An Architecture for Context Prediction

DISSERTATION

zur Erlangung des akademischen Grades

DOKTOR DER TECHNISCHEN WISSENSCHAFTEN

im

DOKTORATSSTUDIUM DER TECHNISCHEN WISSENSCHAFTEN

Angefertigt am *Institut für Pervasive Computing*

Eingereicht von:

Dipl.-Ing. Rene Michael Mayrhofer

Betreuung:

o. Univ.-Prof. Mag. Dr. Alois Ferscha

Beurteilung:

o. Univ.-Prof. Mag. Dr. Alois Ferscha
Prof. Dipl.-Ing. Dr. Hans-Werner Gellersen

Linz, Oktober 2004

Kurzfassung

Pervasive Computing ist eine neue, zunehmend an Bedeutung gewinnende Forschungsrichtung an der Berührungsstelle zwischen Benutzerinteraktion, eingebetteten und verteilten Systemen und Netzwerktechnologien. Das erklärte Ziel ist eine holistische Gestaltung von Computertechnologie, oft gleichgesetzt mit dem “Verschwinden” derselben in den Hintergrund des täglichen Lebens. Ein zentraler Aspekt bei der Verfolgung dieses Ziels ist die Reduktion expliziter, vereinnahmender Schnittstellen zur Mensch-Maschine Interaktion zugunsten impliziter, in Alltagsgegenständen verankerter. Während erstere die Aufmerksamkeit von Benutzern binden, erlauben letzere eine intuitive und unaufdringliche Benutzung. Diese dem Menschen eher entgegenkommende Form des Umgangs mit Computersystemen erfordert jedoch die Anpassung solcher Systeme an den jeweiligen Kontext, in dem sie betrieben werden. Kontext wird hierbei, wie in diesem Zusammenhang üblich, als die Ansammlung von für die Benutzerinteraktion relevanter Information über die aktuelle Situation einer Person, eines Ortes oder eines Objekts verstanden. Die sogenannte kontextbezogene Interaktion, manifestiert durch den Entwurf und die Konstruktion kontextsensitiver Systeme, ist daher einer der Grundpfeiler von Pervasive Computing. Im Zuge dieser wissenschaftlichen Stoßrichtung entstanden innerhalb der letzten fünf Jahre entscheidende Arbeiten zur Erkennung des aktuellen Kontext anhand verschiedenartigster Sensoren, die in Kombination miteinander ein möglichst breit gestreutes Modell der realen Welt liefern sollen.

In dieser Dissertation wird der nächste logische Schritt einer eingehenderen Betrachtung unterzogen: die Vorhersage zukünftiger Kontexte. Der generelle Ansatz ist dabei die Vorhersage abstrakter Kontexte mit dem Ziel, Computersysteme auf zukünftig eintretende Situationen proaktiv vorbereiten zu können. Diese Vorhersage von Kontext auf abstrakter Ebene erlaubt gegenüber der unabhängigen Vorhersage einzelner Aspekte — wie exemplarisch der geographischen Position des Benutzers — eine gemeinsame Betrachtung aller erfassbaren Aspekte. Dadurch wird eine Auswertung von Mustern und Zusammenhängen im Benutzerverhalten, die auf niedrigeren Betrachtungsebenen nicht erkennbar sind, ermöglicht. Die vorliegende Arbeit analysiert die Bedingungen für benutzergerechte Vorhersage von Kontext und entwickelt, aufbauend auf bereits bekannten Methoden zur datenbasierten Vorhersage wie beispielsweise den verschiedenartigen Ausprägungen von Markov Modellen, eine Architektur zur automatischen, im Hintergrund ablaufenden Kontexterkennung und -vorhersage. Besonderes Augenmerk wird dabei auf implizite Benutzerinteraktion ohne Unterbrechung der Benutzer während ihrer normalen Tätigkeiten und auf beständige Adaption der entwickelten Systeme aufgrund sich ändernder Rahmenbedingungen gelegt. Ein weiterer Aspekt ist der sparsame Umgang mit Ressourcen zur möglichen Integration von Kontextvorhersage in eingebettete Systeme. Die entwickelte Architektur wird anschließend in Form eines flexiblen Software-Frameworks umgesetzt und mit aufgezeichneten Daten aus alltäglichen Situationen evaluiert. Diese Betrachtung zeigt, dass die Vorhersage abstrakter Kontexte in Grenzen bereits möglich ist, jedoch noch Raum für Verbesserungen der Vorhersagequalität in zukünftigen Arbeiten offen bleibt.

Abstract

Pervasive Computing is a new area of research with increasing prominence; it is situated at the intersection between human/computer interaction, embedded and distributed systems and networking technology. Its declared aim is a holistic design of computer systems, which is often described as the disappearance of computer technology into the periphery of daily life. One central aspect of this vision is a partial replacement of explicit, obtrusive interfaces for human/computer interaction that demand exclusive user attention with implicit ones embedded into real-world artifacts that allow intuitive and unobtrusive use. This kind of interaction with computer systems suits human users better, but necessitates an adaption of such systems to the respective context in which they are used. Context is, in this regard, understood as any information about the current situation of a person, place or object that is relevant to the user interaction. Context-based interaction, which is pursued by the design and implementation of context-sensitive systems, is therefore one of the building blocks of Pervasive Computing. Within the last five years, a number of seminal publications on the recognition of current context from a combination of different sensors have been written within this field.

This dissertation tackles the next logical step after the recognition of the current context: the prediction of future contexts. The general concept is the prediction of abstract contexts to allow computer systems to proactively prepare for future situations. This kind of high-level context prediction allows an integral consideration of all ascertainable aspects of context, in contrast to the autonomous prediction of individual aspects like the geographical position of the user. It allows to consider patterns and interrelations in the user behavior which are not apparent at the lower levels of raw sensor data. The present thesis analyzes prerequisites for user-centered prediction of context and presents an architecture for autonomous, background context recognition and prediction, building upon established methods for data based prediction like the various instances of Markov models. Especial attention is turned to implicit user interaction to prevent disruptions of users during their normal tasks and to continuous adaption of the developed systems to changed conditions. Another considered aspect is the economical use of resources to allow an integration of context prediction into embedded systems. The developed architecture is being implemented in terms of a flexible software framework and evaluated with recorded real-world data from everyday situations. This examination shows that the prediction of abstract contexts is already possible within certain limits, but that there is still room for future improvements of the prediction quality.

Dedication

To my parents, for not telling but showing me what is important in life and being always there for me. You have made all of this possible. Thank you.

Acknowledgments

Without the support of a large number of people, writing this thesis would most probably have taken significantly longer and would definitely have been less enjoyable. I am especially grateful to:

- My supervisor Prof. Mag. Dr. Alois Ferscha for giving me the opportunity to work in his group and presenting me with a choice of interesting topics to work on. Although he seemed to be too busy to even have some private life, he always managed to be available for questions and offered useful suggestions without confining my research on this thesis.
- Harald Radi, for doing his diploma thesis on the implementation of the concepts presented in this thesis and becoming a good friend during this task. Many of the ideas, concepts, interfaces and ad-hoc solutions to intermediary issues were developed in close cooperation with him and major parts of the framework, including the complete Windows implementation, were completely done by him. He worked independently and with outstanding efficiency while producing excellent code quality. Without him, the implementation of our concepts would not have been possible in such a quality and extent.
- Prof. Dr. Hans-Werner Gellersen for reading this thesis as the second supervisor and giving valuable hints on various topics.
- Dipl. Inf. Dr. Albrecht Schmidt for many suggestions on the topic of context awareness and on writing a PhD thesis in general.
- Dipl.-Ing. Dr. Simon Vogl for sharing the office with me and supporting me in too many ways to list. He always had answers for implementation issues and useful hints for debugging obscure errors and reminded me of creating artificial test data before working on real-world data.
- Dipl.-Ing. Volker Christian, who is a steady source of information about UNIX-type systems and far more reliable then manual pages.
- Manfred Hechinger for helpful discussions on the general topic of proactivity and for relieving me of much project work during the time when I actually wrote this thesis.
- Prof. Dipl.-Ing. Dr. Günther Blaschek for many hints on various topics and discussions about the implications on user interfaces.
- Dipl.-Ing. Dr. Wolfgang Narzt for organizational support and sharing his insights on academic careers.
- Prof. Dipl.-Ing. Dr. Josef Scharinger for suggesting the used quality measure for optimizing the number of clusters for k-means and general encouragement towards an academic career.
- My partner Petra Wagner, who had to endure two years of long working hours while I was working on this thesis and always supported me emotionally in easy as well as hard times.
- All my friends and colleagues for making the time at university as enjoyable as it was.

Parts of this thesis have been published as [MRF03b, MRF03a, MRF04b, May04, MRF04a, RMF04] and I would like to thank the co-authors for helpful suggestions on improving certain sections, which also improved the presentation of the respective extended versions in this thesis.

