In the example, the boundary between normal and bright light is fuzzy, and a two-valued expression would be coarse. With a fuzzy set the value can be something in between the two symbolic values.

The next step is to classify the objects into a set of predefined categories. Each object is a vector representing one second of data, containing all the features (context atoms) calculated from the measurements for each channel. Each element of the vector represents one feature.

6.4.2 Classification

Extracted features can be directly utilised by the applications. They can also be used for further abstracting – e.g., for classification. Within the framework, the classification phase would be performed by the Context Abstractor or Context Source, as was discussed earlier.

The Naïve Bayes classifier (Pearl 1988) was chosen for the study. Analysed against the requirements, it has the following good qualities:

- It is computationally very efficient and thus suitable for online processing in a mobile device. Training and inference both have a linear complexity in the size of input data.
- Low computational complexity of inference allows the use of a large input space without significant performance degradation.
- It is robust in the presence of incomplete information.
- It can use context data described by the ontology as an input, a vector of context atom confidence values. Fuzzy membership values can be applied as virtual evidence.

- It requires no background information modelling, except for choosing the relevant inputs for each network. For instance, based on background knowledge, it is not rational to try to infer the type of ambient music from the context atoms describing the stability of the device, even though it might happen to be a discriminating factor based on the data set.

- Multiple simultaneous contexts of different types can be modelled with multiple parallel networks.

- It is possible to modify the models by updating the conditional probabilities of the network. Since the computational complexity of learning is low, it can be performed online.

- New context types can be modelled with additional networks.

Two separate networks were used in the case study, one for the context type Environment:Audio:Type, and one for Location:IndoorsOutdoors. The environment audio type had the context values RockMusic, ClassicalMusic, Speech, Car, Elevator, TapWater and OtherSound; IndoorOutdoor location had the context values Inside and Outside.

The recognition of the higher level context Indoors from a set of context atoms is illustrated in Figure 15. The Indoor/Outdoor Bayesian network is used for classifying the context atom confidence value vector describing a one-second situation instance. Each context atom (input) is associated with a conditional probability, which indicates the probability of the input given the output. Conditional probabilities have been learned from the training data. During the classification, the Bayes theorem (Pearl 1988) is applied for calculating the output class, given the current values of the inputs and the conditional probabilities. The audio classification utilises the same principle.

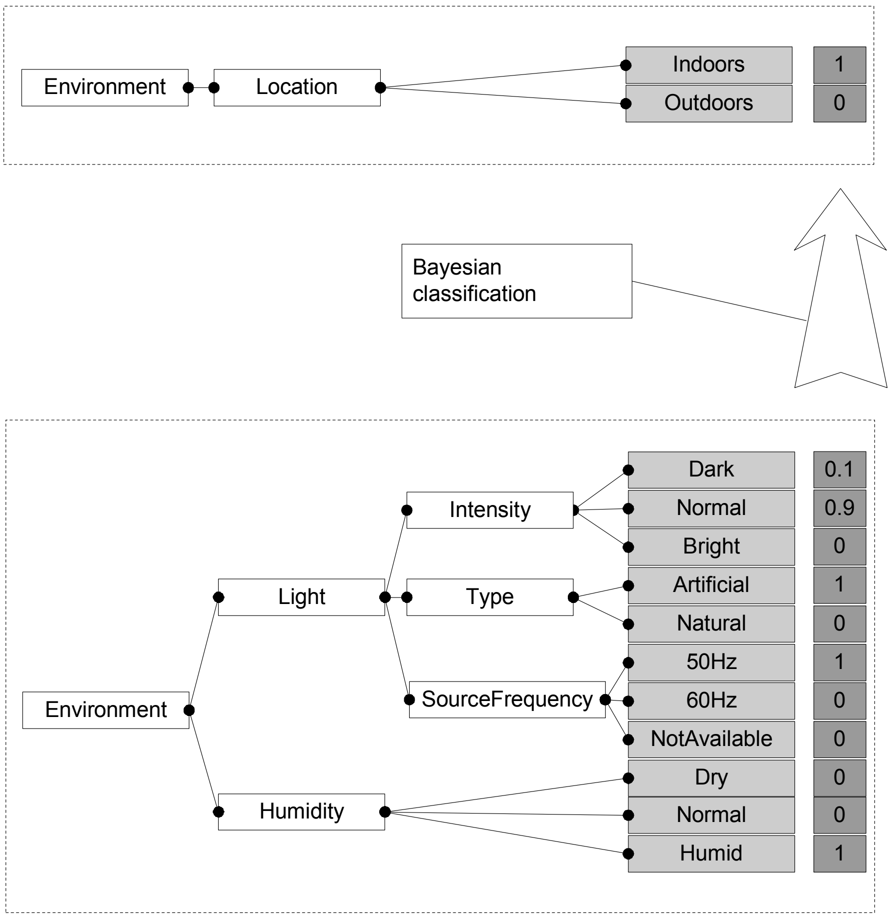


Figure 15. The white rectangular boxes represent the context types for the context values that are represented by the light grey boxes. The dark grey boxes contain the corresponding confidence instance values for the current situation. The Naïve Bayesian network can be used to classify the confidence instance values into one of the previously defined output classes.

6.4.3 Results of the case study

In the case study (Korpipää et al. 2003a), two naïve Bayesian networks were applied to classify the contexts of a mobile device user in their daily activities. The research problem was to recognise nine contexts (Speech, RockMusic, ClassicalMusic, Car, Elevator, TapWater, OtherSound, Indoors, Outdoors) measured from a continuous real-life scenario. Four contexts (Activity, Still, Walking, Running) were abstracted in the feature extraction phase, without classification. Hence the total number of abstracted contexts was 13. Multiple contexts could exist simultaneously in the scenario. The measurement system hardware consisted of a small sensor box (Tuulari 2000) attached to the shoulder strap of a backpack containing a laptop. The user carried the backpack when collecting the scenario data.

Figure 16 presents the classification results for one scenario of nine, showing the recognised context values as a function of time.

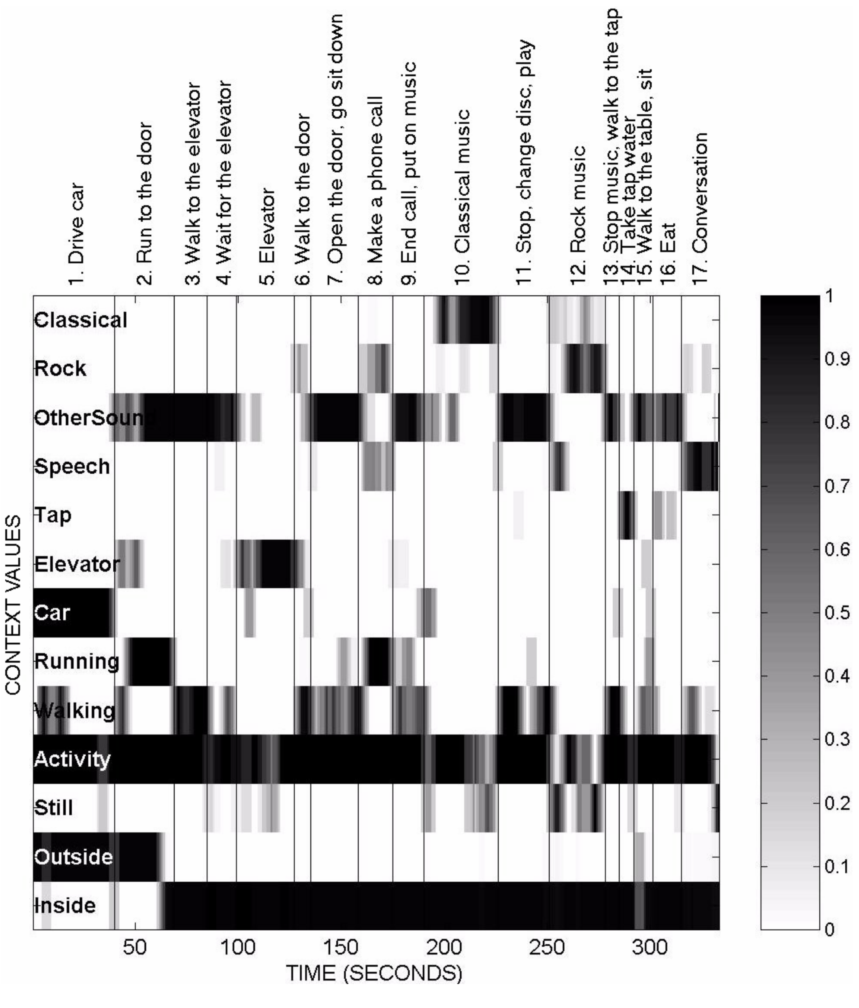


Figure 16. Recognised context values as a function of time in a scenario. The X-axis is time, the Y-axis context, and the grey scale intensity is the probability of context at a time instant.

Table 12 presents the results averaged over all nine classified contexts, and over nine scenarios. The controlled condition data (first row) were measured separately from the scenario data in as ideal conditions as possible. Since the data for each context were measured separately, these data are free of the disturbances that were present while measuring the actual continuous scenario data (the results on rows 2 and 3). Hence the controlled conditions experiment predictably yielded almost 100% accuracy in true positives and true negatives. This can be viewed as maximal accuracy in an optimal setup. A four percent error in true positives suggests that the data contain some similarities that cannot be discriminated with the features. In the real-world situation, even if training and test data are the same, the recognition accuracy falls below 90% (row 2). The main reasons for the decrease are, first, coarseness of annotation, as it is not possible to label the correct answers precisely, and, second, the undefined action that takes place each time a class segment changes. Extra events often occur randomly when they should not, since the real-world situation cannot be controlled. About eight percent of the error in true positives is explained by these factors. The results in the second row of Table 12 are calculated by using each individual data set as training data while the same data set is used as the test data. Hence the second row results can be viewed as a reference accuracy of the classifier in the specified setup with the real-world scenario data. The actual classification results shown in the third row of Table 12 must be viewed in comparison with the results in the second row. The actual performance (row 3) was measured using cross-validation by leaving out in turn one of the nine data sets of the training data set being used as a test data set. Comparison of rows two and three clearly shows that the Naïve Bayesian network models the data satisfactorily in this setup, and the amount of training data has been sufficient.

Table 12. Three stages of classification accuracy results for the nine contexts classified using two Naïve Bayesian networks. The actual real-world scenario results (row 3) can be compared with the maximum scenario results (row 2).

	True positive %	**True negative %**
1. Controlled conditions, test data same as training data	96	100
2. Scenario data, test data same as the training data	88	95
3. Scenario data, leave-one-out cross-validation	87	95

The recognition (and abstracting) results for each of the 13 individual contexts, averaged over nine scenarios, are shown in Table 13. The last row shows the system average, which accordingly includes both the Bayes-classified contexts and the feature-extracted contexts. Some important non-audio contexts, such as Inside and Outside, are recognized well in the scenario. The recognition of Walking is based on detecting the frequency and intensity of vertical movement, and the placement in a trouser pocket or on a belt, not to mention feet, would have been better than a shoulder strap. Another factor that reduces Walking detection accuracy is the coarseness of manual segmentation. For instance, the scenario segments 9 and 11, where the task to walk to the CD player, change the disc and walk back to the sofa is labeled merely as Walking, which is not true while you change the disc.

The data was collected by five persons. The results are thus more generalisable than if the training data and test data had both been given by the same person. User-dependent training would probably enhance the results for contexts that vary a great deal among different users, such as Speech. This was not tested though. In this experiment the recognition of Speech can be considered speaker-independent, and thus the accuracy of 91% is good. Concerning the acceleration-based contexts Walking and Running, some testees produced worse results due to their style of movement containing negligible vertical accelerations of the upper body. Context Activity is designed to indicate any movement of the device. Hence, while the user state is Walking or Running, there is also Activity. Still is the opposite of Activity, and should be on when there is no movement of any kind. Some people are more active than others, so that while for some of the testees the context Still is in the correct segments, other testees are active during the whole scenario. In this sense, the recognition of contexts Still and Activity is correct with respect to the situation, but, as some people tend to move while sitting on the sofa and listening to music, although annotated Still, the actual numbers are much worse. This is partly a problem of scenario design, which should have contained a segment where the device is put on the table, for instance, so that the annotation could be set to Still and the real situation would certainly have been the same.

Table 13. Recognition results for each of the 13 contexts averaged over 9 scenarios. The last row shows the system average over all the 13 contexts, of which 9 are classified and 4 are direct.

	True positive %	**True negative %**
Classical music	80	99
Rock music	68	98
Other Sound	91	94
Speech	91	97
Tap water	91	100
Elevator	92	97
Car	100	100
Running	62	95
Walking	72	83
Activity	100	40
Still	88	92
Outside	69	100
Inside	100	69
System average	**85**	**90**

In the contexts that relied on audio features, the best results were achieved in recognizing Car, Tap water and Elevator. For those contexts there were mainly one or two features that separated them quite clearly from the other contexts. In addition, those audio signals were continuous in time, unlike, for example, a speech signal. The recognition of Classical music, Rock and Speech signals suffers from the strong variation between consecutive analysis windows, as well as from the variation between the training and testing data. One of the problems in the audio feature design was deciding the length of the time window. The features were used as an input for the Bayesian classifier using a one-second interval; thus if the features were calculated using a longer window, delays were evident in the recognition, thereby reducing the accuracy. The duration of the contexts in the scenario data was short, from only a few seconds up to 30 seconds. Even for humans, the average recognition time for some everyday auditory scenes is 20 seconds (Peltonen et al. 2001).

The measurement setup was natural. Sources of confusion include birds singing outside, clinking of keys, the sound of the assistant's footsteps, sounds made by taking the CD out of its case, command replies of the measurement system, etc. The context transitions are not as clear and immediate as they are annotated. For example, after driving a car one does not directly start running; one turns off the car engine, undoes the seatbelt, opens the door, gets out, closes the car door, and then starts jogging. Labeling merely incorporates driving the car and running. Therefore, some of the seemingly odd mistakes in the recognition are actually correct if one looks at the raw data, but they still have to be treated as errors in accuracy calculation. These artefacts cause more relative error in a short scenario than in the longer ones.

6.4.4 Discussion

The goal of the case study was to expand the collection of generally recognizable constituents of context, where personal mobile device usage is concerned. Even though the scenario setup was limited, some conclusions can be made about the genericity of recognition. It is quite obvious that most of the contexts recognized in the scenario are likely to be valid only locally, since most features potentially refer to many possible real-world situations. Progress towards solving ambiguities requires more and more finely grained information at the lower levels of the context hierarchy. Background knowledge was only used in the network structure setup for dividing the classifier into two networks with selected inputs, based on the underlying sensor types, and for the quantization limits. More specific background knowledge modelling can be used to solve ambiguities within a restricted scenario. However, over-specific background knowledge modelling causes the loss of generality of the classification system as more and more contexts become context-dependent instead of being generic descriptors of the environment, such as MPEG-7-related features are for the audio data. In addition to ambiguity, the traditional machine learning problem of generalizing beyond training data (Mitchell 1997) may emerge, although the scenario in this study was limited enough to control that aspect. Failure in generalization may cause difficulties in contexts that are complex at the feature level or contain a lot of variation, such as Speech, which tends to be user-specific.

Bayesian networks are suitable for classifying incomplete information and learning conditional probabilities is straightforward, requiring relatively little training data. Naïve Bayes networks are computationally very efficient, and thus feasible for real-time recognition. In this experiment the Naïve Bayesian classifier almost reaches the reference accuracy, which indicates that there were enough training data to explain the variability introduced in the data on purpose. Even though the Naïve Bayes independence assumption is violated, the classifier still performed well. Concerning time, the classification is performed independent of previous contexts; each second is treated individually. It is evident that the order of events in time provides additional context information, which was not exploited in this experiment. However, although the order of events may help gain better results within a restricted scenario, it is not likely that those results can be generalised well beyond the training examples, knowing that the order of events in the real world varies significantly.

Although the case study is long-term context-awareness research, some of the contexts are recognized reliably enough to suggest even near-term applicability. An example of such a context is Car, which was recognized with an accuracy of 100% both in true positives and negatives. However, even simple applications require prior study of how, for example, different clothing affects the recognition, since in most practical situations the mobile device is placed under clothing. Moreover, in mobile devices the battery power consumption from the continuous audio-based monitoring of the environment poses a big problem for practical utilisation.

6.5 Summary

The context framework supports abstracting context information from a continous sensor data flow into abstracted event-based communication. For utilisation in applications, the abstracted context information is represented in a uniform manner, as defined in the ontology.

The requirements for context recognition methods suitable for use in mobile devices were specified. For the case study, the selected classification method was analysed against the requirements. The case study evaluated recognition of multiple simultaneous contexts from multiple sensor sources. The feasibility of continuous (not pre-segmented) recognition of contexts from real-world sensor data was evaluated quantitatively and compared with a reference classification measured in controlled conditions.

The main contributions of this chapter are a requirements analysis for context recognition methods for use in mobile devices, a conceptual model for the transformation of continuous sensor data flow into abstracted context change events within a blackboard-based mobile device context framework, an experiment and results for the recognition of multiple simultaneous contexts from multiple sensor sources in a mobile device user case study, and a quantitative evaluation of the feasibility of continuous multi-action context recognition.

7. Context Manager API

Previous chapters have introduced the context framework elements, and the ontology for representing the abstracted information that is used through the framework. For an application, or the Application Controller, that uses the Context Manager, the activity in the other framework elements is transparent. The Context Manager provides a single point interface to context information for clients. To summarise, the Context Manager provides a 'publish and subscribe' mechanism and a database. This chapter introduces with examples the application programming interface (API) provided by the Context Manager. The API was partly published by Korpipää et al. (2003b).

7.1 Adding context

Any client may add contexts to the Context Manager blackboard. Each individual add-message is allowed to contain one context. An alternative would be to collect a set of contexts before sending them. The latter option would be more efficient if the collected contexts could be sent at suitable time intervals, but it would complicate the handling of error situations and would be unsuitable if different time resolutions were required for different context types.