Contents

Chapter 1

Introduction

Computing environments are changing rapidly, and the pace of this change is still increasing. Due to the broad availability of computing and network infrastructure, the potential audience of computing, communication or other services of informational nature is growing steadily. Within the last two decades, a number of well-established research fields in the area of computer science started to move the computing infrastructure beyond the desktop, establishing new ways of interacting with computer systems: *Embedded Systems* is concerned with the interaction between software and artifacts, i.e. real-world objects enriched with processing power, and tackles sensors and actuators, real time behavior, fault-tolerance, hardware design and autonomous operation. *Distributed Systems* concentrates on the interaction between software components which are executed concurrently on distributed processors and handles data distribution, synchronization and scalability issues. *Human/Computer Interaction* (HCI) applies psychological, ergonomic and technical means to improve the understandability and efficiency and reduce the error rate of user interfaces. *Software Engineering* attends to methodologies, architectures and tools for designing, implementing and maintaining software and therefore contributes directly to other fields of computer science. These different efforts of advancing computer science, to pervade everyday life and to foster wide availability and acceptance, yielded in 1991 Mark Weiser's seminal article [Wei91]. His vision of *Ubiquitous Computing* is that computer systems should disappear into the periphery of our daily lives, that computing infrastructure as a tool should be available everywhere, everytime, and to everybody.
However, at the time of his writing, the vision was ahead of time and available technology did not allow to realize it in an appropriate form. Only a decade later, technology and engineering had progressed far enough to allow its revival. The most important enabling technologies, namely wireless communication and miniaturization, had matured and were able so solve previously prohibitive issues, manifested by the emergence of industry standards like IEEE 802.11 wireless LAN in 1999 [IEE99] and Bluetooth® version 1.1 in 2001 [IEE02]. At the same time, the *International Symposium on Handheld and Ubiquitous Computing (HUC'99)* started a series of conferences, attracting broad attention and heralding a boom of Mark Weiser's general vision since then. It is still inspiring to a large number of researchers from different fields — ubiquitous computing, pervasive computing, ambient computing, calm computing, sentient computing, autonomous computing, hidden computing, invisible computing, disappearing computing and more terms have been coined over the last years, typically proclaiming a similar vision but with slightly different focus. Within the scope of this thesis, the term pervasive computing will be used, which is synonymous to ubiquitous computing but accentuates that real-life objects are equipped, or pervaded, with technology to make the whole world an interface to computing resources. Portray-

ing a new kind of computer systems, or of interacting with computer systems, research on pervasive computing is inherently inter-disciplinary. In addition to the cornerstone fields that initiated its advent, others like artificial intelligence (AI), robotics, information systems and databases, operating systems and communication technology facilitate further advances towards going beyond the desktop.
As a combination of aspects from this multitude of research fields, two major directions evolved: *information appliances* and *awareness*. Information appliances are special-purpose devices designed to perform a specific function, specialized in information, with the ability to share information with other appliances [Nor98]. Today, e.g. mobile phones, handheld computers, set-top boxes or embedded appliances in room or building installations can be seen as information appliances. The second major direction is to make devices *aware* of their environment, resembling the notion of awareness from social and psychological sciences. When computer systems become aware of the situation they are used in, they can adapt better to the user's needs and engage in more efficient interaction. In the field of CSCW (computer supported cooperative work), awareness also plays a pivotal role and has been defined in [DB92] as:

> *Awareness is an understanding of the activities of others, which provides a context for your own activity.*

In this sense, awareness is a key issue for supporting users during their daily work with computer systems, as only a dynamic adaptation to the task at hand will make computing environments user friendly and supportive. The combination of awareness with information appliances, or rather the implementation of awareness in information appliances became known as *context awareness* [SBG99], since a device should act within its current context of use, by being aware of the various aspects of its current environment. Context awareness is concerned with the situation a device or user is in, and with adapting applications to the current situation.
Many systems have been developed which are aware of the current user's or their own context, ranging from the long-established context sensitive help systems to the controversial "wizards" in current desktop computer applications and to recent research projects on context awareness going far beyond that. But knowing the current context an application or system is used in and dynamically adapting to it only allows to construct reactive systems, i.e. systems which run after changes in their environment. To maximize their usefulness and user support, systems should rather adapt in advance to a new situation and be prepared before they are actually used. This demands the development of proactive systems, i.e. systems which predict changes in their environment and act in advance. To this end, we strive to develop methods to predict future context, enabling systems to become proactive with regard to their context of use. Our concept is to provide software applications not only with information about the current user context, but also with predictions of future user context. When equipped with various sensors, a system should classify current situations and, based on those classes, learn the user's behaviors and habits by deriving knowledge from historical data. The focus of this thesis is to forecast future user context by extrapolating the past.

Judging from the current mobile phone and handheld developments, personal, mobile information appliances in various forms will be the primary digital companions and simultaneously the primary user interfaces for a large number of people within the next years. Therefore, we are currently focusing on embedded systems, which are self-contained computer systems with typically severely limited resources, such as memory or processing power. Methods and algorithms suitable for embedded systems can easily be used on more powerful computer systems, like an infrastructure behind a room or building, but not the other way around. We are thus concentrating on this more challenging class of computer systems. Additionally, information appliances need to be fully operational at all times and will need to be “trained” by their users themselves. To act according to the task at hand, they need to adapt to new and, at design-time, unforeseen situations in an online manner, without maintenance or update periods. This issue has been clearly articulated in a recent article [Dou04]:

> *[...] the major design opportunity concerns not use of predefined context within a ubiquitous computing system, but rather how can ubiquitous computing support the process by which context is continuously manifest, defined, negotiated and shared?*

As we move from one situation (context) to another, which does not necessarily need to correspond to the physical location, our computer systems must adapt and learn new patterns of behavior.
It should be pointed out that the topic of proactivity in computer science is a controversial one, especially in HCI; the general concept of context awareness itself has to be handled with care (cf. [Eri02, BD03]). Because the estimated current or predicted future context and thus implicitly the actions started by the appliance based on these assumption might be erroneous, they might need to be reverted by the user, possibly causing severe problems. Thus, proactivity in applications, when utilized for controlling actuators with impact on the real world, must be exerted cautiously. However, the possible uses for predicted user context in applications are manifold, ranging from simple reminder messages to emergency procedures being started before an accident happens. This thesis is primarily concerned with techniques that enable context prediction in embedded systems and leaves decisions about starting actions to applications built on top of it.

1.1 Challenges

Already in 1999, the area of context awareness was examined in [Lae99] with a similar focus on context awareness on embedded systems. This work laid ground for a separation of low-level sensor data from high-level context information via automatic classification (clustering) of sensor data. A list of problems has already been solved in this work, including a first review of suitable methods with focus on limited resources and the unobtrusive involvement of users by labeling abstract contexts. However, for the general case of making devices context aware, there is still a list of challenges to solve. A few general ones have already been defined in [Sch02a] and are still applicable to current research:

- *Understanding the concept of context.*
- *How to make use of context ?*
- *How to acquire context information ?*
- *Connecting context acquisition to context use.*
- *Understanding the influence on human computer interaction.*
- *Support for building context-aware ubiquitous computing systems.*
- *Evaluation of context-aware systems.*

These challenges occur in every research project dealing with context awareness and some of them have already been addressed by previous work, at least partially. Within this thesis, additional, more specific challenges for this research were identified, as already presented earlier in [May04]:

- Context recognition and prediction should be embedded in information appliances with limited resources. This limits the set of suitable methods and algorithms significantly.
- Learning and adaption should happen online without explicit training, which also prohibits the use of a number of powerful methods and algorithms that rely on batch training.
- User interaction should be kept to a minimum and be unobtrusive.
- Values gathered from typically available sensors are highly heterogeneous and thus many algorithms for statistical analysis and classification are not directly applicable.
- Prediction should respect trends and periodic patterns in context history with a reasonable trade-off between forecast accuracy and computational complexity.

Of these, the topic of context prediction is the most worthwhile but also the most challenging one.

1.2 Problem Statement

Our main focus of research is on *context prediction*. Context, in the field of pervasive computing, has been defined in different ways, and an overview of the different definitions can be found in Section 2.1.2. One of the first definitions of context in [Sch95] states that it comprises computing, user and physical properties. In 2000, this definition got refined by adding time as fourth aspect [CK00]. The definition adopted within this thesis is the one by Dey et.al. [DAS99], according to which context is *any information which can be used to characterize the situation of an entity*, where an entity is *a person, place or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves*. In [DA00], they suggested that, from the many different types of context that can possibly be taken into account, a few are more important than others: location, identity, activity and time, which they called primary context types for characterizing a situation. These four types are only slightly different from other definitions, but explicitly highlight the importance of the activity aspect in recognizing context. Building on this notion of context, the term context awareness denotes that a device incorporates context information into its behavior. Taking current context into account allows a device to react to changes in its environment, but not to act in advance, before a context change has been happening. We define *future context awareness* as incorporating future, predicted context into the device behavior and being prepared to future situations. This is also the focus of our work and we can refine the problem statement as follows:

> What are the necessary concepts, architectures and methods for context prediction in embedded systems?

Thus, the research goal is to evaluate and, if necessary, develop methods for recognizing and predicting context with the limited resources of embedded systems. An important aspect is that such a device must be fully operational at all times.

1.3 Scope and Methodology

It has already been pointed out in 1994, in one of the earliest publications on context awareness, that automatic reconfiguration is difficult and poses many problems [SAW94]:

> *Systems that reconfigure based on context are subject to the same problems faced by reconfigurable systems in general. In addition, if the context is changing rapidly it may be distracting to the user or impractical (due to performance) to adapt to every change. Also, certain adaptions may confuse users, particularly if the context is incorrectly reported, if the user is unaware of what context an application considers relevant, or if the context changes during use. Future work should address these issues.*

This thesis does also not address the general issue of reconfiguration, but concentrates on a basic infrastructure for providing high-level context information to applications. It is left to the applications what the context information is used for; in many cases, simply displaying the current context and adapting the user interface to it might be a better choice than automatically starting certain actions.
The general goal is to enhance interaction with mobile, small or otherwise limited devices in the context of pervasive computing. In this thesis, the main focus lies on forecasting user contexts and actions by deriving knowledge from historical data. Although the developed concept and architecture is applicable to arbitrary context-aware systems, this work is aimed towards personal information appliances, which are typically small, mobile devices with limited resources and which are used within the field of pervasive computing. Thus, real-world scenarios for evaluation are also taken from these application areas.

Research methods applied in this thesis are:

- Literature surveys and reviews.
- Qualitative comparison of classification and prediction methods and algorithms based on their declared and/or determined properties.
- Quantitative comparisons of methods and algorithms based on quality measures.
- Design of models and architectures based on requirements and results obtained from literature studies.
- Design and implementation of prototypes.
- Evaluation of concepts and architectures via prototype implementations.

1.4 Results

Within the scope of this thesis, a general concept for applying prediction methods to context-aware systems has been developed, based on the idea of interpreting abstract contexts as states, where a user or device advances from one state to another. By monitoring these state changes and analyzing the emergent context trajectory, it is possible to extrapolate into the future. This general concept has been refined into a flexible, multi-layered architecture for context prediction, combining methods and techniques from different fields of research. It is based on five steps: sensor data acquisition, feature extraction, classification, labeling and prediction. In these five steps, recognized current and predicted future high-level context information is inferred from low-level sensor data. To demonstrate its flexibility and suitability for solving the defined goals, it has been implemented in a module-based, cross-platform software framework, already ported to multiple architectures of embedded systems. The research goal of coping with limited resources in embedded systems has been considered while designing the architecture, resulting in an efficient implementation of the software framework.

For selecting appropriate methods and algorithms for the main steps in the architecture, namely classification and prediction, a literature survey has been conducted to narrow the list of possible candidates. Context recognition within the framework has already been fully implemented with an initial module for classification which is appropriate for a large class of context-aware systems. For achieving a higher classification accuracy in special application domains, additional classification modules can be easily implemented and are loaded dynamically by the framework. For context prediction, multiple prediction algorithms have been implemented and are readily available as modules within the framework. However, current results suggest to implement further modules to provide a broad range of available prediction methods. This will make it possible to select the most appropriate ones for specific applications.
An evaluation of the architecture and its implementation has been done with an extensive sensor data set collected in a realistic scenario. For both the classification and the prediction step, different methods have been compared quantitatively on the same data set to select the best algorithms for this scenario. Because not all of those algorithms are currently available as modules in the framework, some have been evaluated using Matlab and other available toolkits. The qualitative results of this evaluation demonstrate that it is possible to predict future context with reasonable device resources, but suggest further research to improve prediction accuracy in real-world scenarios. Context recognition data sets and a benchmark for context recognition are currently being collected in a coordinated effort by multiple research groups within pervasive computing. After completion, this benchmark should allow to effectively compare different algorithms and methods and should strengthen future research on context prediction.

1.5 Organization

The remainder of this thesis is split into six chapters. Chapter 2 introduces “context computing”, explaining the notion of context awareness and its aspects and giving a motivation for and a definition of proactivity in the area of context awareness. In Chapter 3, related work is summarized and this thesis is positioned among and against other publications with regard to novelties in our approach and differences to previous work. The main part is Chapter 4, which defines the specific goals and presents the general concept for context prediction and the architectural approach taken in this research project in a top-down manner. After giving an overview of the context prediction architecture, its five steps are explained in more detail, including surveys on classification and prediction methods. Chapter 5 then shows design principles of the developed software framework, while details of the implementation are given in an accompanying diploma thesis [Rad04]. In Chapter 6, an evaluation of the architecture and a selection of methods for context recognition and prediction is presented. Finally, in Chapter 7 the thesis is summarized by pointing out the main arguments and the scientific contribution and giving an outlook on possible future research.

Chapter 2

Context Computing

This chapter introduces the main foundation of the thesis: *context computing*. The term *context awareness* denotes the property of a system to adapt to context; we define context computing as *the combination of hardware, software and models to implement context awareness in computer systems*. A short overview of context in general is given, followed by a motivation on why context awareness can be used to enhance current applications. Then, different definitions of context with regard to pervasive computing are discussed and different aspects of context are examined. Finally, a definition of proactivity within context computing is given to facilitate a further discourse on context prediction in Chapter 4.

2.1 Context

Context computing, as a research topic, is concerned with the realization, i.e. the conception, development and implementation of computer systems that employ context awareness. It is one of the building blocks of pervasive computing applications [Fer03c] and often presented as the key enabler for future applications. Advantages of making applications in the fields of pervasive, ubiquitous or mobile computing context-aware are obvious: user interaction and application behavior can be adapted to the current situation, making devices and their applications easier to use and making more efficient use of the device resources. Many groups have presented their research on context computing and thereby laid solid ground for further work.
Pervasive computing is generally concerned with the interaction between computer systems and the real world, with the embedding of computing resources in real-world objects. Therefore, interfaces between the physical and the virtual world mediate the mutual influence that both worlds have on each other. Interfaces that transport information from the physical to the virtual world are called *sensors*, interfaces from the virtual to the physical world are called *actuators*, as depicted in Fig. 2.1. Pervasive computing applications that have an impact on the real world have to feature interfaces in both directions: sensors for gaining knowledge about the environment and actuators to control certain physical properties of objects. Context computing, situated at the input side of applications, is inherently concerned with sensors as sources of information. But the use of context as a concept is not even limited to any sensing technology. The general approach can be used for arbitrary computer systems, possibly even without explicit user interaction. Although context computing stems from sensor and user interface centric research, it could in principle also be used in embedded network appliances which only handle network traffic, embedded control systems, etc. The involved interfaces in such an environment would be virtual

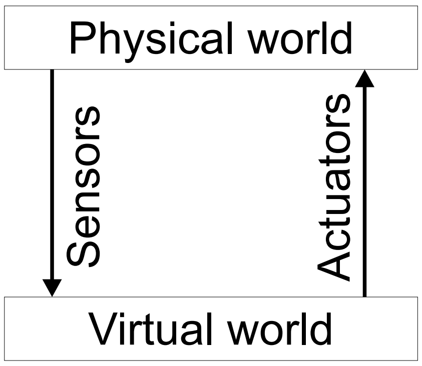


Figure 2.1: Sensors and actuators as interfaces between the physical and the virtual world.

rather than physical, providing contact to other parts of a larger system instead of directly to the real world via sensors and actuators. However, most of the current research on context computing, including this thesis, is directly concerned with sensors as interfaces to aspects of the real world.

2.1.1 Motivation for using Context Awareness

Context awareness stands for the ability of a system to adapt to its environment, which offers a number of advantages and possibilities for new applications. One of the more intuitive advantages is ease of use; if devices can adapt to their — or for personal, mobile information appliances their user's — situation, they can engage in more efficient user interaction. As its definition of "being aware of a situation" suggests, context awareness means that devices can prevent interruptions at inappropriate times, like traffic announcements from the car stereo bursting into a quiet passage of classical music or a conversation between passengers. Other advantages may be less obvious, but context awareness can also lead to reduced energy consumption because devices only need to activate those components that are necessary for operating in the current context. For mobile devices, this is even more important than for infrastructure components, because battery life, which depends on the average power consumption, is one of the most important factors. Examples by means of prototypical implementations or empirical studies have been given for the advantage of more intuitive user interfaces (e.g. [BGS01, DSAF99]), for reducing the power consumption of large systems (e.g. [Moz98]) or for improving battery life (e.g. [Lou01]).
In [Dou04], a number of reasons were presented why context awareness could enrich current computer systems by providing better user interaction and it has been argued why many systems still fail to do so; but the following quote pinpoints deficiencies of current systems:

> *[...] by incorporating notions of context into interactive technologies, those technologies can be made more sensitive to the details of the specific settings of use, which social scientists have often pointed out in critiques of conventional system design. Social scientists have argued that traditional interactive system design often rigidly fails to respond to the setting in which action unfolds [...]*

We think that context awareness, and more specifically context prediction, can offer decided advantages for future applications, allowing them to address those deficiencies.

2.1.2 Definitions

Although — or maybe rather because — context computing as a research field is still young, there are currently many different definitions of context. First of all, the term context is found quite regularly in common language use and is defined by Merriam-Webster's online dictionary in a generic way [Web]:

> *1 : the parts of a discourse that surround a word or passage and can throw light on its meaning*
>
> *2 : the interrelated conditions in which something exists or occurs*

Within computer science, the FOLDOC (Free On-line Dictionary of Computing) describes context as [Fol93]:

> *That which surrounds, and gives meaning to, something else. In a grammar it refers to the symbols before and after the symbol under consideration. If the syntax of a symbol is independent of its context, the grammar is said to be context-free.*