Context objects are used for encapsulating the context instance. In the case of device internal context add, the client (Context Source, abstractor or application) must fill the context object, which contains the properties defined in the ontology structure. Properties context type, context value and source are obligatory, the others are optional. After setting the property values the client can send the AddContext message, which contains the context object. The AddContext function has parameters determining whether the context instance is stored into the permanent database or not, and whether change detection is performed. Table 14 presents two example contexts to add to the blackboard. The contexts are instances from the ontology vocabulary. The Context Manager returns a value according to the success of the operation. Only context type and context value properties and method name are presented for clarity.

Table 14. Context add examples.

Purpose	**Method name**	**Context type**	**Context value**
Add context	AddContext	Environment:Light:Intensity	Bright
Add context	AddContext	Device:Movement:Activity	Activity

7.2 Requests and responses

The client may request context information directly from the Context Manager, which contains a relational context database. The application developer only needs to know the type of context from the ontology, and the method name to make a request (Table 15). Even if the context is abstracted from multiple inputs, the application developer can request it by the context type defined in the ontology vocabulary. The client may also request context information by the sub-branch of the context ontology vocabulary hierarchy tree (example number 2 in Table 15), and in this case the response will contain all the context objects that were found for all the context types of the sub-branch. Requesting by sub-branch makes it easier for the client to get information from multiple subcategories without separately requesting individual context types. Tables 15 and 16 contain a few examples of requests and corresponding responses.

Additionally, queries can be made with the context source as a key. The requests may contain context type and source or either of them. The examples represent only contexts from device internal sources, and thus the source property is omitted. Only context type and context value properties and method name are presented for clarity.

Table 15. Examples of client requests to Context Manager.

Example number	**Purpose of the function**	**Request name**	**Context type**
1.	Request context with a full context type from vocabulary	RequestContext	Environment: Light:Intensity
2.	Request contexts with a partial context type (sub-branch) from the vocabulary	RequestContext	Environment: Light
3.	Request a set of contexts (set consists of one or more context types)	RequestContextSet	{Environment: Humidity, Device: Orientation}
4.	Request contexts of a specified time interval for a context type	RequestContextsOf TimeInterval	Environment: Temperature
5.	Request a number of latest contexts for a context type	RequestLatestN Contexts	Device: Movement: Activity

Table 16. Example responses from the Context Manager for the requests in the Table 15.

Example number	**Context type**	**Context value**
1.	Environment:Light:Intensity	Bright
2.	{Environment:Light:Intensity, Environment:Light:Type, Environment:Light:SourceFrequency}	{Bright, Natural, NotAvailable}
3.	{Environment:Humidity, Device:Orientation}	{Dry, UpsideUp}
4.	Environment: Temperature	{Normal, Cold}
5.	Device:Movement:Activity	{Still, Activity, Still}

In example three, requesting a set of contexts corresponds to performing RequestContext for all the context types in the set. The context set request exists to enable getting all the required contexts by using one command instead of many. In example four, contexts that occurred for a certain type within the specified time interval are returned for the request. In example five, a specified number of latest contexts, starting backwards from current, are returned for the request. Set and time interval requests can only be performed with a full context type; sub-branches can only be used with the RequestContext method.

7.3 Subscriptions and indications

The subscription-indication mechanism delivers the required context information to the clients in an event-based manner. The clients subscribe to context change notifications – the clients essentially tell the Context Manager, "when something about this happens, let me know."

Subscribing to the full context type (path) specified in the vocabulary enables notification about a single context value upon change. Subscribing to a partial context type (sub-branch) enables notification about all context values that are under the specified branch, whenever any of the context values change. Tables 17 and 18 show examples of subscriptions for the contexts specified in the ontology vocabulary examples, and the corresponding indications.

Subscriptions can also be made with the context source as a key. The subscriptions may contain context type and source or either of them. The use of source is not included in the following examples.

Table 17. Examples of context subscriptions.

Example number	**Purpose**	**Subscription name**	**Context type**	**Context value**
1.	Subscribe to context with a full context type from vocabulary	ContextChangeSubscription	Device: Placement	-
2.	Subscribe to contexts with a partial context type from the vocabulary	ContextChangeSubscription	Environment: Sound	-
3.	Subscribe to a numerical context value	ContextChangeSubscription	Environment: Temperature: Absolute	-
4.	Subscribe to context start	ContextStart Subscription	Location: Facility	Movie Theatre
5.	Subscribe to context end	ContextEnd Subscription	Location: Facility	Movie Theatre
6.	Subscribe to a set of contexts (set consists of one or more context types)	ContextSet Subscription	{Environment: Sound: Intensity, Environment: Temperature, Device: Movement: Activity}	-

Table 18. Example indications upon context change for the subscriptions in the Table 17.

Example number	**Purpose**	**Context type**	**Context value**
1.	Indication about a change in context	Device:Placement	AtHand
2.	Indication about a change in contexts under the subscribed branch	{Environment:Sound:Type, Environment:Sound: Intensity}	{Car, Loud}
3.	Indication about change in the numerical value of context	Environment:Temperature: Absolute	20
4.	Indication about the start of a context value	Location:Facility	Movie Theatre
5.	Indication about the end of a context value	Location:Facility	Movie Theatre
6.	Indication about a change in any of the contexts in the subscribed set	{Environment:Sound: Intensity, Environment: Temperature, Device: Movement:Activity}	{Loud, Warm, Still}

The first example notifies the client when the device placement changes. With a subscription to a partial context type, example 2, the client will be notified when any context value of the context types under the branch change. This will simplify the subscription to a category of contexts, which contains many context types. For example, the client could subscribe to device-category contexts with one subscription, instead of separately listing all the device-related context types. Subscribing to a numerical context value should only be used for context types that have been properly treated for change by the Context Source, since indication is given every time the value of the context changes. Subscribing to the start and end of a certain context value is especially useful for context types that have a large set of symbolic values, such as location. For instance, an application might only need the context value MovieTheatre to change the device profile to silent upon arrival and back to previous after leaving. Subscribing to the context value start and end relieves the application from receiving unnecessary indications about all changes in the context type. The application will only get the relevant messages.

The context set subscription is designed for subscribing to any set of context types, which are not necessarily under the same branch in the ontology. The client is informed whenever any of the contexts in the set change. Subscribing to a set corresponds to making several single context subscriptions.

Context abstracting is transparent to the client in the subscription mechanism. The client can subscribe to a higher level context as to any context. The abstractor has subscribed to a set of context types, and whenever any of them change the abstractor executes. If the resulting higher level context has changed from the previous one, the Context Manager indicates this to the subscribed client. Hence the framework and the API offer a solution to acquiring context information from multiple source sensory data, abstracting and recognising contexts from uncertain and imprecise data, and representing the abstracted data with an ontology up to the transparent use of the abstracted context data through the compact API.

7.4 Summary

An overview of the application programming interface provided by the blackboard-based Context Manager was given in this chapter. The API describes how context information can be utilised by, e.g., context-aware applications or an Application Controller through the framework. The number of different API functions is nine, independent of the number and type of Context Sources and other 'hot spot' elements of the framework. For comparison, in the widget model the client directly interacts with multiple and distributed components, each of which has different functions, and may have different addresses, which the client has to know. In the Context Toolkit the result is a complex API with a total of over 50 different messages (Dey 2000), which are dependent on the number and type of widgets. The blackboard-based model offers the advantage of hiding all the other framework components except the blackboard manager from the client. The client will access all context data from any source from the same central node, which simplifies the application programming interface. The client can access the context data simply with context type, and with or without specifying the source of context.

Hence the main contribution of this chapter is a compact API, which is uniform for all context producers and consumers, for providing and using rapidly changing sensor data as abstracted context objects in a mobile device. Published partly by Korpipää et al. (2003b), it is the first blackboard-based API for handling context-related information in mobile devices.

8. End-user development of context-aware applications

This chapter is partly based on the articles by Korpipää et al. (2004a, 2005a, 2005b). The chapter, for the minor part, reuses the results by Korpipää et al. (2004a, 2005b), and, for the major part, by Korpipää et al. (2005a). A few additional details are included.

The hypothesis, that the context framework enables quick development of context aware applications, was set in the introduction. The Context Manager API itself offers a compact uniform programming abstraction that simplifies the development of context aware applications compared with the related work. However, there are no well-established practices for evaluating the usability of a software programming interface. A step further is taken by providing another programming abstraction, which can be evaluated with standard usability evaluation practices. The purpose of the programming abstraction is to enable the end-user development, or customization, of context-aware applications in a mobile device. This chapter explains how the implemented context framework and ontology are utilised for enabling the use of context-aware features in a mobile device, defined with an implemented customization tool.

8.1 Customizer

Customization is a form of end-user development (Fischer et al. 2004). The concept of context-aware application personalisation, and a tool named Context Studio, was originally introduced by Mäntyjärvi et al. (2003). Korpipää et al. (2004a) developed the concept into a customization tool for small-screen mobile devices, and introduced a method for automatically generating graphical UI views based on context ontology. Furthermore, the concept was modified by Korpipää et al. (2004a); the user separately chooses one or more individual conditions – i.e., context type – value pairs for an action instead of choosing one context value for all the available context types to describe the situation for each action. This results in much simpler and more controllable rules that are faster to define. Korpipää et al. (2005a, 2005b) enabled utilising the context framework for the actual use of the features defined with the tool, and included explicit control commands as inputs. The tool is still named Context Studio.

The idea of customization is that instead of implementing context-aware features at design time, a set of available contexts and actions are provided for the user, who decides whether and how to use them. The user performs customization by specifying rules with a graphical user interface. Rules connect contexts to actions, and after the rules are activated in a mobile device the context events correspondingly trigger actions. The rule condition part element is called a (context) trigger, which can be any event or state that can be used for activating an action. The trigger can be either an implicit input or a direct control command given by the user with any available modality. The available triggers, which consist of context type – context value pairs, are defined in a context ontology vocabulary according to the vocabulary model. An action is any application, function or event that can be activated when a set of triggers are fulfilled. An action can also belong to an external device, which thus enables the user to customize how to control external devices with a mobile device. A rule is an expression connecting a set of triggers to an action. The formal Context Exchange Protocol (CEP) syntax was utilised for representing rules generated from the graphical descriptions with the tool.

8.2 Utilising context framework

From the context management viewpoint, the Customizer is a graphical editor for generating and reading context-action rules. Figure 17 shows an overview of an instantiation of the context framework, implemented on the Symbian platform (Digia 2003).

After the user has created the desired context-action behaviour with the Customizer, the context framework handles the background monitoring of contexts and the triggering of actions according to the rules. The Application Controller facilitates the application control inference on behalf of the user or application. The framework separates context management from application code, and no changes need to be made to existing applications when they are augmented with context-aware features.

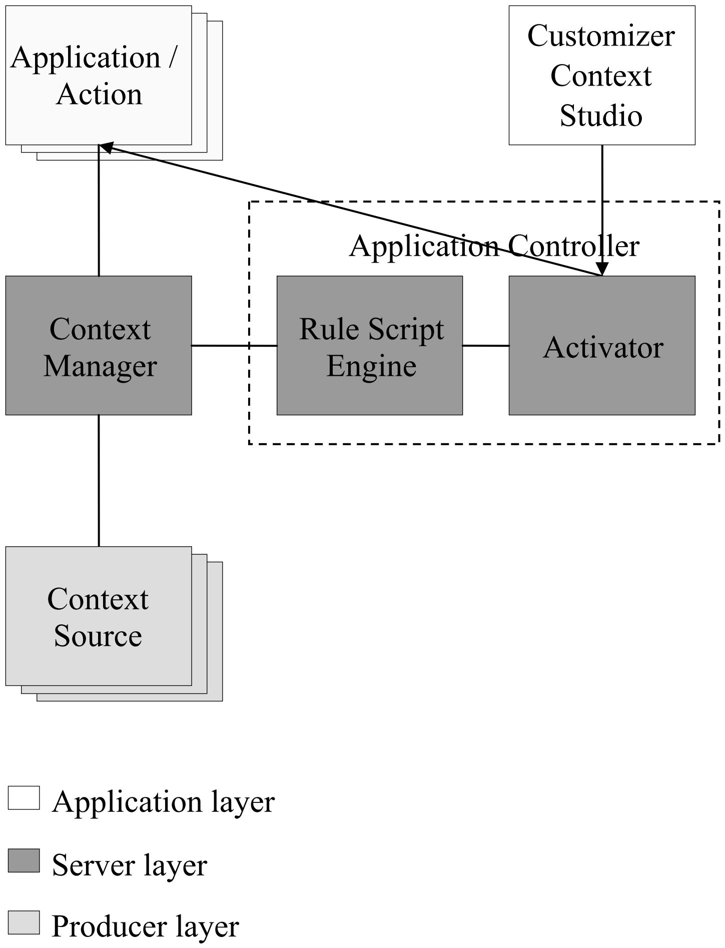


Figure 17. An overview of the implemented context framework with a rule-based Application Controller.

The implemented Application Controller consists of two parts: Rule Script Engine (Lakkala 2003b) and Activator. Hence, as an instance of the Application Controller entity, a rule-based application control inference approach was chosen. Other control inference approaches could be used as well. A rule-based inference does not incorporate mechanisms for handling uncertainty. Uncertainty is eliminated by the producer layer components (Figure 17). Sensor signal uncertainty is thus not transmitted up to the Application Controller and to the user.

The rules created by the user and converted by Customizer to CEP scripts are evaluated by Rule Script Engine, which uses the Context Manager. The rules that the user has created as active are subscribed by the Activator component to the Rule Script Engine. The Context Manager receives all the context information from Context Sources, and indicates changes in those contexts that are used in the active rule scripts to the Rule Script Engine. The Rule Script Engine evaluates the rules for the changed contexts, and if the conditions are fulfilled, indicates triggered rules to the Activator. For triggered rules, the Activator launches the designated application functions or system events.

In the context framework instantiation of Figure 17, change detection is performed by the Context Manager. A separate Change Detector was not required for the applications involved. Context abstracting tasks are handled by the Context Sources, and thus separate Context Abstractors were not necessary for the tasks addressed.

Several Context Sources were implemented for providing inputs for the interaction customization. The implemented Context Sources include accelerometer-based freely trainable gestures and other movement abstractions such as activity level and orientation, physical selection with RFID tags, cellular network-based location, time, Bluetooth devices, and several events from the device platform, such as keyboard, display, battery strength, network strength, charger, profiles, foreground application and keypad lock. Several application actions were implemented for the availability of customizing the interaction. The application actions and system events include call, messaging, camera, profiles, browser, display, keypad, joystick functions, etc., and the external device actions include a set of observation camera functions and image transfer.

The context framework enables the use of multiple modalities for controlling mobile devices, as defined by the user with the Customizer. In addition to implicit inputs for context-aware applications, the framework enables utilising explicit control commands, such as gestures and physical selection. Table 19 presents examples of implicit and explicit control tasks. All input events are managed as context instances within the framework.

Table 19. Examples of direct and indirect control tasks managed by the context framework.

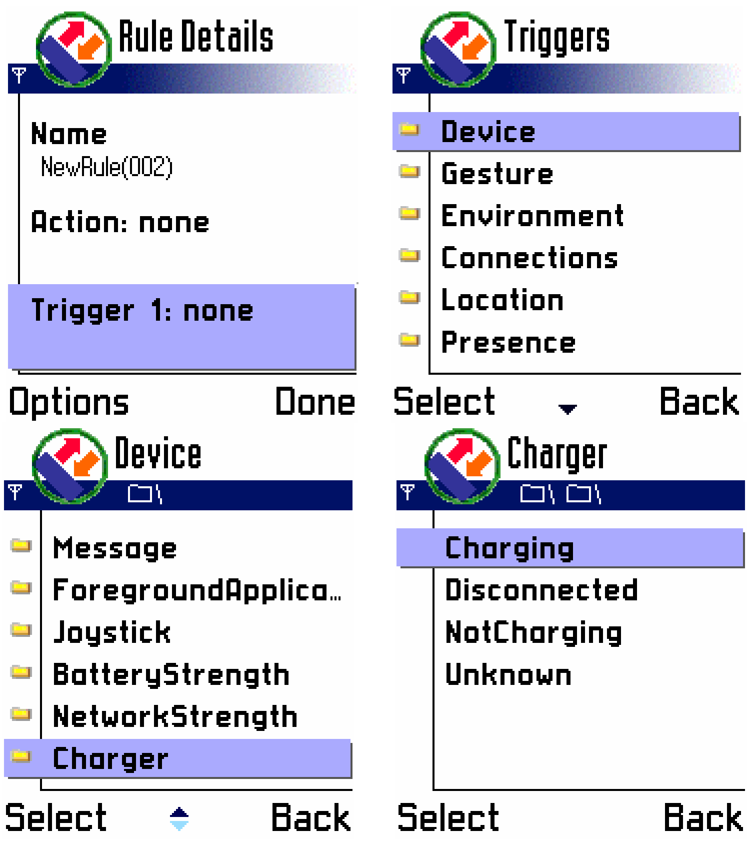
User interaction	**Application category**	**Application**	**Context Source**
Explicit	Movement-based action	Gesture control	Accelerometer
Explicit	Selection-based action	Tangible interface call	RFID tags
Explicit	Selection-based action	Observation camera control	RFID tags
Implicit	Proximity-based action	Social context-based profile change	Bluetooth devices
Implicit	Movement-based action	Activity-based display light switcher	Accelerometer

8.3 Utilising context ontology

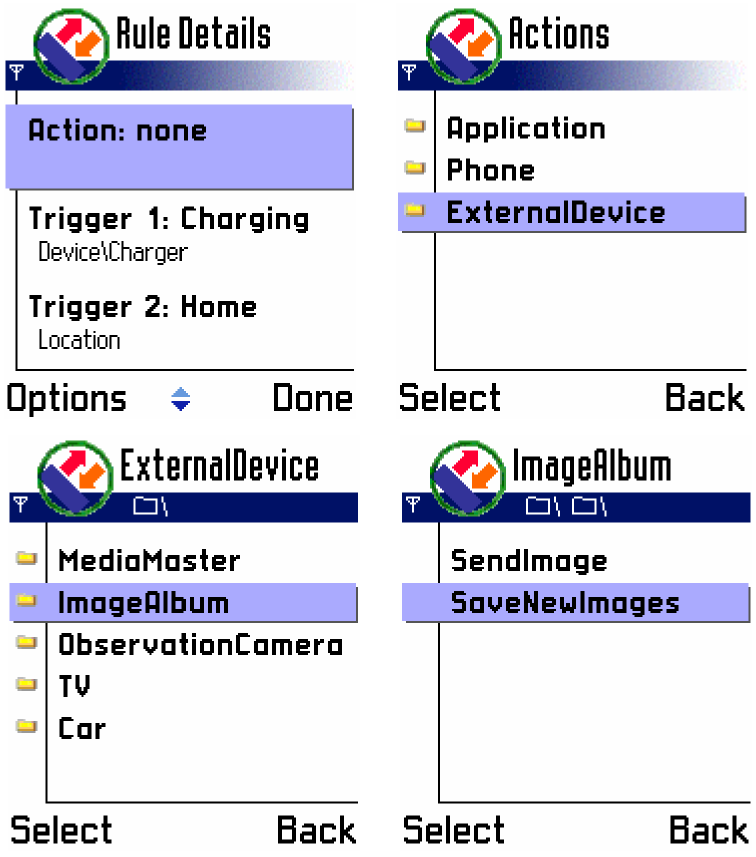
The context ontology is the uniform human-understandable and machine-readable representation within the framework. The representation enables the end-user to connect contexts to actions with the Customizer. Different types of incoming sensor signals and events (the inputs) are abstracted by the Context Sources into the uniform representation. Actions in the action vocabulary describe the various application functions, terminal events (the outputs) and external device actions.