The usage for describing grammars seems to be the oldest use of context in computer science, which describes why the FOLDOC definition is centered around this use. But it can still be seen that the general definition given by the first sentence is valid for current uses of the term in context awareness, which will be explained in the following. The general term context is also used in other fields of computer science: for compilers, there is a distinction between *context-free* and *context-sensitive* grammars [Cho59], whereby a context-free grammar can be accepted by a push-down automata and a context-sensitive one can not (instead, a linear-bounded Turing machine is necessary). To prove that a formal language is not context-free, the so-called pumping lemma is used, but it is difficult to prove the opposite. In operating systems and microprocessor design, a *context switch* denotes switching from one thread of execution to another one [SG97] and typically involves saving and restoring processor registers and memory regions. Applications can be interrupted and return to their state of execution at a later time, enabling the preemptive multi-tasking concepts of current operating systems to achieve a quasi-parallel execution of multiple application. Next, *context sensitive menus* are a well-known concept in HCI, distinguishing the flexible "anywhere" menus from pull-down menus associated with a fixed menu bar. The advantage of context sensitive menus is that they are variable in place and content, depending on the current state of the application and area of invocation. First uses of context sensitive menus can be tracked down to Smalltalk, where this concept has been termed "pop-up menus" (cf. [Gol84]). A highly technical term is the *bundle context* within the OSGi standard [OSG03], which is provided to every component that is deployed in an OSGi compatible container. Using this context, components can access their environment, which includes services provided by the container itself and other components installed in the container. There are undoubtedly many other uses of the term context in computer science, but those are mostly similar to one of the meanings above.
The newer Wikipedia [Wik] already has a definition of context referring to the interaction between devices and users, or more generally, the situations a device is used in:

> *Context is a term with numerous meanings and shades of meaning.*
>
> *1. In archaeology, the context (physical location) of a discovery can be of major significance. See Stratification.*
>
> *2. In linguistics, context refers to the meaning of a word or phrase with regard to its place in the sentence and the topic being discussed.*

> *3. In computer science, context refers to the circumstances under which a device is being used, e.g. the current occupation of the user. (see also context awareness)*

Before going into more detail of the term context in this last usage, the second term, namely “awareness” should also be examined more closely. Merriam-Webster’s defines it as

> *1 archaic : WATCHFUL, WARY*
>
> *2 : having or showing realization, perception, or knowledge*

which already gives a good hint on how it is used in pervasive computing: that devices have and show (by means of their behavior) perception of their environment. Wikipedia also offers an interesting definition:

> *In biological psychology, awareness describes an animal’s perception and cognitive reaction to a condition or event. Awareness does not necessarily imply understanding.*
>
> *Awareness is a relative concept. An animal may be partially aware, may be subconsciously aware or may be acutely aware of an event. Awareness may be focused on an internal state, such as a visceral feeling, or on external events by way of sensory perception. [...]*

Pointing out the difference between awareness and understanding, it approves that research on context computing must not necessarily create a full world model to be useful; understanding the environment in terms of a model is no prerequisite to being aware of aspects of it. This also allows to better explain the definition of awareness given by Paul Dourish and Victoria Bellotti in 1992 [DB92]. According to their work, awareness is *an understanding of the activities of others, which provides a context for your own activity*. Knowing activities of other people or systems involved in some task is important to put ones own behavior into context. However, as has happened with many terms in computer science, awareness was used for describing different things. Having become overused most notably in CSCW, it is important to ask “awareness of what?”, as has been pointed out in [Sch02b]. This gives rise to the compound term “context awareness” now being used within pervasive computing to describe that a system is aware of its context of use.

Since there is, at this time, no common, specific and detailed definition of context within this area, it is reasonable to refer to those being currently used. The following set of definitions is neither exhaustive nor in order of importance, but should reflect the most prominent definitions. It is clear that the notion of context developed alongside the research projects and that it is still evolving as new aspects are investigated.

In their classical paper, Bill Schilit, Norman Adams and Roy Want at Xerox PARC laid ground for many following publications on context computing [SAW94]. Their early definition of context is one of the more general, clearly defining context to include more than merely geographical position information:

> *Three important aspects of context are: where you are, who you are with, and what resources are nearby [...]. Context encompasses more than just the user’s location, because other things of interest are also mobile and changing. Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social situation, e.g., whether you are with your manager or with a co-worker.*

Although this definition clearly states that context has many aspects, the main criticism is the restriction of “important” aspects to computing, user and physical context. But for our current research, the exemplary list of other aspects gives hints on which sensors could be added to our case studies.

In [BBC97], Peter Brown, John Bovey and Xian Chen at the University of Kent suggest a classification of context aware applications into *continuous* and *discrete*. Within this classification, we are mostly concerned with discrete context as the sensors are sampled at discrete time steps. The transition to continuous context depends on the application: if context-dependent information is displayed almost in real time, we can interpret it as continuous context even if sensors are still sampled at discrete, albeit very small, time steps. In their description of a context aware application, they present a model of context defined as a DTD (Document Type Definition). This DTD defines tags for context dimensions and is extensible for different applications. Concepts of hierarchical context dimensions are mapped to a naming scheme for context attributes. The idea of using an open DTD is intriguing because such an internal representation of context could be easily externalized and made accessible to applications or other devices.

James Crowley, Joëlle Coutaz, Gaeten Rey and Patrick Reignier, while defining an ontology for modeling context [CCRR02], distinguish between user context and system context. The user context is defined by the goals which are pursued by the user while interacting with a system, whereas a user may have many different goals in parallel and can switch between those goals. On the other hand, the system context is composed of a model of the user context and a model of its own internal context, where the model of the user context determines the interpretation of observations and the model of its own context allows to hierarchically structure context models. The system context is at the top of the hierarchy, and the user context at the bottom, itself being composed of observations. An interesting discourse on the use of context in natural language processing and AI is given in an introductory section, where the authors describe how the “frame problem” in AI developed into the “grounding problem”. This change in the notion of context motivated researchers to move from a linguistic view of context, i.e. the frame of a symbol, towards a view based on action, which is also more closely related to how we use context in our work.

In his PhD thesis [Sch02a], Albrecht Schmidt presented a “working model for context-aware mobile computing”, which is basically an open, extensible tree structure. The proposed hierarchy of features starts with distinguishing human factors and the physical environment and ramifies from there. Orthogonally to this hierarchy is the time, providing a history of previous feature values. Schmidt argues that it is a general challenge to identify the set of relevant features for a situation. This remark completely applies to our work and is one of the most critical issues for automatic, unsupervised recognition of context (cf. Section 4.6.2).

In [SBG99], Michael Beigl, Hans-Werner Gellersen and Albrecht Schmidt distinguish between *explicit* and *implicit* context, defining explicit context as one that is queried directly from the user and implicit context as the automatically recognized one, derived from sensors and application behavior. In this thesis, the notion of context is actually both: the primary motivation is to derive context from user behavior (implicit context), but the user has the possibility to assign descriptive labels to automatically constructed, abstract context classes (explicit context).

In a very recent work [HLA$^+$04], Lonnie Harvel et.al. describe the context cube, an application of data warehousing methods to context computing. Basically, all context dimensions are used as attributes in a relation (i.e. database table) and are thus recorded as the combination of values. By defining all measured values as key attributes in the table, they can associate additional dimensions for computed values, named “derived context”. On such a context relation, well-known cube operations from data warehousing can be supported. In this model of context, it seems difficult to directly support non-atomic context dimensions (e.g. a list of devices in range), but the cube operations should allow an easier extraction of high-level information from the low-level sensor data. It needs to be evaluated if a context cube can be constructed with an upper memory bound, suitable for usage in embedded systems; the low-level sensor data could be stored in an aggregated cube along with the recognized context classes as derived context (cf. Section 4.2), facilitating data mining techniques for more effective context prediction.

The term *context atom* has been used by Jani Mäntyjärvi, Johan Himberg and Pertti Huuskonen in [MHH03] (after being introduced by Johan Himberg et.al. in [HKM+01]) to describe basic context dimensions which are derived from low-level sensor data by pre-processing. After the pre-processing, which consists of feature extraction and fuzzification, a context atom is simply represented as a continuous, normalized value. This definition is very similar to the internal use of features in this thesis. An *atom vector* closely resembles a feature vector in our architecture (cf. Section 4.2).

One of the most prominent definitions of context has been given by Anind K. Dey et.al. in [DA00]. Context has been defined as

> *any information that can be used to characterize the situation of an entity*, where an *entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.*

In this thesis, we adopt this general definition of context, which seems to become accepted by most of the community (e.g. [CCRR02, FBN01, GPZ04b, GWPZ04]). We understand context as describing *the situation a device or user is in, identified by an abstract label and defined by the combination of sensor values*.

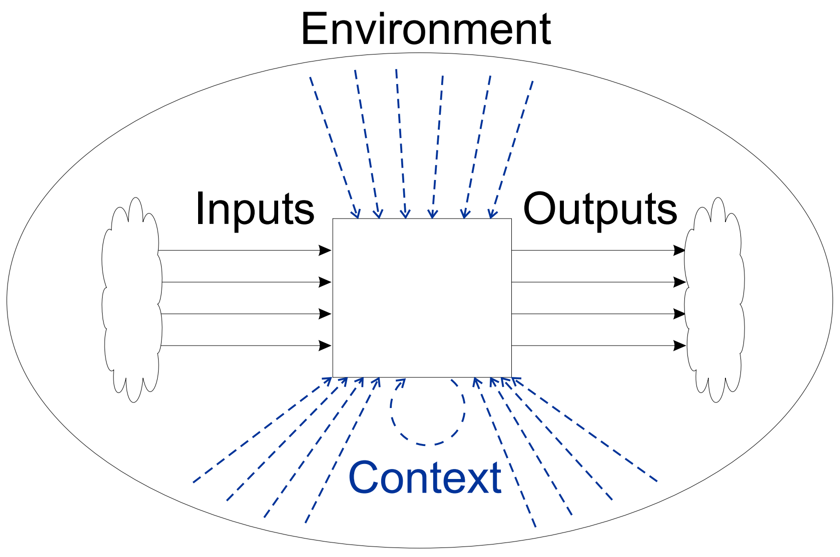


Figure 2.2: Context of a black-box system

As depicted in Fig. 2.2, a system, seen as a black box, has some inputs and outputs which define the normal functionality of the system. Context, however, describes the influence of the whole environment on the system, including the sources of inputs and destinations of outputs, or, as described in [LS00], *Context is everything* but *the explicit input and output*. Context is what surrounds a device and thus, all of its possible aspects can not be enumerated for the general case. It even includes the internal state of the system, leading to a form of self-awareness. In a more detailed view of computer systems in pervasive computing, we can concentrate on the internal influence of context onto the system. As discussed earlier and shown in Fig. 2.1, a computer system observes its environment via sensors and controls it via actuators. Context computing is situated at the input of the system, providing high-level information about the environment to applications, which in turn control certain objects via actuators, as depicted in Fig. 2.3. Thus, we are concerned with extracting high-level information that is meaningful to applications from low-level sensor data. In the following section, we shortly discuss the different aspects of context that describe a situation.

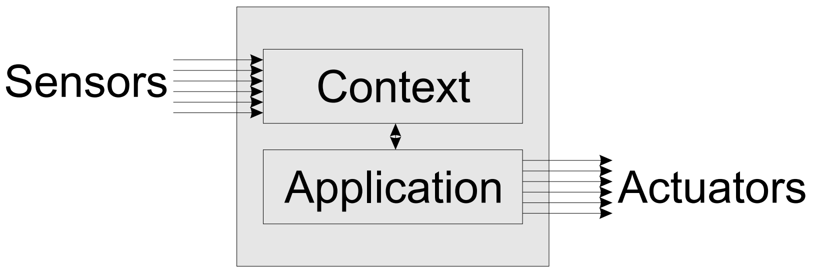


Figure 2.3: Using context within a system

2.1.3 Aspects

In general, context of a user or device, which are similar in the case of body-worn or close-to-the-user information appliances like mobile phones, can not be confined to only location (cf. [SBG99]). There are many aspects which describe the whole context — some of them are important to a specific application or for uniquely identifying a situation, others are not. However, for an architecture for context recognition, we can not restrict ourselves to any aspect of context but must support arbitrary ones. In general, we can identify at least the following aspects of context (an earlier list of aspects has first been presented in [MRF03b]):

- geographical (e.g. country, street, building, floor, office)
- physical (e.g. lighting, noise level, temperature, acceleration, tilt)
- organizational (e.g. institution, department, group, project)
- social (e.g. family, friend, co-worker, married, single)
- emotional (e.g. heart rate, skin conductance)
- user (e.g. profile, location, capabilities, role, access rights)
- task (e.g. documenting, programming, building a house)
- action (e.g. typing, reading, walking, sitting, talking)

- technological (e.g. connectivity, network bandwidth, network latency)
- time (e.g. time of day, weekday, week, month, season of year)

Different applications demand the awareness on different aspects of context, e.g. a multi-modal communication application will depend on the network context (technological), while a tourist guide will be more responsive to geographical context. As described in more detail in Section 4.4, a single sensor does not seem to be appropriate to capture all the different aspects of context. Instead, a multitude of sensors will be necessary to gather even a small subset of the rich aspects of a user or device context.

2.1.4 Context in Time

The term context awareness itself does not state whether a device is aware of its past, present or future context; we specifically consider the predicted future context. Past and current context have been considered by previous work, e.g. explicitly in the model of context presented in [Sch02a]. In our work, we do not store a complete history of past contexts because of memory limitations of embedded systems. Instead, past context is used to derive user behavior and to extrapolate current context into future context by applying this derived knowledge. This general methodology is independent of the prediction algorithm, sliding window approaches can be used as well as online update of prediction models.

2.2 Proactive Context Awareness

The term proactivity has been used in computer science most prominently for software agents, where one important difference between agent oriented programming and object oriented programming is the proactivity of software agents. There are two notions of software agents, the "weak" notion and the "strong" notion. The former is a general way of defining agents and includes autonomy, social ability, reactivity, proactivity, temporal continuity and goal orientedness. The latter specifically describes computer systems and demands the additional properties of mobility, benevolence, rationality, adaptivity and collaboration. For more details on the definition of agents, we refer to [WJ94]. Even in the weak notion of the agent concept, proactivity is one of the basic features, but is related to reaching the goals of a software agent — an agent is considered to be proactive if it takes initiative to reach its defined goals. In pervasive computing, goals are defined by the user of the system and applications should aid the user in pursuing them. A general definition could be "that proactivity is the active processing on how to reach the desired situation" [Toi00], which, stemming from agent oriented programming, is also applicable to context-aware systems. For information appliances in the field of pervasive computing, the ultimate goal of the system is to support the user in saving time. It should be noted that, in this thesis, the term proactivity explicitly describes the use of prediction techniques to refer to future contexts. In other publications, the term proactivity has been used with regard to context with a different meaning, namely for describing the automatic triggering of actions when some context becomes active, e.g in [BBL$^+$00] for types 1 and 4 of their six types of context-aware applications.

Proactivity and reactivity are two sides when classifying systems, but on the other hand, they are not mutually exclusive. For different aspects of a system (e.g. for different applications building upon a context-aware middleware), either reactivity or proactivity might be more important. E.g. for user interfaces, reacting to changes in user or device context will be more appropriate than adapting the interface in advance, while the user is still using it in the old context. Other applications can definitely benefit from proactivity, when reacting to changes in the environment is not enough to provide reasonable services to the user. In the following section, a few important application areas are discussed where proactivity offers decided advantages.

2.2.1 Motivation for using Proactivity

At the current state-of-the-art, we can investigate a few application areas within the field of pervasive computing where future context awareness offers added benefit. These application areas are, on an abstract level:

- Reconfiguration: System reconfiguration in general, not being restricted to context-based reconfiguration, is today one of the most time-consuming tasks associated with computer systems. We can further distinguish between *light-weight* and *heavy-weight reconfiguration*, where light-weight reconfiguration includes modification of the system configuration or general online, near real time adaptation to changed environments. In the following, we will focus on heavy-weight reconfiguration, which includes tasks that impose a noticeable delay during reconfiguration, leaving the system in question out of service until reconfiguration has finished. Boot-up of systems, installation or update of applications, maintenance and infrastructural changes, downloads, searches in large databases, etc. all consume, or even waste, significant amounts of valuable work-time. Any progress towards shortening these reconfiguration times yields a direct improvement for the involved people.
  An example for a current application in this area that exploits context prediction in a very application-specific way is implemented in current upper-class BMW cars. Whenever its key comes into spatial proximity of the car (even before being used to unlock the car via the integrated remote radio control), the on-board navigation system starts its boot-up and initialization phase, which can take up to roughly 30 seconds. This way, the navigation and infotainment system is fully available when passengers enter the car. Another desirable scenario along the same lines would be the desktop computer being booted up and having all required applications loaded when the office is entered (but without running during the night and thus wasting resources).

- Accident prevention: An accident can be seen in the general case as an undesirable system state, and preventing such undesirable states has applications in many different areas. E.g. in telecommunication, an undesirable state is an overload in some network equipment or communication link. Load prediction is already used by larger telecommunication cooperations to prevent system failures by proactively updating or bypassing highly loaded systems in time. In medicine, there is a vast multitude of undesirable or dangerous states and situations than can be monitored with bio sensors and should be predicted to prevent permanent damage. However, it is important to point out that within our approach, such an undesirable situation must have already occurred in the past to be predicted. Lacking an application-specific model of desirable and undesirable situations (which is omitted on purpose within the scope of this thesis), it is only possible to learn from past situations by means of a context history. Yet unknown contexts can not be predicted in a general, application-independent way. Thus, we strongly advise against using this approach for prediction and prevention of undesirable states in safety-critical systems.

- Alerting: This is best known from the domain of PIM (Personal Information Manager) type applications, including calendar, project management, scheduling, appointment and group coordination systems, but includes arbitrary applications that need to alert users in some form. These systems already provide a multitude of alerting capabilities, ranging from message boxes bound to being displayed on desktop computers or PDAs, signal lights, audible notification to sending emails, SMS or pager messages to user's mobile devices. However, events leading to such alerts are either triggered by certain actions (e.g. a colleague entering a virtual meeting room) or have been scheduled in advance, being entered in a calender. When being able to predict future context, a device can autonomously issue alerts before some relevant contexts occur, without the need for manual scheduling.

- Planning aid: Simply displaying predicted future context in a structured way and allowing to interactively browse it can provide a powerful aid for human-driven planning and scheduling. This puts people in the control loop, allowing to manually modify system behavior, but being assisted by predictions of future situations. Due to their informational nature, applications from this area will demand an estimation on the probability of the predictions being true, i.e. on their certainty (cf. section 4.8.1), which might not be strictly necessary for other application areas. This limits the choice of prediction algorithms to ones that are based on probabilistic concepts and disqualifies simpler variants.