The vocabularies describe any implicit or explicit input events. Furthermore, the vocabularies can be dynamic – i.e., they can be changed at runtime, and even personalised by the end-user. For example, the user can train and name the gestures s/he wants to use. The current implementation reads the vocabularies when the tool is started. By modifying their content, the Customizer itself can be customized. The runtime updating of the Customizer UI is further work.

The Trigger, Action, and Rule views in the user interface are generated based on the ontology vocabularies and rule models. The context type hierarchy is transformed into a folder-file representation in the UI. The context and action type concepts are represented as folders, according to the vocabulary hierarchy. The context and action values correspond to files in the UI representation. The vocabulary model facilitates straightforward updating and modification of the contexts and actions into the UI. When new context and action types appear, they can be presented as new paths in the UI, and new context and action values are presented as new files in the folders. Selecting the rule elements resembles navigating a directory tree hierarchy. The screenshots in Figure 18 show the UI navigation, starting from left to right, during rule creation in the implemented Series 60 (Series 60 2005, Digia 2003) style Customizer.



a)


Rule Details

Action: none

Trigger 1: Charging
Device\Charger

Trigger 2: Home
Location

Options Done

Actions

Application
Phone
ExternalDevice

Select Back

ExternalDevice

MediaMaster
ImageAlbum
ObservationCamera
TV
Car

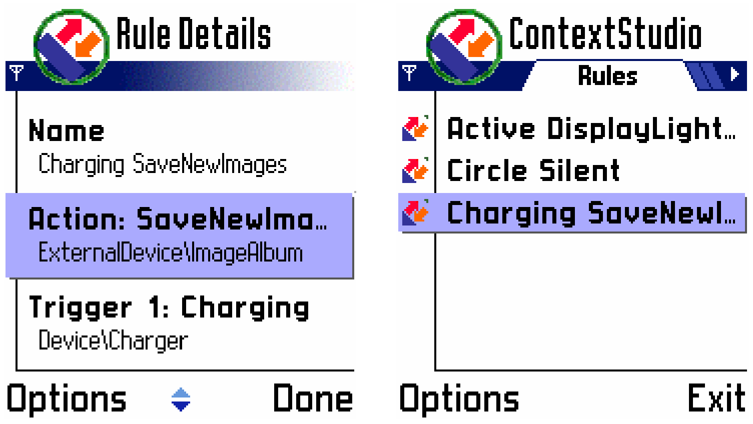
Select Back

ImageAlbum

SendImage
SaveNewImages

Select Back

b)


Rule Details

Name
Charging SaveNewImages

Action: SaveNewIma…
ExternalDevice\ImageAlbum

Trigger 1: Charging
Device\Charger

Options Done

ContextStudio
Rules

Active DisplayLight…
Circle Silent
Charging SaveNewI…

Options Exit

c)

Figure 18. An example of Customizer ontology-based UI navigation during rule creation.

In Figure 18a the user selects a context trigger for the rule by navigating through the Context type Device:Charger and selects the Context value Charging. Similarly the user selects the second trigger, Context type Location and Context value Home. In Figure 18b the user selects an action for the rule by navigating through Action type ExternalDevice:ImageAlbum and selects Action value SaveNewImages. The first screenshot in Figure 18c shows the complete rule after the user has selected the elements; the second trigger can be viewed by scrolling down. The rule name is generated accordingly. After the user selects the option Done, the rule CEP script is generated and the rule is activated and functional in the context framework. The second screenshot in Figure 18c shows the main rule view with the list of active rules. When the rule conditions are met, the context framework automatically performs the action.

8.4 Customized example applications

Customizable multimodal interaction facilitates personalized and potentially more efficient interaction with the device. The new modalities can be used as, e.g., shortcuts or “soft keys”, which will reduce the click distance and user effort in interaction. The click distance is the number of user operations required for performing a certain task. To call a person, for instance, the usual average click distance is five or more (if a specific button shortcut is not used), depending on the phone model, keypad lock state and position of the name in the phonebook. When the user makes the call by touching an RFID tag, the click distance is one. As an example usage scenario, elderly people could more easily contact their relatives with a customized phone by touching a picture containing a tag with the device. The phone could have been customized by a relative, for instance. With the tag read/write accessory, RFID tags can be written to contain any context, which can be connected with Context Studio to any available action. In the future, the RFID tag accessory will be embedded in the phone.

Gestures can also be used to reduce click distance. Gestures here refer to hand movements made by the user with a phone containing acceleration sensors. Using HMMs to model the acceleration-based gestures allows the user to train and use gestures of any form, and gestures can be exchanged among users or provided by a third party. Discrete gesture commands can be used as shortcuts for opening applications, such as messaging, camera, calendar, and opening bookmarks in a browser, sending messages, etc. Profiles and keypad lock can also easily be controlled with gestures.

A simple example of a customized implicit control task is a rule that reads as: if device orientation is display up, and device is active, turn display light on. Activity and orientation can be detected from the accelerometers in the smart phone. Another example is a rule stating that when the location is home and the device charger is charging, save new images into the image album through a Bluetooth connection. Coarse location can be detected from the cellular network IDs, and the charger Context Source indicates the changes in the charger status to the Context Manager.

The main application focus in the dissertation was the customization of the emerging interaction modalities for controlling the applications in the smart phone. Concerning external devices, a demonstrated example is controlling an observation camera that has an SMS (Short Message Service) message command interface. Normally, in order to have the camera take an image and return it to the phone, the user would have to write an SMS message containing a command and send it to the camera. For the same task, the user can, for example, customize an RFID tag-based action ExternalDevice\Observation Camera TakePicture. When the user then touches the corresponding RFID tag with the tag reader, the Activator sends a corresponding SMS message command to the camera. This enables more efficient user interaction for controlling the observation camera.

In a more general external device control setting, the Universal Plug and Play (UPnP 2005) framework can be used for controlling devices such as home appliances with a smart phone. Home devices are connected as UPnP devices to a UPnP control point, which performs a service search for appropriate home appliance functions. The smart phone acts as a control point, which receives available actions from the home appliances and executes commands to the appliances through the UPnP protocol. The control points have an IP connection to the UPnP server through a Bluetooth connection. When used with the context framework, the Activator would connect to the phone's UPnP control point interface for performing the actions triggered by the context events. As an example application, the user could turn on a TV set by making a gesture with the phone. This was not demonstrated in this dissertation.

8.5 Discussion

The framework was tested with over twenty different rules. Multiple rules can be functional simultaneously, and no delays were noticeable in the framework operation. When multiple actions are defined for the same trigger, the rules are executed in the order of creation. If two rules are conflicting – e.g., the same trigger in two separate rules causes the keypad to lock and unlock – they are nevertheless executed in the order of creation. Automated conflict prevention would reduce the flexibility of the system and increase the complexity, since it would require defining all the conflicting actions for each action. One option would be to let the user select from the actions that are defined for the same trigger in the interaction situation. This is not feasible when the actions are required to be automatic after defining the rules, and system interruptions would disturb the interaction. Moreover, the user may want to define multiple actions for the same trigger. Yet another solution is to let the user prioritize the rules. However, it would still be the responsibility of the user to define feasible rules. Possible deadlock situations can be avoided by not allowing actions to function as triggers.

An important challenge in end-user development systems is providing the user with an experience of control. In other words, the system functionality should exactly match the functionality that the user wanted to describe. Some approaches have been proposed. Dey et al. (2004) proposed modelling contexts based on examples, which is feasible when the example contains a single, chosen type of context. When one example contains multiple types of contexts, the programming-by-demonstration approach may lead to functionality that the user did not intend to have, if the user cannot control exactly which contexts are relevant for the intended action. Truong et al. (2004) proposed providing a set of words that the user can arrange for a description, which the system then translates into functionality. The descriptions the users create by freely combining words can be ambiguous and do not always result in the intended functionality.

In the rule-based approach presented in this study, the user defines each condition and action in the rule individually and explicitly with the type-value pairs. The approach yields satisfactory usability and user control, backed up by the usability evaluation results (Korpipää et al. 2005a, Häkkilä et al. 2005).

According to the user tests, the idea of defining device functionality with rules was very well understood, and all users were able to create the correct scenario functionality. Succeeding in this is very much due to an earlier observation that users prefer very simple rules in customization (Korpipää et al. 2004b). The rule structure was thus designed to be reduced, incorporating only the logical operator AND. The logical operator OR is available by creating parallel rules.

The general goal of end-user development was to achieve a low cost of learning while having a wide scope of customizable functionality. The usability evaluation results confirm the low cost of learning to use the tool (Korpipää et al. 2005a). Concerning scope, the rule-based customization approach applies best to actions that operate on single discrete commands. Continuous control tasks, such as increasing volume, or keyboard-intensive tasks, such as text input, are not as well suited to customization with the chosen approach. Sequences of tasks can be realised by making multiple rules for the same trigger. The rule expressiveness in customization has a trade-off with tool usability.

The customization tool can be used for personalising the mobile device functionality, interaction and multimodal interaction. The CEP script created with the tool can be considered a user profile that defines the user preferences of the device functionality based on any input event.

8.6 Summary

An end-user tool was presented for customizing human-computer interaction with a mobile device. The interaction is customized by defining context-action rules with a graphical user interface. The context can be any event or state that can be used to activate an action, including explicit control commands given by the user, in addition to inputs for implicit interaction. Actions refer to a set of available phone functions, and its applications and appliances in the near or remote environment. The tool facilitates customizing multimodal interaction. Customizable interaction modalities include explicit movement sensor-based freely trainable gestures and physical selection with near-field radio frequency tags, in addition to the implicit inputs from sensors, phone platform and Bluetooth devices. The blackboard-based context framework enables the application control based on the features defined with the tool. The user interface views of the tool are generated based on context ontology vocabularies. The vocabularies are transformed into a directory model representation in the UI, which is hence scalable, extensible and easily modifiable.

The related work does not provide a solution, end-user tool and context framework for enhancing existing mobile device applications with context-aware features by using a small-screen mobile device alone. The related work does not provide a solution for customizing the multimodal interaction of novel modalities in a mobile device. Moreover, the approach in this dissertation has a mobile device-centric viewpoint, instead of infrastructure-centric. The software framework, the customization tool and the sensors are located in a smart phone, which tracks the environment, instead of vice-versa. The advantage of this approach is that the mobile device can be customized and used anywhere and independently of any infrastructure. Concerning the input modalities, the related work does not present real-time pattern recognition of freely user trainable acceleration-based gestures in a mobile device and near-field RFID tag read/write accessory where both modalities are customizable to activate any available mobile device functions.

9. Evaluation

In the first chapter, research methods were introduced for each of the three main sub-topics of the dissertation: blackboard-based context framework and API, context representation and ontology, and context abstracting and recognition. For each sub-topic, a literature review was conducted to assess the state of the art, requirements were specified, and corresponding development in the form of design, implementation or experiment was presented.

This chapter focuses on evaluating the design and the implementation of each sub-topic. Context framework computational performance is evaluated quantitatively. The evaluation of context framework API and ontology is mainly qualitative. To date, no detailed reference measurements have been made for other context frameworks and ontologies, which makes quantitative comparison impossible without first evaluating the related frameworks. No corresponding context frameworks yet exist for mobile terminals. The context recognition case study results were evaluated quantitatively. Additionally, this chapter evaluates the implemented applications that use abstracting of sensor data within the framework.

A number of implemented applications were selected for the evaluation. Most of the selected applications were evaluated in a real Series 60 mobile phone. The target platform for each application is marked in the tables 20, 22, and 23. All the applications for which the target platform is marked as ‘Series 60, context framework’, were evaluated in a Series 60 mobile phone having a functional context framework and real Context Sources.

The evaluation criteria are specified for each sub-topic in the corresponding sub-chapter. The aim is to evaluate the implementations of the designs against the requirements. The requirements were set to address the deficiencies found in the related work in the literature review, and to create the desired system by solving the research problems. Hence the fulfilled requirements in the implemented framework and the implemented real context-aware applications utilising the framework can be used to verify the success of the results. This chapter is partly based on the articles by Korpipää et al. (2003b, 2004a, 2005a, 2005b) – i.e., concerning the design and implementation applied in the evaluation. The evaluation itself has not been published prior to this dissertation.

9.1 Context framework

Research methods for the blackboard-based context framework and API include a literature review, development and evaluation. The literature review compared different architecture models and formed a basis for specifying the requirements for the context framework and API. In the development phase, the framework requirements were analysed and the framework was designed and implemented according to the requirements.

The implemented context framework is evaluated in this chapter. A set of Context Sources was implemented to provide real sensor-based and other context data for a set of implemented applications through the Context Manager. Some of the applications was controlled with the Application Controller. The framework is evaluated against two criteria:

1. Which of the requirements are fulfilled by the designed and implemented context framework?
2. To what extent do the selected example applications (a sub-set of the implemented applications) utilise the features specified in the requirements?

These two criteria were selected because the aim is to evaluate the implementations of the designs against the requirements. The fulfilled requirements in the implemented framework and the implemented real context-aware applications utilising the framework verify the success of the results.

9.1.1 Applications

Several applications were implemented utilising the context framework. Eight example applications (Table 20) were selected for the evaluation. The applications were selected based on the following criteria. The criteria were selected to verify the applicability of the framework for different types of applications and different types of input sources – i.e., to verify the scope of the framework applicability. Moreover, to reflect the potential real-world usability of the framework the applications were chosen considering their assumed usefulness, instead of just proving the concept.

1. For verifying the potential scope of the framework applicability: Can the framework support different types of applications?
2. For examining the potential scope of input data types that can be utilised within the framework: Can the framework support applications with different types of Context Sources?
3. For showing the real-world usability of the framework: Can the framework support potentially useful applications?

Table 20. A summary of selected example applications implemented utilising the context framework.

Application name	**Application type**	**Context source(s)**	**Number of context types**	**Target platform**
1. Context Studio	Customizer	Sensors, terminal events	~25	Series 60, context framework
2. Proof-of-concept	Context monitor	Simulated sensors (recorded data)	~10	Series 60, context framework
3. Proximity	Proximity-based action	Device proximity	1	Series 60, context framework
4. WIRSU	Weather monitor	Wireless sensors	3	Series 60, context framework
5. Movement	Movement-based action	Accelerometers	2	Series 60, context framework
6. RFID tag	Tag-based action	RFID tag	1	Series 60, context framework
7. Gesture control	Gesture recognition-based action	Accelerometers	1	Series 60, context framework
8. Observation camera control	Tag-based external device action	RFID tag	1	Series 60, context framework

Each of the example applications is briefly introduced next.

1. Customizer (Context Studio) was introduced previously. The Customizer, together with the extended context framework, enables the end-user development of context-aware applications (Korpipää et al. 2004a, 2005a, 2005b). Any number of Context Sources producing any type of context described in the ontology can be customized with a graphical user interface to a set of available application functions. The context framework handles all context management tasks separately from the applications, facilitating the customization of existing applications without programming.

2. A Proof-of-concept application was built to evaluate the use of the Context Manager API functions. The Context Source produced contexts of multiple types, which were added to the Context Manager blackboard. A context monitoring application could access the data from the Context Manager. The application could perform all functions specified in the Context Manager API, including subscriptions and queries. The context data for the application consisted of data recorded from a real-world scenario (Korpipää et al. 2003a). The data was abstracted offline. The Context Source produced the contexts simulating the real-world scenario.

3. The Proximity application shows that with the framework, a handheld mobile device's proximity (Bluetooth) to another device can be used to activate a function in an application. The Context Source provides device proximity contexts, which are added to the Context Manager blackboard. Based on a certain received context value, the framework executes an application control function.

4. WIRSU is a wireless weather station (Huttunen et al. 2003). A Context Source reads the sensors in the weather station over a wireless connection (Bluetooth) and adds the context data to the Context Manager blackboard. The application utilises the context framework for monitoring the changes in the weather.

5. Movement contexts are produced by a Context Source, which abstracts measurement data from accelerometers embedded in a mobile device. The abstracted context events are added into the Context Manager blackboard.

Two movement context types are combined in a rule for triggering a device system event. When the rule condition is met, the framework executes the device system control function.

6. An RFID tag Context Source adds the received RFID context to the Context Manager blackboard. Based on a certain received context value, the framework executes an application control function.