From these general application classes, we take one more specific scenario for a variety of reasons. First of all, it constitutes a "living laboratory" approach, with researchers being the subjects in the evaluation while the scenario can be refined iteratively by them. Additionally, we only use sensors and other hardware which are available off-the-shelf and resign from developing custom devices as to demonstrate the feasibility of an approach building on currently available technology. And last but not least, it is a scenario taken from everyday life and should thus assist in communicating the concept of context prediction to a wider audience.
Our scenario is based around a fictitious employee, Alice, working primarily with information technology and is taken from the reconfiguration application area. Her desktop and work space is dominated by a desktop computer used for most of the work. Additionally, she owns a smart phone, i.e. a mobile phone capable of running custom software and with communication technology like GPRS, Bluetooth or WLAN. Using various sensors such as lightning, microphone, its GSM interface, the built-in camera and Bluetooth, the smart phone can automatically recognize current and predict future context while its user advances from one situation to another. When Alice is waking up in the morning, the device detects this due to picking up noise and can, now being in the context "getting up" already (reactively) fetch the newest headlines using her home WLAN while Alice spends time in the bathroom. When switching to the context "getting up", the device has already predicted that in roughly 15 minutes Alice will use the device to read today's headlines while having her breakfast (because she has done so regularly for the last 2 months). Thus, 5 minutes before this context "having breakfast" will be entered, the smart phone sends an event to the coffee maker to start brewing fresh coffee. After Alice has finished reading news items, she puts the smart phone into her pocket and leaves for work, which the phone recognizes as context "going to work". Although she does not start working at regular times, sometimes before 7:00, at other days around 11:00, the smart phone can now already contact the company gateway over GPRS, initiating a boot-up of her desktop machine 10 minutes before she is expected to arrive (by predicting the time she will be spending in the context "going to work" from the mean time it took during the last months). Context prediction in this sense does not only save time for Alice. Co-workers who want to know when she is likely to be in the office need no longer call her on the mobile phone, potentially

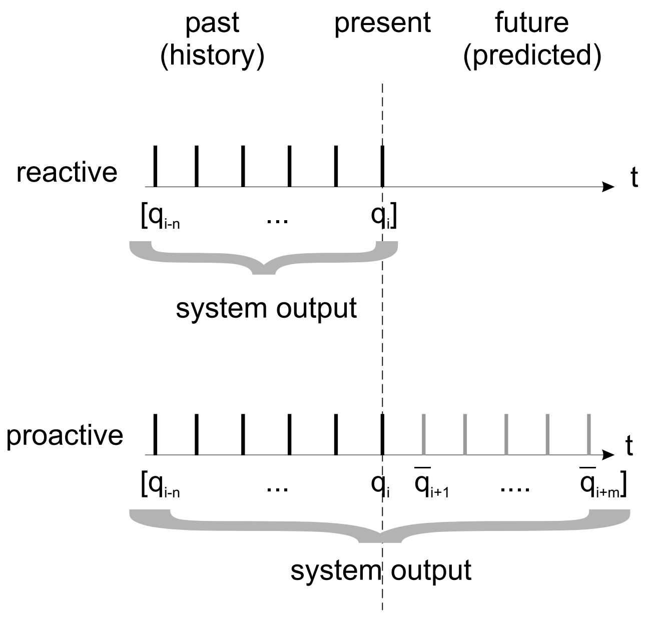


Figure 2.4: Proactivity vs. reactivity

disrupting her way to work, but can simply query to prediction of when she will likely be in the context "at work".
This simple scenario shows how automatic context prediction could be used to save valuable time by making devices proactive with regard to context. Additional operations that consume significant amounts of time like synchronizing the home computer with the company network could also be performed automatically before the user will arrive at the new location. In general, we try to provide a middleware for context prediction and leave it to the creativity of application designers which actions should be initiated based on predicted, high-level context information.

2.2.2 Definition

The following definition of proactivity has also appeared in [MRF03b, MRF04b].
Formally, the difference between reactivity and proactivity can be seen in the dependence of the current system output on the system state trajectory (cf. Fig. 2.4). If interpreted as a (Moore) state machine, the internal "state" of a system at time t can be described as $q_t = \delta\langle q_{t-1}, a_{t-1}\rangle$, where q_t is the current state, q_{t-1} is the last state and a_{t-1} is the input value at time $t-1$ (cf. [Pic75]). In this definition, system inputs and state transitions are assumed at discrete time steps $t \in \mathbb{N}_0$. The system output depends on the state, and this is how the difference between reactivity and proactivity is defined in the context of this thesis. In a reactive system, the output b_t at time t only depends on the current and — implicitly — on past states:

$$b_t = \lambda(q_t)$$

In a proactive system, it can also depend on predicted future states:

$$b_t = \lambda\left\langle q_t, \overline{q}_{t+1}, \overline{q}_{t+2}, \ldots, \overline{q}_{t+m}\right\rangle$$

The future states $\overline{q}_{t+1},\ldots,\overline{q}_{t+m}$ for m discrete time steps are predicted recursively by some arbitrarily complex process $\overline{q}_{t+i} = p\left\langle q_t, \overline{q}_{t+1},\ldots,\overline{q}_{t+i-1}\right\rangle$; if only q_t is needed for predicting any $\overline{q}_{t+i}$, then p can simply ignore the predicted states $\overline{q}_{t+1},\ldots,\overline{q}_{t+i-1}$.

Chapter 3

Related Work

This chapter presents recent research projects and groups which are related to context computing in general or more specifically to context prediction. For a good survey considering related work until 2000, we refer to [CK00]. Nonetheless, the most important pioneering projects which helped to define the notion of context awareness and/or had great influence on the development of this field are briefly listed in Section 3.2. Other related work regarding specific steps in our architecture or used methods and algorithms is mentioned in Chapter 4 where appropriate, including publications concerning the classification and prediction methods that were used as the base for our literature survey. Some of the presented material has been published in [MRF03b, MRF03a, May04, MRF04b].

Before going into more detail about the specific projects and their relation to this thesis, a few general remarks on parts of the developed architecture shall be mentioned. Our general architecture for context recognition is very similar to the one presented in 1999 in [Lae99], but with improvements towards continuously running devices and the addition of context prediction. The term context prediction itself has been used by different research groups, but, as explained in more detail for the related projects, most predictions are performed for lower-level location information. We aim to predict high-level context identifiers, i.e. classes of situations a user or device usually is in. E.g. Jan Petzold et.al. refer to context prediction [PBTU03a, PBTU03b, PBTU03c, PBTU04] when in essence it is location prediction, but on a qualitative instead of quantitative level (in the same way we are using context identifiers to predict the next abstract context).
The notion of feature extraction, one of the cornerstones of current work on application- and sensor-independent context awareness, has been applied in a similar way in [CP98, CSP98, Hea03, MHH03]. However, feature extraction and sensor fusion have also been studied extensively in the field of robotics, but with focus on geometrical properties of the environment [Bru99]. For context computing, the different sensors deliberately capture different aspects of the environment (cf. Section 4.4). According to [Bru99], this contradicts the definition of sensor fusion. Thus, in the opinion of the author, sensor fusion can currently not be applied successfully within context computing with state-of-the-art context "world models". To achieve redundancy and quality improvements by applying multiple sensors which capture the same aspect of context, a common model of quantitative context data will be necessary. Applying fusion techniques at a higher level of abstract context identifiers seems more realistic than at the level of sensor data processing. In robotics, similar considerations have been articulated [BFB94]:

Sensor data-level fusion, that is fusing data at the sensor level is in general only feasible between identical sensory devices, all having the same perspective. Feature (or primitive) *level sensor fusion however, provides a reasonable level of abstraction.*

Although feature extraction and classification are well-known in different fields of research, covered by standard literature (e.g. [Bov00, Nel01]), most publications only cover numerical, continuous features. For context computing in mobile devices, systems with custom sensors (e.g. accelerometers, light sensors) often use minimum, maximum, mean and variance as features (e.g. [Lae01]). Because the restriction to numerical features severely limits the choice of sensors, we introduce a model for utilizing heterogeneous features (e.g. list of MAC addresses and WLAN ESSID) in a common classification step (for details, cf. Section 4.5). For a Kohonen SOM, a method has been described to cope with heterogeneous input values [NB01]. However, the general case of context recognition with heterogeneous features does not seem to have been covered before and is one novelty in this thesis.
Time series forecast has been explored in different areas, including distributed simulation [Fer99], software agents [RC98], data value prediction in processors [SS97], data mining from health records [WLW01] and theoretically for neural networks [TF93]. Therefore, we are not developing new methods for time series analysis or prediction, but applying appropriate, known methods.

3.1 Projects and Groups

Context computing is still being defined by different research groups and in different projects. Thus, it is reasonable to review what has been done recently, but it would not be sensible to split research groups from current projects while some of them still define their focus and some research groups are not clearly distinguishable from project groups. The following list of projects is non-exhaustive and by no means complete, as there have been too many publications to list which were dealing with context computing in some way during the last few years. It should present those publications that are more closely related to the main parts of this thesis: general frameworks and middleware for context computing, work on context prediction and work on dealing with context on embedded systems with limited resources.

TEA

The Technology for Enabling Awareness (TEA) project was a joint project between Starlab NV/SA in Belgium, TecO at the University of Karlsruhe in Germany, Omega Generation at the University of Bologna in Italy and Nokia Research in Finland (`http://www.teco.edu/tea/`). Its aim was research on context awareness with handheld and wearable devices with a focus on low-cost sensors and mobile phones [Lae99]. The project has developed a hardware board for collecting sensor data and connected it to a standard mobile phone. The term “adaptive context awareness” has been used to describe the use of machine learning algorithms on low-level sensor data; those algorithms should adapt to frequently seen user contexts. One of the project goals also was unobtrusiveness, i.e. to demand as little human supervision/attention as possible. An architecture has been described that builds on a combination of SOM and k-means clustering, at the disadvantage that the number of context classes detected by the SOM was static. This static and pre-defined number of classes as well as the proneness of the SOM to forgetting previously learned clusters necessitated a second, higher-level clustering step. However, even with a variable internal topology due to a variable number of classes, the presented approach might not be suitable for running continuously in the background. As described in [Lae01], it is necessary to decrease the learning rate of the SOM to get stable results. In a continuously running online application, it is not possible to steadily decrease or increase parameters; only dynamic adaptions of parameters are possible. The principal architecture in this thesis is similar to the one used in TEA, but aimed towards life-long operation and extended by the context prediction step. In [Lae99], a simple Markov chain model is used on top of the higher-level k-means clustering step to increase the context recognition accuracy. This simple form of prediction via a 1-step Markov model helps to resolve ambiguities between context classes when switching from one context to another (by taking the most probable successor of the last context into account when computing the current context class). In our work, the prediction step is not only used as another indicator for detecting the current context, but is used to actually forecast future context. Additionally, we intend to improve the architecture by a clear definition of abstract interfaces between its steps to simplify the exchanging of methods.

Smart-Its

The project Smart-Its [GSB02] is building on the experiences gathered with the MediaCup [BGS01] and the TEA projects. It changed the focus of their research from context awareness in single devices to ad-hoc networking of context-aware devices. Smart-Its, as an experimental hardware platform, integrate sensing, processing and communication capabilities into a single, small device. Their design is modular, separating the “sensor unit” from the “core unit”, which provides processing and communication. With Smart-Its, ad-hoc networking, sharing of context information and sensor fusion across devices are investigated. To support the rapid prototyping of applications, additionally tackled issues are to minimize the physical size and power consumption and to use Bluetooth for communication to ensure interoperability with consumer devices. A worthwhile topic for future research is to evaluate if the concepts and methods developed in this thesis could be ported to run on the Smart-Its platform, which constitutes low-end embedded systems that are still capable of processing sensor input.

GUIDE

The GUIDE system is a cell-based location system, broadcasting dynamic information to mobile units with a web browser [CDM$^+$00]. Using the IEEE 802.11 wireless LAN standard, the infrastructure is open and flexible and can be used by a wide range of clients. Location information is transmitted by the base stations, which restricts the accuracy of location information to the coverage area of WLAN access points (up to 300m). With current state-of-the-art WLAN base stations, a finer resolution could be achieved by analyzing the respective signal strength from multiple base stations. The displayed information depends on personal (i.e. interests, location) and environmental (i.e. current time, opening hours of attractions) context and acts as a location-sensitive tour guide for visitors in Lancaster (UK). One interesting experience from the field trial in this project was that context-aware systems should not be constrained too much with regard to current context. The first versions of the user interface running on the mobile units were only allowing to access information that was associated with or relevant to the current context. However, users were dissatisfied that they could not access other information and services whenever they desired to. Thus, one of the experiences of GUIDE was that context-aware systems should allow to access certain important information or services at all times.

IST CONTEXT

The main objective of the currently ongoing CONTEXT project (`http://context.upc.es`) is to enable development of context-based services in active networks. The publications that are publicly accessible at the time of this writing indicate that the project is network- and infrastructure-centric [XRCA04] and utilizes mobile agents [YGT03]. Therefore, it is not comparable to our approach of making devices context-aware without requiring any infrastructure, but has fairly different goals. Despite the different goals, the necessity of a middleware to decouple applications from the acquisition of context has also been identified in this project, which is called “Context Management System (CMS)” in [XCS$^+$04] and is based on database management systems. The project is currently working on an execution environment based on active networks which should provide a context awareness middleware to applications.

Equator

The Equator Interdisciplinary Research Challenge (`http://www.equator.ac.uk/`) is a six-year project to promote the integration of the physical with the digital world. By developing “experience projects”, they tackle three research challenges which have been defined for this project: “interaction”, “devices” and “infrastructure”. Within this scope, devices like the Bristol Cyberjacket [RM02] have been developed or applications like weight surfaces [SLS$^+$02] have been built. A publicly available framework for C++ and Java, called Equip, supports a shared data service for inter-application communication and event handling and has already been used in a few applications within the project. Context is used within Equator partly for user interaction with real-world objects (like the weight surfaces), but most work has focused around context in embedded systems [BR02, CCLG02, LSG02].

Georgia Institute of Technology (US)

The “Aware Home” is built as a living laboratory for empirical research on ubiquitous computing, with the goal to sense contextual information about itself and its inhabitants [KOA$^+$99]. One of the motivations for this project is to enable support for elderly to be built into their homes. Kidd et.al. reported about the status of this project, which should also learn user’s habits, but it was not finished at the time of this report. The Context Toolkit [SDA99, DAS99] has been developed to support work on the Aware Home project. Consequently, it is a very flexible framework for abstracting context sensing from applications in a distributed, heterogeneous network environment. Aimed towards context sensing embedded into the infrastructure, it uses HTTP and XML for communication between sensors (represented by their “context widgets”) and higher-level components (named “aggregators” or “servers” and “interpreters”). It is not directly addressing context prediction or targeted at embedded systems; we aim to implement context recognition and prediction locally at each device, without the need for infrastructure components, while the Context Toolkit intentionally is an infrastructure approach for context sensing. The Conference Assistant [DSAF99] was one of the prominent applications built upon the context toolkit. There are some limitations of the context toolkit, described in [DAS99]: it does not support continuous context and does not deal with unreliable/unavailable sensor data.

Exeter (UK)

At Exeter, context awareness was applied to the field of information retrieval to improve the relevance of retrieved information with regard to the user’s situation. In [BJ02] they suggest that the current context is often changing gradually and thus is “semi-predictably”. However, their notion of proactivity describes only the retrieval of documents when conditions based on the current user context are met. Thus, it should rather be described as being *reactive with regard to context*. Nonetheless, their developed “Context Diary” keeps track of previous contexts, which is used for predicting future aspects of the user context. They were restricting their selves to gradually changing and thus easily predictable aspects of context (they give the examples of location and temperature). For context-aware retrieval, advantages of predicting aspects of user context were identified in this work, but they do not seem to be generally applicable to context prediction. A prototype system for context-aware information retrieval has been built, but at the time of this writing it does not represent a general framework for developing context-aware applications.

MIT Media Lab (US)

In [CMP00], Brian Clarkson et.al. describe a wearable system with a video camera and a microphone, capable of distinguishing coarse locations (although it is stated that the approach is not restricted to location). Similar to their previous work [CSP98], they used Hidden Markov Models (HMMs), which are a well-known technique for recognizing time series, as a basic model for recognizing context. After feature extraction (cf. Section 4.5 for a detailed explanation), they used unsupervised clustering (cf. Section 4.6) to distinguish between user contexts; this approach is very similar to the one taken in our architecture, but has been implemented only for audio and video. Because of their method selection, there was also the distinction between a training and a test phase, which we explicitly seek to avoid as it lacks adaptability.

MavHome

The MavHome project [CYEOH$^+$03] by Diana J. Cook et.al. aims to create an agent-based intelligent home with a vision in the spirit of the Aware Home. Their multi-layer architecture consists of a “physical” (i.e. sensors), a “communication” (i.e. network), an “information” (i.e. database) and a “decision” layer and is based on CORBA for remote method invocation. This architecture is similar to the one used in this thesis, as it is also a bottom-up structure for recognizing context. The distinguishing factor of MavHome is its use of different prediction methods to automate actions within the house. Although the Neural Network Home also applies machine learning to predict inhabitant actions, there is a more detailed examination of prediction in the MavHome project. They have developed different algorithms for categorical time series prediction (cf. Section 4.8.3): SHIP [CYEOH$^+$03], a simple sequence-match algorithm, Active LeZi [GC03], a prediction by partial match algorithm based on Markov models, TMM [CYEOH$^+$03], a Markov model based on higher-level action sequences and Episode Discovery [HC03, DCB$^+$02], which uses data-mining techniques and the minimum description length principle to predict periodic patterns. Those different prediction algorithms are used to forecast user actions, but parts of the prediction seem to rely on database support and batch training, i.e. it does not seem to be possible to use them in an online manner.

Aura Project

The Aura project [GSSS02] at CMU (Carnegie Mellon University, US) aims to build a “distraction-free”, i.e. unobtrusive, ubiquitous computing environment. It is an umbrella project with different sub-projects collected under its central goal, including well-known projects like the Coda file system [Sat90]. Proactive adaptation of applications has also been explored in [JS03], which describes CIS, the “Contextual Information Service” as a lightweight interface to obtaining context information. Their notion of “Contextual Information Providers” can be compared to feature extractors in our work, described in Section 4.5, or to “cues” in the TEA project. However, CIS follows the approach of adapting the environment (i.e. infrastructure), while we intend to adapt the device to a changing environment, which includes changing user behavior. The use of meta-attributes in CIS at the level of context primitives seems to be an interesting concept; we currently use confidence as a meta-information only at higher levels (after classification and after prediction).

Portolano

The Portolano project [EHAB99] is a testbed for invisible computing at the University of Washington (US) and also divides into a few sub-projects. Related to context awareness is the "Location Stack", a layered model for handling location [HBB02]. In [Hig03] the challenge of automatically labeling and predicting location based on previous work like the Location Stack is described. This is closely related to the focus of this thesis, but we try to generalize from predicting location to predicting context.

Nexus

Nexus is a research project on context awareness (`http://www.nexus.uni-stuttgart.de/`) at the University of Stuttgart (DE) that aims at theoretical foundations for context awareness as well as context-aware applications. They have already defined various XML schemata for representing and querying context data, but most of their results cover mobile and ad-hoc networks and location-based services, only a few include aspects of context other than location. Nexus is focused on location, while in this thesis, location is regarded as one of many different aspects of context. However, hardware prototypes with different sensors like temperature, humidity or brightness are also considered in more recent publications [BBHS03] (incidentally also called “ContextCube” like a model already presented in Section 2.1.2).