7. The user explicitly performs a gesture, which is recognised from the acceleration signals by a Context Abstractor implemented as a Context Source, which adds the recognition result as a context object to the Context Manager blackboard. The context framework performs an application control function based on the context event.

8. An RFID tag Context Source adds the received RFID context(s) to the Context Manager blackboard. Based on a certain received context value, the framework executes an external device (observation camera) control function.

The example applications 5–8 were developed by using the Customizer.

9.1.2 Requirements realization

According to the evaluation criteria, each requirement in the requirements analysis chapter is revisited to evaluate the realization with the designed framework and implemented application examples. The application examples that fulfill the requirement are referred to with the application number.

1. Concurrent context management in mobile device

The framework facilitates the use of the context for the applications that are located in the mobile handheld device (applications 1–8). The context framework is able to handle the information acquired from the device internal sources (applications 1, 2, 5, 7) and external sources (applications 3, 4, 6, 8). The external context information may come from the local infrastructure (applications 3, 4, 6, 8) or the global (IP networked) infrastructure. IP-based sources were not included in the example applications. The management of the sensor information is supported (applications 1, 2, 4, 5, 7). Handling multiple device internal sources concurrently is supported (applications 1, 2, 4, 5). Concerning internal sensor sources, the processing of the information is done in the device itself (applications 1, 2, 4, 5, 7). The Context Manager and the blackboard reside in the handheld device carried by the user (applications 1–8).

Concurrent processing of acquiring, abstracting, storing and delivering context from multiple sources is supported (applications 2–8). The context management system is able to handle multiple contexts that appear at the same time (applications 1, 2, 4, 5). Concurrent use of multiple contexts by multiple applications is supported. Managing multiple applications was tested with four applications that each simultaneously utilise the same context. The example applications were tested individually. The Context Manager has a queue for incoming contexts.

Since the blackboard manager is in the device, disconnection does not prevent the functioning of context-exploiting applications (applications 1–8). Disconnection is seen by the application as having no changes in the context from external sources.

2. Requirements for the application programming interface

The Context Manager provides a set of services that can be used by any client through an API (applications 2–8). Client is here referred to as a device local client.

Any client is allowed to add context to the blackboard, and any client is allowed to use it (applications 2–8). The clients are allowed to subscribe to be informed about changes in the context (applications 2–8). When a context event occurs, and there is a change in the context, the client is either informed or controlled, but otherwise no data is sent to the client (applications 2–8).

Three types of basic subscriptions are supported: Context change, Context start and Context end. Context change informs the client every time the context value of the subscribed context type changes (applications 2–8) (the client can be Application Controller). Context start and Context end inform the client about the start and end of a context value for a context type (application 2). The client can unsubscribe all the subscriptions that it owns (applications 2–8).

Another way of using the context information is directly requesting it from the Context Manager, similar to making queries from a database. Three basic types of queries are supported: Context set request, Latest contexts request, and Time interval request (application 2). The context (object) is returned based on context type, source, or both (application 2). Context set request returns the contexts of a given set of context types or sources (application 2). Latest contexts request returns a given number of latest contexts, and contexts of a given time interval are returned in time interval request (application 2).

3. Flexibility in handling new contexts

Adding new contexts and new elements that produce, process or use context do not require making changes to the Context Manager, nor to any other frozen spot element in the framework (applications 1–8). The framework elements, other than the central Context Manager, are plug-ins to the Context Manager. The example application elements that use the Context Manager connect to it at device boot (applications 1–8). New contexts are handled as data objects (applications 1–8). The context framework is able to handle new context types and values without changes to the framework entities (assuming that the Context Source(s) have the capability of receiving new contexts). This is not shown by the example applications. The application always gets the context from the same blackboard, regardless of the source, and the sources have one place where context data is written (applications 2–8).

4. Context abstracting and recognition

It is possible to add, modify and remove Context Abstractors, Context Sources, and Change Detectors from the framework online – i.e., it is possible to plug in the components. In the example applications the Context Sources connect to the Context Manager at device boot (applications 5, 7). The framework supports abstracting and recognizing context from multiple sources and time sequences (applications 5, 7). The framework supports sensor fusion. Recognition of higher level contexts from existing ones can be performed from a set of contexts, and from a context history (not shown by the example applications).

The result of a previous recognition can be used as a further input for another recogniser. From the client viewpoint, the use of abstracted contexts is transparent (applications 1, 2, 4, 5, 7). The client may subscribe to a higher level context as to any context.

Every time a context in the recogniser input set changes, recognition is performed and new higher level context is added to the blackboard, if it has changed. The client is informed about the possible change normally. The example applications do not demonstrate these two features.

The abstraction level of the data received by the Context Manager may vary. Three cases can be identified:

- Context Manager receives from the source event-based abstracted contexts that do not require further abstracting to be used by the application (applications 2, 5, 7).
- Context Manager receives from the source event-based abstracted information that requires further abstracting. In this case Context Recognisers can be used. This is not demonstrated by the example applications.
- Context Manager receives raw measurement data (application 4) that is updated continuously and possibly with a high frequency.

In the third case, if the frequency is low, the source may add the data directly to the blackboard, and the abstractors receive the data by subscribing to it and perform further abstracting (application 4). If the frequency of continuous input is high, the source itself abstracts the incoming data before adding it to the blackboard (applications 5, 7). The source also performs change detection, so that the incoming high-frequency data is converted to event-based data (applications 5, 7).

The example applications demonstrate context abstracting (applications 4, 5, 7) and HMM-based classification (application 7).

5. Event-based communication of context to application

The blackboard-based context framework is primarily used for delivering data to applications as events (applications 2–8). If the incoming data from the sources is continuous, the framework abstracts it so that data provided to the application or application control with the subscription-indication mechanism is event-based (applications 5, 7). There are two basic ways of dealing with continuous incoming data within the framework:

- Context sources that receive information from external sources simply forward it to the blackboard. Possible context abstracting and change detection will be performed after the Context Manager. This approach is feasible if the frequency of incoming data is low (application 4).
- The Context Source itself performs abstracting and change detection before adding the context to the blackboard. This approach is preferred if the frequency of incoming data is high (applications 5, 7).

6. Context database

The Context Manager contains a relational database to store the context instances into a permanent memory (applications 2, 4). In order to maintain the availability of the context history for the device applications in the event of reboot, a permanent storage is provided for the context information (applications 2, 4). The history length for each context type stored in the database is configurable. (applications 2, 4). The number of context types in the database is configurable.

The use of the permanent storage database provided by the Context Manager is optional (applications 2–8). This option improves the performance, especially with high rates of context additions (application 5). For short life span data, the Context Manager contains a fast cache memory, which has the history length of one for every context type (applications 2–8).

7. Context caching

Clients are not allowed to delete contexts from the context table. Deleting context values is handled by the Context Manager, so that the specified context history length is maintained for each context type (applications 2, 4).

In the evaluated implementation, the context types were not deleted centrally by the Context Manager due to the low number of different context types in the experiments. Applying memory caching techniques for managing the number of context types in the database is further work. Renaming, if required, is supported by the abstractors. Renaming is not shown by the example applications.

8. Time resolution of context

The maximum communication frequency for continuous communication through the blackboard is reduced by performing change detection in Context Sources (application 5). Moreover, for sources that produce context data at a quick pace, the permanent context history is configured short or omitted (application 5). The example applications have Context Sources that produce different kinds of context data to the Context Manager, categorized in Table 21. Categorisation follows that specified in Table 7.

Table 21. Categorization of context data added by Context Sources to the blackboard, with the example applications.

Category number	**Data abstraction level**	**Data communication type**	**Data frequency**	**Example applications**
1.	Raw	Continuous	Low	4
2.	Raw	Continuous	Moderate	-
3.	Raw	Continuous	High	-
4.	Symbolic	Continuous	Low, Moderate	2
5.	Symbolic (or raw)	Event	Low, Moderate	3, 5, 6, 7, 8

Application 2 was used to add an average of ten symbolic contexts per second continuously to the blackboard, with inserts into the relational context database. The Context Manager had a history length between 10 and 100 for all of the context types. Context Source did not perform change detection; this was performed by the Context Manager.

Applications 3, 5, 6, 7 and 8 had Context Sources that produced symbolic event-based data. For tag and proximity applications (applications 3, 6, 8), the frequency of produced context events is typically low. Application 4 produced continuous raw data, and had a configurable context add frequency, which was typically set low.

In application 5, context abstracting and change detection were performed by the source, and the contexts were not stored in the permanent database. In Application 5, the category of data incoming to the Context Source was 3 (raw, continuous, high), but to the Context Manager it was 5 (symbolic, event, moderate). The frequency of incoming continuous data was transformed from 32Hz of raw data to an event-based symbolic data flow of 2Hz maximum frequency. As was discussed in Chapter 4.4, the Context Source should perform the abstracting and change detection in order to reduce the traffic to the Context Manager with context data categories 2 and 3. In application 5, the maximum frequency of adding contexts in the Context Manager blackboard was 2Hz for each context type. This was implemented so that after adding a context instantly upon change, the Context Source imposed a half-second delay before the next context could be added. The number of simultaneously produced context type values was five, and hence the maximum number of event-based contexts added per second was ten. No perceivable delays were noticed in the system response to the user interface during user interaction. The maximum limits for context types and frequency were not reached or explored.

In application 8, the input data was not continuous since only the gesture data, marked by the user with a button, was captured by the Context Source. The input data abstraction level was raw, and the frequency high. Context Source performed the gesture recognition. The recognised context was not stored in the permanent database. No perceivable delays were noticed in the system response to the user interface during user interaction.

9. Change detection

The Context Manager performs basic string match change detection for each new context it receives, compared with the previous one stored in the cache memory. If there had been a change, subscribers to that context are informed about the change, and a new context instance is added into the database (applications 2, 4). Using a separate Change Detector is not shown by the example applications.

The context change detection is optional and chosen by the Context Source (applications 1–8). Applications with implicit interaction use change detection either in the Context Manager (applications 2, 4) or in the Context Source (3, 5). Applications with explicit interaction do not use change detection (applications 6–8).

10. Context confidence

The data object representing context, handled by the Context Manager, has a confidence attribute. Hence instances of context information contain a property that can describe the confidence of the instance. In the example applications, confidence property is used to describe fuzzy membership (applications 2, 5). However, confidence is not utilised for any task in the example applications, mainly since the chosen inference framework was rule-based for the selected applications. The use of context confidence is optional – i.e., the attribute is not required to be set by the Context Sources that add context objects to the blackboard.

11. Context representation

Adding new contexts and new elements that produce, process or use context does not require making changes to the Context Manager, nor to any other frozen spot element in the framework (applications 1–8). Similarly, no changes are required in the frozen spot elements based on the syntax of the incoming context. Common context properties and naming conventions were defined to enable the use of contexts through a common API (applications 1–8). The context representation facilitates the use of context data with the Context Manager API functions (applications 1–8). The context representation defines the structure of the context, and enables the creation of vocabularies that describe useful context types with a sufficient level of detail for use (applications 1–8).

The context framework does not strictly restrict the context representation, but it does provide a template and instructions for producing contexts that facilitate the simplified use of, e.g., sensor-based data with the given API (applications 1–8).

12. Application control

Context-based application control was separated from the applications themselves (applications 5–8). Existing applications can be called from the Activator to control them (applications 5–8). The context framework has an Application Controller entity, which can be used for connecting context events to the available functions of the applications (applications 1, 5, 6, 7, 8). The Application Controller enables controlling applications based on context events, without modifying the applications (applications 1, 5, 6, 7, 8).

13. Customization

Context-action (input-output) mappings are not hard-coded into the framework entities (applications 5–8). Applications 2–4 include control functions. The connection between inputs and outputs can be defined without programming executable code (applications 5–8). During the framework operation it is possible to delete, change and modify the mapping between inputs and outputs (applications 5–8).

The end-user has the possibility of customizing the way of interacting with applications and external appliances (applications 1, 5, 6, 7, 8).

There were several usability requirements for the customization tool: it was required to be easy to learn, effective, efficient, and satisfying, and the user should feel in control of the system (application 1). The usability was evaluated with a user test of ten users. The test was carried out with a Series 60 smartphone, with real context sources and a fully functional context framework. Before evaluating the implementation, the user interface of the tool had been tested with two iterations of paper prototypes during the development process, and improved accordingly (Korpipää et al. 2004). The iterative user testing during development is considered valuable as it reduced the need for corrections after the implementation. The test results indicated that the usability requirements were well satisfied (Korpipää et al. 2005a, Häkkilä et al. 2005). A detailed analysis of usability is beyond the scope of this dissertation.

9.1.3 Discussion

Nearly all requirements were fulfilled by the framework and widely used in the example applications. However, context history was only used in the proof-of-concept application 2. The permanent storage of context was not required in other example applications. The type of example applications, and the type of contexts they utilise, requires only short-term memory. Only one context needs to be stored for detecting change when a new context value appears. Based on the examples, a conclusion can be made that context information utilised in mobile handheld devices is active, short-term information, which rarely requires permanent storing. However, it can also be speculated that, for instance, location-tracking applications would benefit from context history, permanent storage and the API functions for accessing it.

Context abstraction and recognition functionality was implemented in the Context Sources in the example applications – i.e., Context Abstractors were implemented as part of Context Sources. The gesture recognition was performed by a separate process, to which a Context Source was subscribed. Implementing an abstractor in a Context Source is the most efficient solution. Based on the example applications, it seems that Context Abstractors as separate entities are not needed. However, there were no cases in the example applications where a higher level context was required to be classified from other contexts. The separate abstractor entity is necessary in these kinds of cases. The same applies to Change Detector entity; the example applications did not use a separate Change Detector. Change detection was performed by Context Source or Context Manager, which is performance-wise the most efficient solution.

Concerning sensor-based data management within the framework, compared with using a direct flow of raw data from the source to the application, the message traffic up to the application decreases significantly. The application can process other tasks while no important changes occur in the context. With the Application Controller, all context management tasks are performed by the framework, up to activating an application or platform event based on context.

The framework provides an application programming interface for simplified development of context-aware applications. Evaluating the usability of a programming interface is beyond the scope of this dissertation. However, a comparison of the number, complexity and uniformness of API functions with the widget-based approach shows a clear advantage of the blackboard-based API. The hypothesis of simplified development was verified by providing an end-user development tool that uses the API.

The scope of possible context-aware features to develop utilising the framework and the Customizer depends on the number and quality of available contexts produced by the Context Sources and Abstractors, and the number and scale of available applications and actions. The scope is increased by allowing any event or state that is relevant to the user interaction with the device or an application to be used as a context trigger. This includes implicit and explicit events, such as direct control commands. Several types of applications with several types of Context Sources were built utilising the framework and Customizer. Several dozen context values and application actions were available for customization, including a few external device actions.

Different types of applications were selected for verifying the potential scope of the framework applicability, and different types of Context Sources for examining the potential scope of input data types that can be utilised within the framework. Based on the sample set of applications, the framework can be concluded to have a wide scope and the ability to cover different types of input data. The discussed sample set is sufficient grounds for a generalisation: the framework offers a generic platform for abstracting and managing different types of sensor-based information and a wide range of other input abstractions, and for enabling customizable context event-based application control.

9.2 Context representation and ontology

Research methods for context representation and ontology include a literature review, development and evaluation. The literature review compared information representation methods from the literature and analysed their suitability for context information representation. In the development phase, based on the literature review, the requirements and design principles were specified for an ontology, and an ontology was designed for mobile device sensor-based context-awareness according to the requirements.

The designed ontology is evaluated in this section. The evaluation is based on the applications that have been developed utilising the ontology. As stated before, the representation refers to the entire representation of the context, including the ontology, the selected syntax and the context-action rules. The ontology refers to the context data structure and properties, the vocabulary model and the domain vocabularies. The context representation and ontology are evaluated against two criteria:

1. Which of the requirements are fulfilled by the designed and implemented representation and ontology?

2. To what extent do the selected example applications (a subset of the implemented applications) utilise the features specified in the requirements?

These two criteria were selected because the aim is to evaluate the implementations of the designs against the requirements. The requirements were set to address the deficiencies found in the related work in the literature review, and to create the desired result by solving the research problems. Hence the fulfilled requirements in the implemented representation and ontology utilised within the framework, and the implemented real context-aware applications utilising the representation and ontology within the framework, verify the success of the results.

9.2.1 Applications

Several applications were implemented utilising the ontology. Eight example applications (Table 22) were selected for the evaluation, based on the following criteria. The criteria were selected to verify the applicability of the ontology structure and vocabulary model for different types of applications and for expressing abstractions derived from different types of input sources – i.e., to verify the scope of the ontology structure and vocabulary model applicability. Moreover, to reflect the potential real-world usability of the ontology, the applications were chosen considering their assumed usefulness, instead of just proving the concept.

1. For verifying the potential scope of the ontology structure and vocabulary model applicability: Can the ontology support different types of applications, including applications outside the context framework?
2. For examining the capability of expressing different types of abstractions with the ontology: Can the ontology structure and vocabulary model support applications with different types of input context data?
3. For showing the real-world usability of the framework: Can the ontology support potentially useful applications?

Table 22. A summary of selected application examples implemented utilising the context ontology model.

Application name	**Application type**	**Context source(s)**	**Number of context types**	**Target platform**
1. Context Studio	Mobile device personalisation	Sensors, terminal events	~25	Series 60, context framework
2. Proof-of-concept	Context monitor	Simulated sensors (real data)	~10	Series 60, context framework
3. Context sharing	Mobile context sharing	Sensors	~5	Pocket PC
4. Airport service	Location-based service	Location, proximity	~20	PC, Series 60 device
5. Bus schedule	Presentation adaptation	Simulated sensors (real data)	~5	Series 60 emulator
6. Movement	Movement-based action	Accelerometers	2	Series 60, context framework
7. RFID tag	Tag-based action	RFID tag	1	Series 60, context framework
8. Gesture control	Gesture recognition-based action	Accelerometers	1	Series 60, context framework

1. Context Studio was previously introduced. Context Studio utilises the ontology for automatically creating user interface views from the vocabularies defined according to the vocabulary model. The context triggers and application actions are both represented based on the ontology vocabulary model. Moreover, the vocabulary model is used for defining context-action rules, which are presented in the UI and encoded formally with CEP, so that the framework can automatically execute user-defined context-action rules.

2. The proof-of-concept application was previously introduced. The application utilises the ontology for describing contexts that were recorded with a set of sensors from a real-world scenario and abstracted offline. The sensors included a 3D accelerometer, 2 light sensors, humidity, temperature and touch.

3. Keränen et al. (2003) experimented with a context-sharing application implemented on a PDA. The context ontology was utilised for sharing and expressing sensor-based context information related to another person(s). Moreover, the context values were presented using a graphical visualisation. This application was not built using the context framework presented in this dissertation.

4. Airport service is an application where the user is provided with a service in his/her mobile device based on the location of the user. The context ontology vocabulary was used to model the domain contexts, such as the location, identity, user and flight status information. The context vocabulary defined all the available contexts, and CEP was used for encoding and transferring the context instances. The application utilises the blackboard-based networked PC MUPE context engine (Suomela et al. 2003).

5. Mäntyjärvi and Seppänen (2002) presented an application that utilised the context ontology for describing contexts related to user activity and environment. The contexts were used for fuzzy adaptation of information presentation regarding bus schedules. The application was studied in an emulator and used recorded real-world data abstracted offline. This application was not built using the context framework presented in this dissertation.

6. The movement-based action application was introduced previously. The movement contexts are presented utilising the ontology, and combined into a context-action rule based on the graphical description created by the user with Context Studio. The action description also utilises the context vocabulary model. The framework uses the rule for automatically triggering a device system event when the rule condition is true.

7. The tag-based action application was introduced previously. The tag contexts are presented utilising the ontology. A context-action rule is generated based on the graphical description created by the user. The framework uses the rule for automatically triggering an application function when the rule condition is true.

8. The gesture control application was introduced previously. The gestures are presented utilising the ontology. Based on the graphical description created by the user, a context-action rule is generated. The framework uses the rule for automatically triggering an application function when the rule condition is true.

9.2.2 Requirements realization

The feasibility of the representation and ontology is assessed by analysing the designed ontology against the requirements, and the example applications that have been implemented utilising the ontology. According to the evaluation criteria, each numbered requirement is revisited to evaluate the realization with the designed ontology and implemented application examples. The application examples that fulfill the requirement are referred to with the application number.

1. Simplicity

The ontology structure and vocabulary model are simple enough to be easily utilised by application developers (applications 1–5). The vocabularies are easily understandable by the user (applications 1, 6, 7, 8). The end-user's ability to easily understand an ontology vocabulary, in the form of a Customizer UI, was evaluated in two user studies with 7 (Korpipää et al. 2004a) and 10 users (application 1) (Korpipää et al. 2005a, Häkkilä et al. 2005). The latter user study was conducted with a real prototype framework, Customizer and Context Sources in a Series 60 phone. In both studies the users were presented with five scenarios that implicitly referred to making rules with the briefly introduced tool. Although some of the users needed advice during the first scenario in both studies, and initially performed a search to find the correct rule elements from a directory structure having over a hundred context and action values, all users were able to complete all scenarios and defined the correct and intended functionality with the tool.

2. Practical access

The ontology structure and vocabulary model enable practical and efficient queries and subscriptions to context information through the Context Manager API (applications 2, 6, 7, 8). Queries can be made by using Context type or

Source, or both properties as a search key (application 2). The Context type sub-concepts enable queries of a sub-tree of the context hierarchy formed by the Context type concepts (application 2). All context objects matching the partial tree are returned. Subscriptions by Context type enable the client to receive indications of changes in context values for the corresponding type (applications 2, 6), and to receive indications without change detection for explicit inputs (applications 7, 8).

3. Flexibility, expandability

The context ontology is expandable to new domains, which is achieved by defining vocabularies that describe the contexts of a domain, following a vocabulary model (applications 1–8). The example applications show that multiple domains of use are covered. Moreover, in the Nomadic Media project context vocabularies were defined for airport, hospital and home domain scenarios, according to the vocabulary model. The vocabulary model also facilitates describing application actions (applications 1, 6, 7, 8). The existing vocabularies are modifiable (applications 1–8). The vocabularies are even definable by the end-user, who can set new Context values (applications 1, 8). The vocabularies are completely independent from the context framework, but all context information described as defined according to the ontology structure can be used within the framework.

4. Domain

The ontology supports easy utilisation of the abstracted context information of multiple domains, including sensor-based information (applications 1, 2, 3, 5, 6, 8).

5. Facilitate inference

The representation enables efficient inference by the Context Abstractors and Application Controller. The efficiency of a method is dependent on the type of the task. The developed representation does not restrict the inference to any single method. In Context Sources and abstractors any method is allowed for producing symbolic contexts (applications 6, 8). The Application Controller utilises a rule-based inference (6–8), but other inference frameworks are possible for application control.

6. Genericity

The ontology supports different types of context information (applications 1–8). The representation structure applies across domains, and domain-specific concepts are defined in the extensible vocabularies (applications 1–8).

7. Efficiency

The representation is memory-efficient. Within the framework implemented in the target mobile device platform, a context instance encoded as a context object, according to the ontology, consumes at maximum 249 bytes with Unicode strings, and less when compacted in the relational database (applications 1, 2, 6, 7).

8. Expressiveness

The possible amount of detail in describing any single vocabulary context instance is not high, and the number of fixed context properties is low. The semi-informal representation of vocabularies does not include defining complex constraints. Complex context instances can be decomposed (application 4). The trade-off of satisfying the previous seven requirements is in the lower detail level of a single context expression. However, the Attributes property is designed to add details if necessary. The Attributes property was not needed to convey any information in the example applications.

9.2.3 Discussion

The requirements for the representation and ontology were fulfilled by the implemented ontology structure, vocabulary model, and and domain vocabularies that were defined for the different types of applications in the example application set. The example applications show that the ontology structure and vocabulary model are applicable in multiple application domains. The ontology vocabulary is machine-readable and the common data structure facilitates the processing and use of context within the framework, and automatically generating user interface views based on it. The structure of the ontology is easily understandable by humans. The vocabularies can form a shared conceptualisation of a domain.

Of the eight requirements, expressiveness was a trade-off to the other seven requirements. However, in light of the example applications, a more expressive ontology structure was not even required. Complex context information structures can be decomposed to the atomary expressions that can be represented with the ontology structure. All the necessary contexts in the domains of the example applications could be represented with the name-value tuples formed by context type and context value. From the properties defined in the ontology structure, the Attributes-property was not needed in any of the example applications.

In addition to the example applications, sensor-based vocabularies of the context ontology have been utilised in multiple research studies. The ontology has been used as an underlying representation for an explorative analysis of the structure and dynamics of higher level contexts derived from sets of lower level contexts by segmenting time series of atoms, and for unsupervised clustering from both sets and time series of atoms, in an attempt to raise the abstraction level of the context data (Himberg et al. 2001; Mäntyjärvi et al. 2001, Flanagan et al. 2002). Mäntyjärvi et al. (2002) use the representation as a basis for experiments in utilising information from multiple mobile devices in recognizing the context of a group of mobile terminals and their users collaboratively.

Different types of applications, including applications outside the context framework, were selected for verifying the potential scope of the ontology applicability. Applications with different types of input context data were selected for examining the capability of expressing different types of abstractions with the ontology structure and vocabulary model. Based on the sample set of applications, the ontology structure and vocabulary model can be concluded to have a wide scope and the ability to express a wide variety of different types of abstractions. The ontology vocabulary model facilitates describing application actions in addition to contexts. The Customizer, the ontology and the context framework together facilitate easy end-user development of context-aware features into existing mobile device applications without modifying the application code.

9.3 Context abstracting and recognition

Research methods for context abstracting and recognition include a literature review, development, experiment and evaluation. The literature review introduced a few widely applied machine learning and inference methods potential for context abstracting and recognition in a mobile device. The development included implementing Context Abstractors and recognisers in the context framework. The experiment included choosing suitable method(s) for context recognition, and evaluating them with a case study based on a scenario in a home environment. For evaluating the context abstracting and recognition experiment, quantitative measures were presented for context recognition accuracy in the case study.

This chapter evaluates the feasibility of context abstracting and recognition within the context framework functioning online in a mobile phone. The implemented framework elements and applications that operate utilising the elements are evaluated against the requirements set for the abstracting and recognition methods.

9.3.1 Applications

Applications for the evaluation of context abstracting and recognition (Table 23) were selected based on the following criteria. The criteria were selected to verify the applicability of the framework for online context abstracting and recognition, and for analysing what could be recognised from the environment with a set of sensors.

1. To verify the applicability of abstracting and recognition methods within the implemented context framework (applications 2, 3): Does the framework support sensor-based context abstracting and recognition for real-world applications?

2. What can be recognised from the environment with a set of sensors, bearing in mind the restrictions of a mobile device (application 1)?

3. For showing the real-world usability of the framework: Can the framework support potentially useful applications that utilise context abstracting or recognition?

Table 23. A summary of selected application examples implemented utilising sensor-based context abstraction and recognition.

Application name	**Application type**	**Context source(s)**	**Number of context types**	**Target platform**
1. Context classifier	Context monitor	Sensors and audio (offline)	~10	PC, offline
2. Gesture control	Gesture recognition-based action	Accelerometers	1	Series 60, context framework
3. Movement	Movement-based action	Accelerometers	2	Series 60, context framework

1. Context classifier is the application presented in the context recognition case study. The offline application reads recorded sensor data and visualises the classified contexts as a function of time. Naïve Bayesian networks are used as the context recognition method.

2. Gesture control is an explicit interaction modality utilising the context framework. The acceleration sensor signal is processed up to the classification phase of the pattern recognition process. Hidden Markov Models (HMM) are used as the model for gesture training and recognition (Mäntylä 2001, Mäntyjärvi et al. 2004, Kela et al. 2005). Gesture recognition accuracy with the applied method has been evaluated quantitatively by, e.g., Mäntyjärvi et al. (2004). Gesture recognition and control as an explicit modality are not within the focus of this dissertation. Gesture recognition is used in this dissertation to evaluate the feasibility of HMM-based classification within the context framework, and can be utilised as additional interaction information for context-aware applications and multimodal interaction. Gesture control was evaluated in the target hardware, a Series 60 mobile phone having accelerometers and the context framework.

3. The Movement-based action application represents the context abstracting category, where the signals are processed up to the feature extraction phase in the pattern recognition process. This application category was introduced in the framework evaluation section.

9.3.2 Requirements realisation

The applications in Table 23 use methods for feature extraction, and applications 1 and 2 use classification methods. The requirement realisation discussion is focused on the applied machine learning (classification) methods, i.e., Naïve Bayesian networks and HMMs. Both methods were discussed in the literature review, and the conclusions in this section are based on references in the literature, in addition to the experiments that were conducted.

1. Efficiency

The Naïve Bayes network (application 1) inference is computationally very efficient. As a context recognition method, it is very well suited to continuous real-time recognition in low computing power mobile devices, although application 1 was tested offline in a PC. The HMM (application 2) inference is also quite efficient. It is suitable for mobile device event-based recognition when either an application or the user explicitly starts the recognition, as is the case in application 2. In application 2, the HMM-based classification for gesture control was applied in a Series 60 phone with the context framework, and there were no perceivable delays in the user interaction with phone applications.

2. Handle multidimensional input data

The Naïve Bayes network can handle multidimensional input data (application 1), whereas HMM is designed for modelling one-dimensional sequences. In the case of multidimensional input data, such as accelerometer data, a dimensionality reduction is required for the three-channel data before applying HMM (application 2).

3. Handle uncertainty

Bayesian networks and HMMs are both suitable for classifying incomplete and noisy data (applications 1, 2).

4. Updating flexibility

Learning conditional probabilities in the Naïve Bayes network is fast and requires little training data. New training data can be utilised incrementally. New contexts to classify can be modelled as new networks, independent of the others. HMM requires also relatively little training data in the example application. Training is more computationally expensive, and causes a small delay in the example cases in a Series 60 phone. HMMs can also be incrementally trained, and new contexts (gestures) are added as new HMMs. The learning phase is not currently performed within the context framework in both applications. New gestures can be trained separately in the mobile device.

5. Scalability

The computational complexity of the Naïve Bayes network is small in inference and learning. Hence large models are still efficient. Furthermore, when the inputs are divided for different networks based on background knowledge, many networks can function in parallel (application 1). In application 2, each HMM corresponds to one gesture, and each HMM is independent from the others, which enables straightforward adding, removing and updating of models. Since the inference is moderately efficient, handling a large number of HMMs is computationally feasible in a mobile phone.

9.3.3 Discussion

Application 1 was selected for analysing what can be recognised from the environment, bearing in mind the restrictions of a mobile device. With Naïve Bayes networks, multiple contexts can be recognised from multi-sensor data with a one-second resolution in a restricted scenario. Analysed against the requirements, the Naïve Bayes is suitable for classification use in mobile devices; it is efficient, can handle multidimensional input data and uncertainty, is flexible to update, and scalable.

Empirical tests show that HMMs can be applied for real-time online classification in a mobile phone within the context framework for recognising discrete accelerometer based gesture commands. Recognition delays are unnoticeable. Explicit gesture commands can be described and used as any context events within the framework. HMMs are computationally moderately efficient, can handle uncertainty – i.e., noise in the acceleration data and incomplete training data – are flexible to update, and scalable as independent models.

Applications two and three were selected based on assumed application usefulness and for verifying the applicability of the selected abstracting and recognition methods within the implemented context framework. Based on the sample set of applications, it is verified that applying feature extraction and classification with the selected methods is feasible within the context framework in a Series 60 mobile phone. Furthermore, continuous concurrent online abstracting of multiple context types is feasible within the framework, as is the use of the abstracted contexts as events for application control.

9.4 Context framework performance

9.4.1 Computational complexity estimation

The computational complexity of the context framework without the effect of specific signal processing algorithms (some of which were separately addressed) in Context Sources or Context Abstractors can be roughly estimated. Without the effect of specific algorithms, the computational complexity of Context Source, Context Abstractor, and Change Detector can be regarded as constant. The variables that potentially most affect the framework performance are the number of contexts C, number of applications A, number of rules R, and number of operations in a rule P.

The context framework is designed so that each server computing element (Context Manager, Rule Script Engine, and Activator) has a queue for incoming data traffic. Hence the computational complexity for C and A is constant. The Rule Script Engine is designed so that each context in a rule has a subscription to the Context Manager. For each incoming context, if the rule does not have the incoming context type in it, the rule is not processed. The computational complexity for the triggered rules (that do have the incoming context type in them) R' is linear O(R'). Each triggered rule R' has a number of operators P, which have a linear computational complexity O(P). Typically both R' and P are low. The computational complexity of the context framework can thus be approximated as O(R'P).

9.4.2 Performance evaluation in target hardware through usage

Tables 20, 22, and 23 show the applications that were used in the evaluation. The applications whose target platform was specified in the tables as Series 60, context framework, were tested in 1–5 different Series 60 phone models having ARM9 104MHz and 123MHz (ARM 2005) processors. Computing performance refers here to the speed of executing a task in the target hardware.

The computing performance of the context framework applied with real applications was tested empirically. The performance of the implemented framework with the real context sources – for fast changing context data that was not stored in the permanent database – was unexceptional in the target hardware. The assessment is based on observing the operation in multiple test use cases. In the test use cases the conditions for a certain action were intentionally fulfilled, and the user monitored whether there is a delay prior to the known action execution. The delay from the framework operation was unnoticeable to the user in the observed non-persistent context data use cases, which include the applications that were discussed in the evaluation. For those test applications that stored context data in the permanent database continuously and with a high frequency, the delay was observable. However, as discussed earlier, there was no need for permanent context storage with the evaluated applications.

The framework operation in customization usage was tested with over twenty test cases (active context-action rules), which were defined using the customization tool. Delays from the framework operation were not noticeable.

The ability of the context framework to handle multiple concurrent applications was tested by performing different numbers of simultaneous different application activations based on context, a maximum of four. The test was conducted by defining four rules, where four different application actions were defined for the same context event. The context framework managed the application control in sequence, in the order of subscription – in this case in the order of the creation of the context-action rules – without noticeable delay. It must be noted that in all test cases the acceleration channels were continuously sampled with a moderately high frequency, and a multitude of Context Sources continuously performed data abstracting for different context types and produced context events to the Context Manager blackboard without a noticeable effect on the framework performance.

Hence the empirical tests indicate that the context framework performance is unexceptionable in the target hardware – i.e., real mobile phones.

9.4.3 Performance evaluation in target hardware quantitatively

The context framework computing performance was measured numerically in the target hardware, in a Series 60 Symbian phone having an ARM9 206MHz processor. The test setup included a context data simulator that provided recorded and abstracted real context data, and seven context-action rule scripts. Context add parameters were set as 'persistent false' and 'change detection false'. The software configuration consisted of Context Manager, Rule Script Engine, Activator, Context Studio, and one Context Source. The memory size of the packet was about 80 kilobytes. Runtime memory consumption was not measured but is insignificant when contexts are not stored into the database.

The processing time used by Context Manager process was first measured with contexts that did not trigger rules. About 250 context atoms were each added to the Context Manager and processed ten thousand times. The average processing time for processing one context atom once was 25 microseconds.

The processing time of Rule Script Engine was measured for triggered rules. There were six contexts in the dataset that triggered a rule; each of them was run ten thousand times within the Rule Script Engine. The average processing time for evaluating one rule operation once was 12 microseconds.

The processing time of the chain Context Manager – Rule Script Engine – Activator – Application (system function call) was measured for five triggered contexts each a hundred times. The average processing time of the whole server chain was 374 microseconds, including the Application call. The extra time compared with the processing time in each individual component was mainly due to the communication overhead, i.e. context switching between the processes, which is a fairly expensive operation in the operating system (Tasker et al. 2000). Another source of overhead is data transfer between processes, although this is not as significant since the data amount to transfer is not large. Although the performance is already unexceptionable, it can be further optimised by rearranging the computational elements into the same process having multiple threads.

As a summary it can be stated that the context framework performance is beyond reproach and more than sufficient for a context-aware application in a mobile phone. Moreover, since the worst-case computational complexity is linear, the context framework is also performance-wise scalable.

9.5 Summary

The implemented context framework, ontology, abstracting and recognition, and customization were evaluated in handheld mobile devices having a Symbian operating system. Several applications of different types were used to evaluate the applicability of the framework. The implementation of the framework was evaluated against the requirements, which were set for answering the research problems. The fulfilled requirements verified the implementation. The use of the features specified in the requirements in different types of applications verified the applicability and scope of the framework.

The contribution of this chapter is the evaluation, based on which the following claims are made. Based on the sample set of applications, the framework is claimed to offer a generic platform for abstracting and managing a wide range of different types of abstractions including sensor-based information, and for enabling context event-based application control for different types of applications. Based on the sample set of applications, the ontology structure and vocabulary model are claimed to have a wide scope and the capability of expressing a wide variety of different types of abstractions. The ontology vocabulary model facilitates describing application actions in addition to contexts. Based on the sample set of applications, it is verified that classification with the selected method, continuous concurrent online abstracting of multiple context types, and the use of the abstracted contexts as events for application control are feasible within the framework in a mobile phone. The context framework is computationally efficient and performance-wise scalable. The Customizer, the ontology, and the context framework together facilitate simple and easy end-user development of context-aware features into existing mobile device applications without programming.

10. Discussion

10.1 Verification of the research problems and hypothesis

The research problems and hypothesis are revisited to verify how they have been answered in the dissertation. The research problems are answered first.

1. **The problem:** What is required to flexibly and efficiently handle all relevant aspects of sensor-based mobile terminal-centric management of context-related information?

   The answer: The blackboard-based software framework, capable of acquiring, storing, and abstracting human-interpretable context information from multiple sources, including sensors, was designed, implemented and evaluated. The framework can deliver context information to mobile device applications in an event-based manner with a publish and subscribe mechanism. The framework can separate context management, including application control, from application code. New Context Sources, Abstractors, Change Detectors and Applications can be plugged in, and Abstractors and Change Detectors can also be created as scripts. The framework was evaluated with multiple applications of different types, performance estimation and measurements, use cases of different types implemented in the target hardware, and a user test was performed with the target hardware.

2. **The problem:** How to represent context information so that it can be systematically processed, stored, used by the applications, and understood by application developers, while maintaining representation extensibility?

   The answer: A human-understandable and machine-processable extensible context ontology was developed and evaluated. The ontology structure was the same across different application domains. Domain vocabularies were defined according to the vocabulary model. The use of the ontology within the framework was evaluated with real applications in the target hardware. The human understandability was evaluated with two user studies, which involved 17 people from outside the mobile phone industry.

3. **The problem:** How can context be recognised and abstracted online into a common representation from many different sources, especially device sensors, producing possibly incomplete and imprecise information?

   The answer: Context abstraction and recognition from multi-sensor data is feasible with the probabilistic machine learning methods that were selected and evaluated in the dissertation. The accuracy of recognising multiple simultaneous activities from multi-sensor data was evaluated quantitatively in the case study. The use of a HMM-based classifier implemented as a Context Source within the context framework in the target hardware was evaluated with gesture recognition from a three-channel acceleration signal.

4. **The problem:** What kind of application programming interface is required for the simplified development of context-aware applications, and, further, what kind of tool would be required for end-user development in mobile handheld devices?

   The answer: The dissertation presents a compact application programming interface based on the blackboard model, where context abstractions described in the ontology are accessed uniformly from one node independent of the source of context. Furthermore, the simplified development of context-aware applications was evaluated with an end-user tool that uses the framework. The tool enables users to easily customize new context-aware features into existing applications. The tool is easy to learn and simple to use, which was evaluated with a user test with the implemented framework, the tool, real Context Sources and abstractors in the target hardware.

The main hypothesis of the dissertation was set as follows.

By solving research problems 1–4 it will be possible to create a functional software framework and tool that enable end-users to quickly customize versatile context-aware applications in a mobile device.

The hypothesis is verified in summary as follows. The created software framework was used in handheld mobile devices. The use was evaluated with the target hardware with several implemented applications, performance tests, and user tests. The framework contains reusable elements of context-aware applications. Multiple context-based application features were developed with the implemented framework without programming changes to the framework code. The framework transparently handles the production, abstraction and delivery of the context information. Transparency means that the application developer does not need to know the underlying operation for producing the context information. Acquiring, abstracting, recognition and delivery of the context was implemented and evaluated with several applications. The framework can perform application control on behalf of the application or the user. Existing applications were enhanced with context-aware features without programming executable code. The framework provides an application programming interface, which was used for multiple applications. An extensible context ontology was created and used within the framework for representing the abstracted context information. The context framework and a customization tool facilitate easy-to-learn end-user development, which was evaluated with user studies.

Hence it is claimed that research problems 1–4 were solved and a functional software framework and tool that enable the quick development of versatile context-aware applications in a mobile device were created.

10.2 Comparison with related work

10.2.1 Summary of contributions

In comparison with the related work, this dissertation has multiple scientific contributions. Table 24 summarises the contributions, followed by a summary comparison with the most significant related work in each sub-topic of the research and the contributions. For the most part, the summarised contributions relate to the scientific publications first-authored by the author of this dissertation, the related patents and the pending patents.

Table 24. Summary of contributions. As an estimated rate, + refers to a contribution, and ++ refers to a major contribution. The rating ++ is based on the significance of the contribution in an article published in an international scientific journal or magazine.

Contribution	**Rate**
Context framework	
1. Extensive requirements analysis for a mobile device context framework	+
2. Software framework for developing mobile device sensor-based context-aware applications	++
3. Mobile device framework support for providing fast event-based abstracted contexts defined in the ontology	+
4. Framework support for context abstracting and context recognition process in a mobile device	+
5. A compact API, which is uniform for all context producers and consumers, for providing and using rapidly changing sensor data as abstracted context objects in a mobile device	++
6. Blackboard-based management of context information in a mobile device with a publish and subscribe mechanism	++
7. Relational context database for mobile device context awareness	+
8. Software framework support for application control and interaction customization in a mobile device	++
9. Implementation and evaluation of a blackboard-based context framework with a set of applications in a mobile device	+
Context representation and ontology	
10. Requirements analysis for context representation and ontology for a mobile device	+
11. Structure and properties for domain-independent representation of context information as data objects	+
12. Generic vocabulary model for describing context instances and vocabularies	+
13. Method of utilising context ontology in customizing context-aware applications	++
14. Method of ontology-based UI generation for mobile device customization	++

Table 24 continues.

15. Implementation and evaluation of context representation and ontology within the context framework with a set of applications in a mobile device	+
Context abstracting and recognition	
16. Requirements analysis for context recognition methods for use in mobile devices	+
17. Recognition of multiple simultaneous contexts from multiple sensor sources	+
18. Model for the transformation of continuous sensor data flow into context change events within a context framework	+
19. Evaluation of the feasibility of continuous multi-action context recognition quantitatively	++
20. Applying HMM-based classification from sensor data involving uncertainty within a context framework for real mobile device applications	+
21. Implementation and evaluation of context abstracting and classification within the context framework with a set of applications in a mobile device	+
Customization and personalisation	
22. Solution for enabling end-user development of context-aware applications in mobile handheld devices	++
23. Solution for enabling user interaction customization in handheld mobile devices	++
24. Solution for enabling customizing multimodal interaction of novel modalities in handheld mobile devices	+
25. Implementation and evaluation of a rule-based customization approach and tool within the context framework with a set of applications in a mobile device	++

10.2.2 Context framework

In comparison with the most relevant related context frameworks, this dissertation particularly focuses on sensor-based information processing in handheld mobile devices. The approach to context awareness is mobile device-centric as opposed to an environment-centric “smart space” approach. As far as is known, no other significant blackboard-based context framework for mobile terminals existed in the literature review. The related environment-centric context frameworks have been prototyped with PCs and laptops (Dey 2000, Mitchell 2002, Ranganathan & Campbell 2003, Wang et al. 2004). Experiments and simulations of context awareness in a networked PC environment do not reflect the constraints and requirements from the mobile handheld device usage viewpoint. Such requirements were thoroughly analysed in this dissertation. The related frameworks do not provide a simplified API for using rapidly changing context information abstracted from sensors and defined in the ontology for a mobile device.

Winograd (2001) argued that the blackboard model is superior for context management. The blackboard-based model offers the advantage of hiding from client all components except the blackboard manager. The client accesses all context data from any source from the same central node. Korpipää et al. (2003b) presented a blackboard-based framework for mobile device context awareness. Wang et al. (2004) utilise a central knowledge base as the context repository, one for each smart space, and use Semantic Web tools for inference. The knowledge base is suggestive of being blackboard-based, but the inference results are not allowed to be stored in the knowledge base. The authors evaluated the system performance with a 2.4 GHz Pentium 4 workstation and report that with the prototype Java-based application the reasoning delays sometimes matter to users. The related context framework literature does not offer a complete mobile device-centric solution to the process of acquiring and storing rapidly changing context information from multiple source sensory data, abstracting and recognising contexts from noisy data, and representing the abstracted data with an ontology up to the use of the abstracted context data through an API and customizable application control in applications in a handheld mobile device.

10.2.3 Context representation and ontology

In comparison with the related work on context ontologies, this dissertation focuses on sensor-based context abstractions, the common properties of the context information, and applying context conceptualisations in mobile device customization. Early mobile device-centric related work contributes the basic structure and categorisation of sensor-based context information (Schmidt et al. 1999a) to this dissertation but does not specify an extensive set of the common properties of context data structure, or a detailed generic model for describing new context vocabularies.

In more recent work, Wang et al. (2004) apply OWL to define an ontology for describing context information related to smart spaces. The ontology consists of an upper-level context ontology and extended context ontologies. The upper level ontology contains three classes of real-world objects (user, location and computing entity, which has a sub-class of device) and one class of conceptual objects (activity) for characterising smart spaces. The ontology defines the class-specific properties required for describing each of the mentioned context sub-classes. The authors identify as future work providing the capability to manage context information uncertainty with reasoning methods such as probabilistic logic, Bayesian networks and fuzzy logic, since sensor-based contexts are not always precise. Korpipää and Mäntyjärvi (2003) propose a set of common basic properties for *all* context objects, with representation support for handling context information uncertainty. Moreover, Korpipää et al. (2003a, 2003b) propose and evaluate methods for context recognition and reasoning for handling uncertainty, such as Bayesian networks and fuzzy logic. Ranganathan and Campbell (2003) use first order logic for describing context-aware behaviour in a PC environment. The related context-aware computing literature does not offer solutions for utilising an extensible context ontology in end-user development of context-aware applications, or generating graphical user interface views based on the ontology, and do not offer a vocabulary model for specifying context vocabularies.

10.2.4 Context abstracting and recognition

The context-aware computing literature does not analyse in detail which context recognition methods are suitable for use in mobile devices. Schmidt (2002) discusses sensor-based context acquisition and use of the context information in several applications related to aware artefacts and sensing environments, and presents a conceptual model for sensor data processing. Concerning sensor data processing, Schmidt (2002) discusses the sensing and feature extraction phases of the pattern recognition process, but does not discuss classification – i.e., context recognition as defined in this dissertation.

Mäntyjärvi (2003) focuses on applying statistical and machine learning methods for an explorative data analysis of the context data. In other words, the author performs context data mining for discovering the context data patterns that correspond to real-world situations. The next step after the explorative data analysis approach, where results are analysed qualitatively, is the traditional pattern recognition approach taken in this dissertation to enable a quantitative evaluation of how well certain contexts can be recognised from the sensor data in a certain scenario. These results can give directions as to which contexts can be recognised reliably and could be applied in practice in the future.

Two more recent studies with pattern recognition and a quantitative evaluation approach analyse context recognition from wearable sensors. Bao and Intille (2004) use five 2D accelerometers attached to five locations on the human body (dominant arm wrist, dominant leg ankle, thigh, arm and hip) for offline recognition of 20 pre-segmented indoor activities. The application area of the study is wearable computing, whereas this dissertation has a mobile device-centric view. Korpipää et al. (2003a) present multi-sensor recognition from a continuous unsegmented data stream, where multiple contexts are classified at the same time instant. Lukowicz et al. (2004) study the recognition of wood shop activity by using microphones and accelerometers worn on the body. Nine different activities, such as sawing, drilling, etc., are recognised one at a time from continuous data by first segmenting the data. One classification result is given for each identified segment, and in one dataset there were 25–30 coarse partitions of different lengths. Korpipää et al. (2003a) classify multiple simultaneous contexts with a one-second resolution – i.e., the classification result is given for each second in the continuous data. In mobile device context-aware applications, such as performing a user interface action based on recognised context, too coarse a segmentation may lead to delays that are annoying to the user.

The related work does not present solutions to classifying multiple simultaneous contexts from multiple sensor sources, and does not evaluate the feasibility of continuous multi-sensor multi-action context recognition quantitatively. Furthermore, this work contributes in presenting and applying a software framework to the transformation of continuous sensor data flow into context change events in a mobile device, and applying classification from noisy multidimensional data within a context framework for real mobile device applications.

10.2.5 Customization and personalisation

As a significant application of context framework, ontology and abstracting, this dissertation presents a tool for end-user development and interaction customization in mobile devices. The idea and concept of personalising mobile device applications was first introduced by Mäntyjärvi et al. (2003). According to the original concept, the user was to mark all the pre-defined context instances that describe a certain situation as a trigger for a certain action. In the customization approach developed into a working tool in this dissertation the concept is modified so that the user only selects the context value(s) the user considers relevant for triggering an action. Moreover, the tool user interface is designed for small-screen devices, the user interface views of the tool are generated automatically based on the ontology, and the context framework enables the actual use of the features that were defined with the customizer.

For comparison, Dey et al. (2004) experiment with a programming-by-demonstration approach for prototyping context-aware applications. The authors have developed a tool for a PC environment that enables the user to train and label models of context, which can be connected to actions. Context models are defined as examples. User-activated creation of context models based on example is feasible when it is performed for a single type of context that is known to the user. In the case of multiple input sources, the programming-by-demonstration approach may lead to behaviour that the user did not intend to have if the user cannot control exactly which input(s) define the situation.

This dissertation presents and evaluates with users a customization tool, the framework, and the ontology that, together, facilitates end-user-controlled customization of novel input modalities for human-computer interaction with a mobile device and external appliances, and facilitate the real use of the customized features in the mobile device.

10.3 Significance of the results

The results related to this dissertation have been published in international scientific journals and conferences, and a workshop. A total of 12 closely related and co-authored articles have been published (Himberg et al. 2001, Häkkilä et al. 2005, Kela et al. 2005, Korpipää et al. 2003a, 2003b, 2004a, 2004b, 2005a, 2005b, Korpipää & Mäntyjärvi 2003, Mäntyjärvi et al. 2001, 2004), of which four are scientific journal articles, seven are scientific conference articles, and one workshop article. The author of the dissertation is the first author in seven of those publications, including three journal articles. In addition, three related and co-authored patents have been either published or a US patent is pending.

The main result of the work is the first blackboard-based context framework and customization tool for mobile device context awareness. The framework can be used for managing any, but especially sensor-based, information as events. The customization tool can be used for personalising user interaction in handheld mobile devices, where the interaction can be either explicit or implicit. An evaluated and viable model and implementation are provided for quick and simplified development of versatile context-aware applications.

The results do not concern merely context-aware computing, they can be generalised to and applied in customizing the use of novel input modalities for interaction with mobile devices, as is discussed in Korpipää et al. (2005b). Any events, including sensor-based events, can be used as inputs for interaction. The framework enables quick evaluation of new interaction modalities, and hence the deployment of new modalities will be much quicker than before. The framework can be used as a customisable and easily modifiable platform for enabling multimodal interaction with a mobile device, where implicit context-based interaction can be seen as one interaction modality that can be combined with others.

The customization tool can be considered a generic profiling tool for specifying personal mobile device functionality. By applying the results of this dissertation, the use of mobile devices can be made more personal, potentially more efficient, and better suited to varying individual user needs, which may change over time.

About thirty different real-world mobile phone application features were created and tested using the context framework and the customization tool. Some of those applications and application enhancements were selected for the evaluation. The scope of the applicability of the framework and the tool is wide.

The context-awareness usability and usefulness viewpoint is not within the focus of this dissertation. However, two separate user studies, for seven and ten people, were arranged to evaluate the usability and usefulness of the customization tool. After applying the tool, the users were asked: "Do you feel you would benefit from the customization tool?" Fifteen of the seventeen participants answered affirmatively. Korpipää et al. (2005b) and Häkkilä et al. (2005) give a more detailed discussion of the tool usability.

The results of this dissertation can be utilised by mobile application developers, mobile phone end-users, and product developers. Application developers can directly utilise the results of this dissertation by programming applications with the API provided by the context framework. End-users can directly apply the results for freely personalising their smart phones using the customization tool, if the software is released for public use. Moreover, product developers can potentially use Context Studio for defining platform- or application-specific user interaction or context-based features, and package the features as scripts with context framework into different products, expediting deployment.

The results of this dissertation have significance for industrial utilisation and commercial value. The results are being further developed for application in mobile phone- and mobile operator-related industries for application domains such as enhanced usability and personalization, novel sensor-based interaction modalities, mobile workforce, context-based security for enterprises, and context-based multimedia management.

10.4 Future work

In this chapter a few topics and challenges are pointed for future research.

1. Multimodal user interfaces

The context framework enables the use of multiple modalities for controlling mobile devices. In addition to implicit inputs, the framework enables the use of explicit control commands, such as gestures and RFID tags. The user viewpoint of utilising these new modalities in mobile device control, and for the control of external appliances with the mobile device, should be studied further.

2. Usability of context-aware applications

An analysis of the usefulness and usability of context-aware applications is required to examine the practical implications of context awareness for mobile computing in general. Studies should especially analyse how context-aware features are used and affect the normal daily lives of the users.

3. Automated creation of user profiles

It was shown in this dissertation that end-user development of context-aware application features is possible. The user can customize the context-aware features manually, after which the device functionality is as the user specified it in a rule, which is a kind of user profile. If the circumstances change, the user can re-customize the device functionality as required.

A challenge still remains concerning the automated creation and adaptation of the user profile based on automated monitoring of the behaviour of the user. However, fully automated adaptation has many problems, such as how to automatically choose the relevant inputs for the actions when learning actions from a set of inputs. The user should give feedback to the system, which creates another challenge of how this feedback should be given.

4. Battery consumption from continuous monitoring of sensors

In practical mobile computing, battery consumption is a critical measure, and no extra actions should be made without a clear need or advantage gained. Continuous monitoring of sensors for implicit input consumes battery power, and the advantages are not always as clear as the cost. In a basic case of context awareness, the application actions are invoked by the context instances abstracted from the sensor data. However, continuous monitoring is not always necessary. The application may initiate the context data acquisition. For example, in image augmentation context acquisition can be started when the camera is started and a snapshot of the context space is augmented in the image. Moreover, the abstracting and change detection processes, producing events from the continuous stream, should be performed as close to the hardware as possible, to reach the lowest possible power consumption. Ideally, the sensor hardware should have a quick wake-up time from standby and contain low power processing capability to directly output abstracted events, to be added to the Context Manager blackboard.

5. Utilising contexts recognized from multi-sensor data

There are many limitations for the rapid utilisation of a wide variety of contexts derived from multi-sensor data. More sensors and a large data collection are needed to gain enough information to discriminate reliably between contexts in a general mobile device usage setting. Some contexts are additionally ambiguous or subjective, and, as such, application-specific. Challenges in reaching generalisable and reliable multi-sensor context recognition with mobile devices still remain remarkable and practical applicability is unclear.

6. Documenting and sharing context and action vocabularies

Expanding the set of contexts that can be used by the applications leads to a wider scope of applications. Sharing and communicating context information by using ontologies promotes the availability of the context vocabularies. Providing tools for creating formal vocabularies for sharing, possibly using Semantic Web technologies, should be studied further, but with consideration of the special characteristics of mobile computing.

In current implementation, the context-action rules and context instances are formally described with CEP for sharing purposes. The vocabularies that list the available contexts and actions have semi-informal representation for rapid utilisation. The use of a formal expressive language for describing large context and action vocabularies would bring advantages such as describing context value constraints, enabling centralised context information consistency checking and systematic conflict handling. The possible drawback is a decreased human readability of the information, unless proper editors are available for creating the vocabularies. Furthermore, the ontology vocabulary should serve as a general documentation of context and action information for application developers and end users, who both need to know what kind of information they use.

7. New customizable modalities

New customizable modalities, such as gestures, physical selection and implicit inputs, potentially increase the efficiency of human-computer interaction with mobile devices. Further work includes quantitatively evaluating the average improved efficiency of using these new customizable modalities. Furthermore, the user feedback has suggested that, in addition to trainable gestures, customizable context sources could be useful.

8. Using mobile device as a control interface for external devices

The interaction convergence to one device continues, and an increasing number of different external devices will become controllable with a mobile phone in the near future. Providing a uniform user interface for using external devices with heterogeneous interfaces is a challenge. The Context Framework and customization tool could be further developed to flexibly facilitate a wider scope of customizing user interaction with external devices.

9. Customization tool

Using a customization tool is easy to learn since it has a reduced rule expression. However, even simple rules do not guarantee complete user control if the user is totally unfamiliar with what a certain trigger does – for example, a gesture named Square received from a friend. The customization tool should provide a wider explanation of the exact meaning of the trigger – for example, as a help function – possibly with a visualization. This explanation should be documented in the ontology vocabulary, read by the Customizer.

Furthermore, the users might want to define and name their own context triggers – e.g., locations, social situations based on Bluetooth devices, temperature abstractions, light levels, etc. Further work includes providing a graphical user interface for personalizing specific Context Sources, and the ability to update the vocabularies at runtime. Currently, the user can train and name the gesture-type triggers, and name RFID tags by physically writing the tags with the RFID tag read/write accessory.

Moreover, it should be studied whether other kind of user interaction, for example Wizard-style, in creating and managing rules could be even more user-friendly.

10. Application Controller

For maximal flexibility of application control with the context framework, the applications to be controlled should have the capability of dynamically registering their controls to the Application Controller. Some phone applications, such as calling, profiles, etc., require an application-specific function call interface, rather than a platform-level interface. It should be possible to plug in these function call interfaces to the Activator. The plug-in action interfaces should be able to check whether the corresponding action function exists in a current instantiation.

Dynamical actions registering requires a common action language and vocabulary that defines both the machine readable function parameters for calling the application and the human understandable action expression that can be used for customization. Currently the Customizer reads this action vocabulary configuration at startup. The generated CEP rule scripts hence contain the human understandable expression and function parameters. CEP scripts can be subscribed to the Application Controller, which at inference time uses the function parameters in the script to execute application fucntions. As a further work then, ideally, the new application or new external device could dynamically plug in its available functions to the Application Controller. Basically the application action plug-in would state "These are my function calls, and these are the names for them for the user to understand", facilitating wide-ranging mobile end user development.

11. Summary

The research problems concerned the development of a context framework and tool for mobile device context awareness. The first problem was stated as what is required to flexibly and efficiently handle all relevant aspects of sensor-based mobile terminal-centric management of context-related information. The second problem concerned how to represent the context information so that it could be systematically processed, stored, used by the applications, and understood by the application developers, while maintaining representation extensibility. The third problem was stated as how can context be recognised and abstracted online into a common representation from many different sources, especially device sensors, producing possibly incomplete and imprecise information. The fourth problem concerned what kind of application programming interface is required for the simplified development of context-aware applications, and, further, what kind of tool would be required for end-user development in mobile handheld devices. Following the problems, the main hypothesis was set as follows:

By solving research problems 1–4 it will be possible to create a functional software framework and tool that enable end-users to quickly customize versatile contex-aware applications in a mobile device.

The research problems were answered and the hypothesis was verified as follows. The created software framework was applied to developing and using context-aware applications in a mobile phone. The framework contained the reusable elements of the context-aware applications. With the implemented framework, multiple context-based features were developed without programming changes to the framework code. The framework transparently handled acquiring, storing, abstracting and delivering the context information with a publish and subscribe mechanism and a database. The framework could perform application control based on any context events. Existing applications were enhanced with context-aware features without programming executable code. The framework provided an application programming interface, which was used for multiple applications. An extensible context ontology was created and used within the framework for representing the abstracted context information. A context framework and a customization tool facilitated end-user development.

The implemented context framework, ontology, abstracting and recognition, and customization tool were tested and evaluated in handheld mobile devices having a Symbian operating system. The evaluation had the following basis. The requirements were set for answering the research problems. The implementation of the framework was evaluated against the requirements. The fulfilled requirements verified the implementation. The use of the features specified in the requirements in different types of example applications verified the applicability and scope of the framework. The framework computing performance was evaluated in target hardware through usage and numerical measurements. The evaluation verified that the framework offers a generic scalable high performance platform for abstracting and managing sensor-based information, and other abstractions, for enabling context event-based application control. The ontology has the ability to express a wide variety of different types of abstractions. Context recognition, continuous concurrent online abstracting of multiple context types, and the use of the abstracted contexts as events for application control are feasible within the framework. The Customizer, the ontology and the context framework together facilitate simple and effortless customization of context-aware features into mobile device applications.

The related results have been published in international scientific journals and conferences, and a workshop. A total of 12 closely related and co-authored articles have been published, of which four are scientific journal articles, seven are scientific conference articles, and one workshop article. The author of the dissertation is the first author in seven of those publications, including three journal articles. In addition, three related and co-authored patents have been either published or a US patent is pending.

The customization tool can be considered a generic profiling tool for specifying personal mobile device functionality. By applying the results of this dissertation, the use of mobile devices can be made more personal, potentially more efficient, and better suited to changing user needs. In two user studies, 88 percent of the 17 users announced that they would benefit from the system.

The results of this dissertation have significance for industrial utilisation and commercial value. The results are being further developed for application in mobile phone and mobile operator-related industries for application domains such as enhanced usability and personalization, novel sensor-based interaction modalities, mobile workforce, context-based security for enterprises, and context-based multimedia management.

References

Aamodt, A. & Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications, Vol. 7, No. 1, pp. 39–59.

Ahmed, K., Ayers, D., Birbeck, M., Cousins, J., Dodds, D., Lubell, J., Nic, M., Rivers-Moore, D., Watt, A., Worden, R. & Wrightson, A. 2001. Professional XML meta data. Wrox Press. 567 p.

ARM. 2005. ARM products & solutions.
Available: http://www.arm.com/products/CPUs/families/ARM9Family.html.

Bao, L. & Intille, S. 2004. Activity recognition from user-annotated acceleration data. Proceedings of International Conference on Pervasive Computing, LNCS 3001. Pp. 1–17.

Bechhofer, S., Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P. & Stein, L. 2003. OWL Web Ontology Language reference, W3C proposed recommendation 15 December 2003.
Available: http://www.w3.org/TR/2003/PR-owl-ref-20031215.

Bellotti, V. & Edwards, K. 2001. Intelligibility and accountability: Human considerations in context-aware systems. Human-Computer Interaction, Vol. 16, No. 2, 3 & 4, pp. 193–212.

Berners-Lee, T., Hendler, J. & Lassila, O. 2001. The Semantic Web. Scientific American, Vol. 284, No. 4, pp. 34–43.

Brickley, D., Guha, R.V. & McBride, B. (eds.). 2004. RDF vocabulary description language 1.0: RDF schema, W3C recommendation 10 February 2004. Available: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

Brown, P., Bovey, J. & Chen, X. 1997. Context-aware applications: from the laboratory to the marketplace. IEEE Personal Communications, Vol. 4, No. 5, pp. 58–64.

Bray, T., Paoli, J., Sperberg-McQueen, C.M. & Maler, E. (eds.). 2000. Extensible Markup Language (XML) 1.0 (Second Edition), W3C recommendation 6 October 2000. Available: http://www.w3.org/TR/2000/REC-xml-20001006.

Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. & Wilkinson, K. 2003. Jena: Implementing the Semantic Web recommendations, technical report HPL-2003-146, Hewlett Packard Laboratories Bristol. Available: http://www.hpl.hp.com/techreports/2003/HPL-2003-146.pdf.

Castro, P. & Muntz, R. 2000. Managing context data for smart spaces. IEEE Personal Communications, Vol. 7, No. 5, pp. 44–46.

Chen, H., Finin, T. & Joshi, A. 2003. Semantic web in a pervasive context-aware architecture. Proceedings of Ubicomp'03 Workshop on Artificial Intelligence in Mobile Systems. Pp. 33–40.

Clarkson, B. & Pentland, A. 1998. Extracting context from environmental audio. Proceedings of International Symposium on Wearable Computers. Pp. 154–155.

Clarkson, B., Mase, K. & Pentland, A. 2000. Recognizing user context via wearable sensors. Proceedings of International Symposium on Wearable Computers. Pp. 69–76.

Connolly, D., Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P. & Stein, L. 2001. DAML+OIL (March 2001) reference description, W3C Note 18 December 2001. Available: http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218.

Cresswell, M. 1973. Logics and languages. Harper & Row Publishers. 273 p.

Crowley, J., Coutaz, J., Rey, G. & Reignier, P. 2002. Perceptual components for context awareness. Proceedings of International Conference on Ubiquitous Computing, Springer-Verlag. Pp. 117–134.

Cox, E., Taber, R., O'Hagan, M. & O'Hagen, M. 1998. The fuzzy systems handbook, second edition. AP Professional. 713 p.

Dey, A. 2000. Providing architectural support for building context-aware applications. Ph.D. dissertation, Georgia Institute of Technology. 170 p.

Dey, A. & Abowd, G. 2000. Towards a better understanding of context and context awareness. Proceedings of What, Who Where, When and How of Context-Awareness Workshop in CHI2000 ACM Conference on human factors in computer systems.

Dey, A., Abowd, G. & Salber, D. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, Vol. 16, No. 2, 3 & 4, pp. 97–166.

Dey, A., Hamid, R., Beckmann, C., Li, I. & Hsu, D. 2004. a CAPpella: programming by demonstration of context-aware applications. Proceedings of International Conference on Computer-Human Interaction, CHI 2004, ACM. Pp. 33–40.

Digia. 2003. Programming for the Series 60 platform and Symbian OS. Wiley. 521 p.

Duda, R., Hart, P. & Stork, D. 2001. Pattern classification, second edition. John Wiley & Sons. 654 p.

Engelmore, R. & Morgan, T. (eds.). 1988. Blackboard systems. Addison–Wesley. 602 p.

Erickson, T. 2002. Some problems with the notion of context-aware computing. Communications of the ACM, Vol. 45, No. 2, pp. 102–104.

Fahy, P. & Clarke, S. 2004. CASS – Middleware for Mobile Context-Aware Applications. Proceedings of Mobisys 2004 Workshop on Context Awareness. Boston, Massachusetts, USA.

Fallside, D. (ed.). 2001. XML schema part 0: Primer, W3C recommendation, 2 May 2001. Available: http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/.

Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. & Uthurusamy, R. (Eds.) 1996. Advances in knowledge discovery and data mining. AAAI Press. 611 p.

Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A.G. & Mehandjiev, N. 2004. Meta-design: A manifesto for end-user development. Communications of the ACM, Vol. 47, No. 9, pp. 33–37.

Flanagan, J., Mäntyjärvi, J. & Himberg, J. 2002. Unsupervised clustering of symbol strings and context recognition. Proceedings of the IEEE Conference on Data Mining. Pp. 171–178.

Fox, A., Johanson, B., Hanrahan, P. & Winograd, T. 2000. Integrating information appliances into an interactive workspace. IEEE Computer Graphics & Applications, Vol. 20, No. 3, pp. 54–65.

Gomez-Perez, A., Fernandez-Lopez, M. & Corcho, O. 2003. Ontological engineering. Springer-Verlag. 403 p.

Greenberg, S. 2001. Context as a dynamic construct. Human-Computer Interaction, Vol. 16, No. 2, 3 & 4, pp. 257–268.

Gruber, T. 1993. A translation approach to portable ontology specification. Knowledge Acquisition, Vol. 5, No. 2, pp. 199–220.

Guarino, N. 1998. Formal ontology in information systems. Proceedings of International Conference on Formal Ontology in Information Systems. IOS Press, Amsterdam. Pp. 3–15.

Henricksen, K., Indulska, J. & Rakotonirainy, A. 2002. Modeling context information in pervasive computing systems. Proceedings of International Conference on Pervasive Computing, LNCS 2414, Springer-Verlag. Pp. 167–180.

Himberg, J., Mäntyjärvi, J. & Korpipää, P. 2001. Using PCA and ICA for exploratory data analysis in situation awareness. Proceedings of IEEE Conference on Multisensor Fusion and Integration in Intelligent Systems. Pp. 127–131.

Hirsh, H. & Hearst, M. 2000. AI's greatest trends and controversies. IEEE Intelligent systems, Vol. 15, No. 1, pp. 8–17.

Hong, J. & Landay, J. 2001. An infrastructure approach to context-aware computing. Human-Computer Interaction, Vol. 16, No. 2, 3 & 4, pp. 287–303.

Hull, R., Neaves, P. & Bedford-Roberts, J. 1997. Towards situated computing. Proceedings of International Symposium on Wearable Computers. Pp. 146–153.

Huttunen, M., Shelby, Z. & Hyyryläinen, J. 2003. Henkilökohtainen sääasema. Prosessori, November 2003, pp. 43–45.

Häkkilä, J., Korpipää, P., Ronkainen, S. & Tuomela, U. 2005. Interaction and end user programming with a context-aware mobile application. Proceedings of Human-Computer Interaction – Interact 2005, LNCS 3585, Springer-Verlag. Pp. 927–937.

IEEE Std 610.12-1990. 1990. Standard glossary of software engineering terminology. IEEE. 84 p.

IEEE Std 1471-2000. 2000. IEEE recommended practice for architectural description of software-intensive systems. IEEE. 29 p.

ISO 9241-11:1998. 1998. Ergonomic requirements for office work with visual display terminals (VTDs). Part 11: Guidance on usability. ISO. 22 p.

Kela, J., Korpipää, P., Mäntyjärvi, J., Kallio, S., Savino, G., Jozzo, L. & Di Marca, S. 2005. Accelerometer-based gesture control for a design environment. Personal and Ubiquitous Computing Journal special issue on Multimodal Interaction with Mobile and Wearable Devices, Springer-Verlag. In Press, published online. Available: http://www.springerlink.com.

Keränen, H., Rantakokko, T. & Mäntyjärvi, J. 2003. Sharing and presenting multimedia and context information within online communities using mobile terminals. Proceedings of International Conference on Multimedia and Expo, Vol. 2. Pp. 641–644.

Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M. & Tran, L. (eds.). 2003. Composite Capability/Preference Profiles (CC/PP): Structure and vocabularies 1.0, W3C proposed recommendation 15 October 2003. Available: http://www.w3.org/TR/2003/PR-CCPP-struct-vocab-20031015/.

Kofod-Petersen, A. & Aamodt, A. 2003. Case-based situation assessment in a mobile context-aware system. Proceedings of Ubicomp'03 Workshop on Artificial Intelligence in Mobile Systems. Pp. 41–48.

Kohonen, T. 2001. Self-organising maps, third extended edition. Springer-Verlag. 501 p.

Korpipää, P., Häkkilä, J., Malm, E., Rantakokko, T., Kyllönen, V., Kela, J., Känsälä, I. & Mäntyjärvi, J. 2005a. Enabling user interaction customization in smart phones. IEEE Pervasive Computing Magazine. In Press.

Korpipää, P., Malm, E., Salminen, I., Rantakokko, T., Kyllönen, V. & Känsälä, I. 2005b. Context management for end-user development of context-aware applications. Proceedings of International Conference on Mobile Data Management MDM'05. Pp. 304–308.

Korpipää, P., Häkkilä, J., Kela, J., Ronkainen, S. & Känsälä, I. 2004a. Utilising context ontology in mobile device application personalisation. Proceedings of International Conference on Mobile and Ubiquitous Multimedia, ACM. Pp. 133–140.

Korpipää P., Pärkkä, J. & Cluitmans L. 2004b. Representing features and contexts in a data library. Proceedings of Pervasive Computing Conference Workshop: Benchmarks and Database for Context Recognition. Pp. 32–37.

Korpipää, P., Koskinen, M., Peltola, J., Mäkelä, S.-M. & Seppänen, T. 2003a. Bayesian approach to sensor-based context awareness. Personal and Ubiquitous Computing Journal, Vol. 7, No. 4, pp. 113–124.

Korpipää, P. & Mäntyjärvi, J. 2003. An ontology for mobile device sensor-based context awareness. Proceedings of International and Interdisciplinary Conference on Modeling and Using Context, LNAI no. 2680, Springer-Verlag. Pp. 451–459.

Korpipää, P., Mäntyjärvi, J., Kela, J., Keränen, H. & Malm, E.-J. 2003b. Managing context information in mobile devices. IEEE Pervasive Computing Magazine, July–September special issue: Dealing with Uncertainty, Vol. 2, No. 3, pp. 42–51.

Korpipää, P. 2001. Visualising constraint-based temporal association rules. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Vol. 15, pp. 401–410.

Kumar, S., Cohen, P. & Levesque, H. 2000. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. Proceedings of international conference on multi-agent systems. Pp. 159–166.

Kurki, M., Taipale, T. & Korpipää, P. 1998. A large scale fault diagnosis system for a hot strip mill. Proceedings of IFAC Symposium on Automation in Mining, Mineral and Metal Processing, IFAC.

Laerhoven, K. & Cakmakci, O. 2000. What shall we teach our pants? Proceedings of International Symposium on Wearable Computers. Pp. 77–83.

Laerhoven, K. & Aidoo, K. 2001. Teaching context to applications. Personal and Ubiquitous Computing Journal, Vol. 5, pp. 46–49.

Lakkala, H. 2003a. Context exchange protocol specification. 28 p.
Available: http://www.mupe.net.

Lakkala, H. 2003b. Context script specification. 22 p.
Available: http://www.mupe.net.

Larman, C. 2002. Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process. Prentice Hall.

Lenat, D. 1995. CYC: A large-scale investment in knowledge infrastructure. Communications of the ACM, Vol. 38, No. 11, pp. 33–41.

Lenat, D., Miller, G. & Yokoi, T. 1995. CYC, Wordnet, and EDR: Critiques and responses. Communications of the ACM, Vol. 38, No. 11, pp. 45–48.

Lenat, D. 1998. The dimensions of context-space. Cycorp report. Available: http://www.cyc.com. 78 p.

Lenat, D. 1999a. Features of CycL. Cycorp report.
Available: http://www.cyc.com.

Lenat, D. 1999b. The upper Cyc ontology. Cycorp report.
Available: http://www.cyc.com.

Lukowicz, P., Ward, J., Junker, H., Stäger, M., Tröster, G., Atrash, A. & Starner, T. 2004. Recognizing workshop activity using body worn microphones and accelerometers. Proceedings of International Conference on Pervasive Computing, LNCS 3001. Pp. 18–32.

Mandato, D., Kovacs, E., Hohl, F. & Amir-Alikhani, H. 2002. CAMP: A context-aware mobile portal. IEEE Communications magazine, Vol. 40, No. 1, pp. 90–97.

Manola, F., Miller, E. & McBride, B. (ed.). 2004. RDF primer, W3C recommendation 10 February 2004.
Available: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

McGuinness, D. & Harmelen, F. (eds.) 2003. OWL Web Ontology Language overview, W3C proposed recommendation 15 December 2003. Available: http://www.w3.org/TR/2003/PR-owl-features-20031215.

Miller, L., Seaborne, A. & Reggiori, A. 2002. Three implementations of SquishQL, a simple RDF query language. Proceedings of International Semantic Web Conference, LNCS 2342, Springer-Verlag. Pp. 423–435.

Minar, N., Gray, M., Roup, O., Krikorian, R. & Maes, P. 2000. Hive: Distributed agents for networking things. IEEE Concurrency, Vol. 8, No. 2, pp. 24–33.

Minsky, M. 1988. The society of mind. Simon & Schuster. 339 p.

Minsky, M. 2000. Commonsense-based interfaces. Communications of the ACM, Vol. 43, No. 8, pp. 67–73.

Mitchell, T. 1997. Machine learning. McGraw–Hill. 414 p.

Mitchell, K. 2002. Supporting the development of mobile context-aware Systems. Ph.D dissertation, Lancaster University. 224 p.

Mäntylä, V.-M. 2001. Discrete hidden Markov models with application to isolated user-dependent hand gesture recognition. Espoo: VTT Publications 449, 104 p. http://www.vtt.fi/inf/pdf/publications/2001/P449.pdf

Mäntyjärvi, J., Kela, J., Korpipää, P. & Kallio, S. 2004. Enabling fast and effortless customization in accelerometer based gesture interaction. Proceedings of International Conference on Mobile and Ubiquitous Multimedia (MUM), ACM. Pp. 25–31.

Mäntyjärvi, J. 2003. Sensor-based context recognition for mobile applications. Ph.D dissertation. Espoo: VTT Publications 511. 118 p. + app. 60 p. http://www.vtt.fi/inf/pdf/publications/2003/P511.pdf

Mäntyjärvi, J., Tuomela, U., Häkkilä, J. & Känsälä, I. 2003. Context-Studio – tool for personalizing context-aware applications in mobile terminals. Proceedings of Australasian Computer Human Interaction Conference, University of Queensland, Brisbane Australia. Pp. 64–73.

Mäntyjärvi, J., Huuskonen, P. & Himberg, J. 2002. Collaborative context determination to support mobile terminal applications. IEEE Wireless Communications, Vol. 9, No. 5, pp. 39–45.

Mäntyjärvi, J. & Seppänen, T. 2002. Adapting applications in mobile terminals using fuzzy context information. Proceedings of International Symposium on Mobile Human Computer Interaction, LNCS 2411, Springer-Verlag. Pp. 95–107.

Mäntyjärvi, J., Himberg, J., Korpipää, P. & Mannila, H. 2001. Extracting the context of a mobile device user. Proceedings of IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems. Pp. 445–450.

Myllymäki, P. & Tirri, H. 1998. Bayes-verkkojen mahdollisuudet. National Technology Agency, Finland. 110 p.

OMA 2003. User agent profile (UAProf) specification, version 2.0, May 2003. Available: http://www.openmobilealliance.org/release_program/uap_v20.html.

Ouellet, R. & Ogbuji, U. 2002. Introduction to DAML. XML.com Web Publication. Available: http://www.xml.com/pub/a/2002/01/30/daml1.html.

Pascoe, J. 1998. Adding generic contextual capabilities to wearable computers. Proceedings of International Symposium on Wearable Computers. Pp. 92–99.

Pascoe, J., Ryan N. & Morse D. 1999. Issues in developing context-aware computing. Proceedings of International Symposium on Handheld and Ubiquitous Computing, LNCS 1707, Springer-Verlag. Pp. 208–221.

Pearl, J. 1988. Probabilistic reasoning in intelligent systems, revised second printing. Morgan Kaufmann Publishers, San Francisco. 552 p.

Peltonen, V., Tuomi, J., Klapuri, A., Huopaniemi, J. & Sorsa, T. 2002. Computational auditory scene recognition. Proceedings of International Conference on Acoustics, Speech and Signal Processing, IEEE. Pp. 1941–1944.

Peltonen, V., Eronen, A., Parviainen, M. & Klapuri, A. 2001. Recognition of everyday auditory scenes: potentials, latencies and cues. Proceedings of 110th Audio Engineering Society Convention, Amsterdam, Netherlands.

Picard, R. 1998. Affective computing. The MIT Press. 292 p.

Pokraev, S., Koolwaaij, J. & Wibbels, M. 2003. Extending UDDI with context-aware features based on semantic service descriptions. Proceedings of International Conference on Web Services. Pp. 184–190.

Pree, W. 1994. Meta-patterns – a means for capturing the essentials of reusable object-oriented design. Proceedings of ECOOP, Springer-Verlag, pp. 150–162.

Pyle, D. 1999. Data preparation for data mining. Morgan Kaufmann. 540 p.

Rabiner, L. & Juang, B.-H. 1993. Fundamentals of speech recognition. Prentice Hall. 507 p.

Raggett, D., Hors, A. & Jacobs, I. (eds.). 1999. HTML 4.01 specification, W3C recommendation 24 December 1999.
Available: http://www.w3.org/TR/1999/REC-html401-19991224.

Ranganathan, A. & Campbell, R. 2003. An infrastructure for context-awareness based on first order logic. Personal and Ubiquitous Computing Journal, Vol. 7. No. 6, December, pp. 353–364.

Riekki, J., Huhtinen, J., Ala-Siuru, P., Alahuhta, P., Kaartinen, J. & Röning, J. 2003. Genie of the net, an agent platform for managing services on behalf of the user. Computer Communications, Vol. 26, No. 11, pp. 1188–1198.

Rodden, T., Cheverst, K., Davies, K. & Dix, A. 1998. Exploiting context in HCI design for mobile systems. Workshop on Human Computer Interaction with Mobile Devices.

Russell, S. & Norvig, P. 1995. Artificial intelligence: A modern approach. Prentice Hall. 931 p.

Schilit, B., Adams, N. & Want, R. 1994. Context-aware computing applications. Proceedings of International Workshop on Mobile Computing Systems and Applications. IEEE Computer Society. Pp. 85–90.

Schilit, B. 1995. System architecture for context-aware mobile computing. Ph.D. dissertation, Columbia University, New York. 144 p.

Schmidt, A. 2002. Ubiquitous computing – computing in context. Ph.D dissertation, Lancaster University. 294 p.

Schmidt, A. 2000. Implicit human computer interaction through context. Personal Technologies, Vol. 4, No. 2 & 3, pp. 191–199.

Schmidt, A., Takaluoma, A. & Mäntyjärvi, J. 2000. Context-aware telephony over WAP. Personal Technologies, Vol. 4, No. 4, pp. 225–229.

Schmidt, A., Aidoo, K.A., Takaluoma, A., Tuomela, U., Laerhoven, K. & Van de Velde, W. 1999a. Advanced interaction in context. Proceedings of International Symposium on Handheld and Ubiquitous Computing. LNCS 1707, Springer-Verlag. Pp. 89–101.

Schmidt, A., Beigl, M. & Gellersen, H.-W. 1999b. There is more to context than location. Computers&Graphics, Vol. 23, No. 6, pp. 893–902.

Series 60. 2005. Series 60 platform. Available: http://www.series60.com/.

Smith, M., Welty, C. & McGuinness, D. (eds.) 2003. OWL Web Ontology Language guide, W3C proposed recommendation 15 December 2003. Available: http://www.w3.org/TR/2003/PR-owl-guide-20031215.

Sohn, T. & Dey, A. 2003. iCAP: An informal tool for interactive prototyping of context-aware applications. Interactive poster in the extended abstracts of ACM Conference on Human Factors in Computing Systems. Pp. 974–975.

Studer, R., Benjamins, V. & Fensel, D. 1998. Knowledge engineering: Principles and methods. IEEE Transactions on Data and Knowledge Engineering, Vol. 25, No. 1 & 2, pp. 161–197.

Suomela, R., Räsänen, E., Koivisto, A., Mattila, J. & Koskinen, T. 2003. Multi-User Publishing Environment (MUPE) application platform. 17 p. Available: http://www.mupe.net.

Tasker, M., Allin, J., Dixon, J., Forrest, J., Heath, M., Richardson, T. & Shackman, M. 2000. Professional Symbian Programming. Wrox Press. 1031 p.

Theodoridis, S. & Koutroumbas, K. 1999. Pattern recognition. Academic Press. 625 p.

Toivonen, S., Kolari, J. & Laakko, T. 2003. Facilitating mobile users with contextualized content. Proceedings of Ubicomp'03 Workshop on Artificial Intelligence in Mobile Systems. Pp. 124–134.

Truong, K., Huang, E. & Abowd, G. 2004. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. Proceedings of Ubicomp. Pp. 143–160.

Tuulari, E. 2000. Context aware hand-held devices. Espoo: VTT Publications 412. 82 p. http://www.vtt.fi/inf/pdf/publications/2000/P412.pdf

UML. 2005. UML resource page. Available: http://www.uml.org/.

UPnP 2005. UPnP forum. Available: http://www.upnp.org/.

Vildjiounaite, E., Malm, E.J., Kaartinen, J. & Alahuhta, P. 2003. Context awareness of everyday objects in a household. Proceedings of European Symposium on Ambient Intelligence, LNCS 2875, Springer-Verlag. Pp. 177–191.

W3C Semantic Web. 2001. W3C Semantic Web document. Available: http://www.w3.org/2001/sw/.

Wang, X., Dong, J., Chin, C., Hettiarachchi, S. & Zhang, D. 2004. Semantic space: An infrastructure for smart spaces. IEEE Pervasive Computing Magazine, July–September, pp. 32–39.

Ward, A., Jones, A. & Hopper, A. 1997. A new location technique for the active office. IEEE Personal Communications, Vol. 4, No. 5, pp. 42–47.

Wikipedia. 2005. Wikipedia, the free encyclopedia: Software framework. Available: http://en.wikipedia.org/wiki/Software_framework.

Winograd, T. 2001. Architectures for context. Human-Computer Interaction, Vol. 16, No. 2, 3 & 4, pp. 401–419.

XML Schema repository. 2001. W3C XML schema repository. Available: http://www.w3.org/XML/Schema.

Yau, S. & Karim, F. 2001. Context-sensitive middleware for real-time software in ubiquitous computing environments. Proceedings of IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. Pp. 163–170.

Zadeh, L. 1965. Fuzzy sets. Information and control, Vol. 8, pp. 338–353.

Zadeh, L. 1996. Fuzzy logic = computing with words. IEEE Transactions on Fuzzy Systems, Vol. 4, No. 2, pp. 103–111.

Published by

VTT

Series title, number and report code of publication

VTT Publications 579
VTT–PUBS–579

Author(s) Korpipää, Panu	

Title

Blackboard-based software framework and tool for mobile device context awareness

Abstract

The usage needs of a mobile device vary according to context. Mobile context awareness research aims at providing the device user with a way of usage that suits the situation. Interaction based on context requires acquiring, abstracting and delivering information from multiple sources, such as sensors, to the application or application control. A generic software framework and tool for facilitating the rapid development of mobile device context-aware applications were developed in this work. The blackboard-based framework supports all tasks that are required for context-based application control, where contexts can be any events that are relevant to user interaction with the application, including explicit inputs. The core component of the framework, Context Manager, provides a publish and subscribe mechanism and a database for the applications and application control. The framework provides an application programming interface (API) for developers. As a higher abstraction-level programming interface, a customization tool enables easy end-user development of context-aware features into existing applications without changing them.

An extensible ontology is used as a uniform context representation within the framework. The purpose of the ontology, together with the API, is to enable easy access, use and reuse of human-understandable context information. Context information sources, such as sensors, often produce a continuous stream of low abstraction-level data. The framework supports the transformation of a continuous data stream into abstracted context events, described in the ontology. Context information is delivered to applications or application control as abstracted events. The main result of the dissertation is a software framework, ontology and tool, which facilitate the customization of sensor-based human-computer interaction in mobile devices. The practical applicability, scope, and computational efficiency of the implemented framework and customization tool are evaluated with performance measurements and multiple applications implemented in a mobile phone with real sensor sources.

Keywords

mobile computing, context-aware computing, mobile interaction, mobile context awareness, application control, blackboard-based architecture, software framework, context management, information model, application programming interface, sensor-based interaction, customization, end-user development, personalization

Activity unit

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FI–90571 OULU, Finland

ISBN 951–38–6669–6 (soft back ed.) 951–38–6670–X (URL:http://www.vtt.fi/inf/pdf/)			Project number
Date October 2005	Language English	Pages 225 p.	Price E
Name of project		Commissioned by	
Series title and ISSN VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)		Sold by VTT Information Service P.O.Box 2000, FI–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374	

VTT PUBLICATIONS 579   Blackboard-based software framework and tool for mobile device context awareness   Panu Korpipää

Mobile context awareness research aims at providing the mobile device user with a way of usage that suits the situation. New input sources, such as embedded sensors producing interaction-related information, are becoming available for mobile devices. These input sources enable novel ways of interacting with the devices, and even open possibilities to create entirely new types of applications. To facilitate the full potential of utilising such new input sources, a software framework is required with a uniform means of acquiring and processing interaction-related information, and providing it for the applications. The main result of this dissertation was a software framework and tool for facilitating the rapid development of mobile device context-aware applications. The framework provides a publish and subscribe mechanism, database, and a customisable application controller. For developers the framework provides an application programming interface. The customization tool enables end-user development of interaction-related features in mobile devices. The results have commercial value; they are utilised by the telecommunication industry for application domains such as enhanced usability and personalization, novel sensor-based interaction modalities, mobile workforce, context-based security for enterprises, and context-based multimedia management.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU	VTT INFORMATIONSTJÄNST	VTT INFORMATION SERVICE
PL 2000	PB 2000	P.O.Box 2000
02044 VTT	02044 VTT	FI–02044 VTT, Finland
Puh. 020 722 4404	Tel. 020 722 4404	Phone internat. +358 20 722 4404
Faksi 020 722 4374	Fax 020 722 4374	Fax +358 20 722 4374

ISBN 951–38–6669–6 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6670–X (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)