Sentient Computing

The Sentient Computing project (`http://www.uk.research.att.com/spirit/`) uses a building-wide ultrasonic location system with mobile devices called "bats" and a fully covering space model of the building. Using information extracted from this location system (user identity, location and nearby persons and things), documents are tagged with meta data to ease a later "context-aware" information retrieval. As a building-wide project, it can rely on powerful infrastructure support like database systems for storing context information.

ETH Wearable Lab (CH)

WearNET [LJS+02] is a distributed, multi-sensor system for wearable computers to collect context information. It uses multiple motion sensors worn on the body, a microphone and other simple sensors like temperature, humidity and atmospheric pressure sensors. With this system, the authors were able to show that different situations and activities could be detected with their combination of sensors, but refer to future work for automatically deriving high-level context information. This thesis presents one possible way to do this; it would be an interesting combination to use the presented framework on top of the context collection from WearNET, since the framework has been designed for exactly such embedded systems and a collection of multiple, simple sensors. Besides WearNET, there is additional research on context recognition based on multiple simple sensors. Currently, much of this is done with accelerometers, e.g. action recognition from accelerometers and microphones has been presented in [LWJ+04]. In a workshop environment, they were able to recognize more complex actions than sitting, standing, walking etc. Actions at that granularity have usually been handled in previous research on action recognition. However, they also used supervised learning methods (HMM for classifying the time series), requiring a second test subject to label activities during the experiments. In the opinion of the author, their use of a detailed script for conducting the experiments during data collection should be followed when sampling further sensor data. It has the advantage that sensor values which mark a specific context are at least roughly reproducible by different test subjects.

Changing Places/House_n project

Also within the Changing Places/House_n project, multiple random test subjects have been used in an experiment [TIL04]. A large number of state-change sensors, i.e. simple switches, were used in multiple homes to detect activity of its inhabitants. While e.g. the group at the ETH Wearable Lab is using body-worn sensors, they are integrating simple sensors into the environment, tackling another area of activity recognition. Most of the requirements stated for activity recognition algorithms, namely “supervised learning”, “probabilistic classification”, “model-based vs. instance-based learning”, “sensor location and type independent”, “real-time performance” and “online learning” are very similar to our requirements on context classification algorithms (cf. Section 4.6.2), first formulated in [MRF03b]. However, we explicitly aim for unsupervised learning and thus have a rather different set of algorithms to select from. In their description of the experimental setup, the authors describe the "experience sampling method", which required test subjects to enter their current activity every 15 minutes. This definitely gives rise to our intention of creating unobtrusive systems; the described experiment already exhibited effects of the monitoring bias in the form that test subjects often did not answer the system’s questions and thus made supervised learning only possible with intervention by researchers.

University of Helsinki (FI)

Currently, the work that has the most similar goals to this thesis has been presented in [LRT04]. The authors aim to recognize and predict location information in an unsupervised, unobtrusive and online way. They are currently using GSM cell data as location information and apply a custom clustering algorithm to generate higher-level location information. This work is similar to our research in that the system is built to automatically recognize context which is somehow important to the user, and, by learning user behavior, predict future context on that level. They are also working without infrastructure support and focus on devices with limited resources, primarily mobile phones. As to create higher-level location information, a density-based clustering similar to the DBSCAN clustering algorithm known from data mining (which is explained in Section 4.6.3) is performed. The authors describe similar issues with recognition and prediction in embedded system like the ones we are facing in our current research, e.g. that only rough estimates of the achieved clustering accuracy are available because no “correct” result can be specified. It is also worth mentioning that some similar ideas for improving prediction have been stated independently, e.g. to take time of day into account instead of only using sequential pattern prediction (cf. Section 4.8.2 for a discussion of the different options for prediction). Their experimental results are currently also indicating that prediction on this level of detail can indeed be performed with limited resources in an online way, but suggest further research to improve the accuracy. In their concluding remarks, the authors also state accuracy issues with their used sensing technology (GSM), which we also expect to be a reason for limited prediction accuracy in our research. At the moment, the most notable difference between our research and the one presented in [LRT04] is that we are not focusing on location but including arbitrary sensing technology, whatever is available on a given device. In personal communication, it turned out that the group from the University of Helsinki is also working on including other information than location into their context sensing software.

University of Washington (US)

In [PLFK03], high-level user behavior is inferred from low-level sensor data by adding knowledge of real-world constraints to user location data. A variant of Dynamic Bayes Networks (DBN) is used in an unsupervised way to predict transport routes based on GPS data. By adding constraints on the routes that could be learned by the training algorithm, the prediction accuracy was significantly improved. However, as we aim to perform prediction at the middleware, independent of the application area, we do not see a general way of incorporating real-world knowledge (that is not learned automatically from the sensor data) a priori into our framework. In the “Activity Compass” application, this was possible because the work was restricted to location prediction with map material being available. Although it might improve context prediction accuracy, this approach does currently not seem applicable to our work on general, application-independent context.

University of Queensland (AU)

A complex, formal model of context with explicitly taking dynamic context (i.e. context attributes changing over time), probabilistic effects and multiple granularities into account has been developed in [HIR01]. It is based on a graph structure of context attributes resembling the well-known entity-relationship model from database design. According to the authors, neither the standard entity-relationship model nor the UML modelling technique are suited well to create models of context, giving reason to the development of their own context model. Recently, they have developed a prototypical framework for handling context information based on their context model, mapping it to a relational database [HI04]. They propose the use of two programming models named *branching* and *triggering*, the former known from context-aware retrieval and the latter denoting the standard use of triggers in computer science, e.g. in database systems. Both models have been integrated in a custom, multi-layered Java programming toolkit, which is currently only partially implemented as proof-of-concept.

Owl

The Owl context service supports heterogeneous context sources, privacy and meta information like age, i.e. the time since the sensor was last sampled, and confidence of context data [EHL01]. It is one of the earlier works that already mentions the possibility of inferring future user behavior from learned habits. To this end, their context service was designed to manage context history in addition to the current context. It is also one of the few projects that explicitly deal with the issue of privacy by implementing access control measures. However, it was still developed as a centralized service with a database system as back end and thus requires a persistent connection between clients and infrastructure components and faces issues of fault-tolerance.

Solar

Solar is a Java middleware system for context awareness [CK02] that is based on the notions of sources, events, operators and applications: sources emit certain events, which are processed by a number of operators before being sent to the applications. Although this middleware has an event-based structure, semi-continuous context can also be realized by forcing sources to emit regular events with their current value. The operators in this model can be compared to reusable methods in code libraries and thus constitute an interesting concept in dealing with context information. Additionally, they provide better scalability due to the distribution of computation over distributed devices and reduce necessary network bandwidth by reusing event streams. However, the system relies on a single, central component called "star", which is responsible for handling application subscriptions to certain events, making the system also dependent on infrastructure components.

ORESTEIA

The ORESTEIA project (`http://www.image.ntua.gr/oresteia/`) is concerned with hybrid intelligent artifacts and aims to create such artifacts that embed local decision capability, online adaptability and communication with other artifacts. Recent publications concentrate on various topics like vibration-driven power generation, inferring underlying functions from data [ABM+03] and on context awareness in software agents [BTK+03]. However, it depends on a priori training of artifacts by a vendor in a special training phase and explicit retraining phases for adaptation [Ore02]; we seek to avoid the distinction of operation and training phases so that a device can be fully operational at all times.

SOCAM

At the National University of Singapore, a "Service-Oriented Context-Aware Middleware" (SOCAM) has very recently been developed [GPZ04b]. It is based on an ontology-oriented approach, i.e. on a vocabulary for representing context, and its model of context is defined using the Web Ontology Language (OWL), allowing to share context between different entities and giving rise to reasoning about context. Their middleware provides the standard services of acquiring, interpreting and disseminating context, but also takes steps towards deriving high-level context from low-level context, which they call "context reasoning" and which can be performed in description logic and first-order logic [HGZP04]. To cope with limited resources in mobile devices, the authors divide possible situations into sub-domains (e.g. home domain, office domain) and switch between the ontologies defined for these sub-domains. However, this middleware is also a service- and infrastructure-oriented approach with a service registry, which might make its use in infrastructure-independent embedded systems difficult. Adapting applications to changed context is performed via the standard way of predefined rules and triggers. The described implementation of the context interpreter, which performs the reasoning process within the OWL model, is suitable for infrastructure-based context services, but not for resource-limited devices. In recent work, the authors proposed a probabilistic extension to OWL and added a Bayesian Network approach to deal with uncertainty in sensor data [GPZ04a]. Although the ontology-based approach and its automatic transformation to a Bayesian Network to deal with uncertainty could offer distinct advantages, currently there does not seem to be a method for automatically deriving this structure of the ontology from sensor data. The designer of a system respectively its application needs to manually define this structure. Therefore, we can currently not employ such an approach when we aim for a general, application-independent and unobtrusive context recognition.

CAMPUS

Another context middleware called CAMPUS has been developed at the Kyushu University in Japan [HNKF03] and consists of two layers, the "real-time context deriving layer" and the "transitive context deriving layer". In the first layer, context is computed from the given sources (sensors or databases) by applying only functions without an internal state. To deal with uncertainty of sensor data like accuracy or freshness, their model of context builds on fuzzy set theory. For each of the quality criteria, a membership function is defined and then scaled by multiplying it with the data value obtained from the source. This method allows to fuse different sensors that represent the same aspect, e.g. two temperature sensors in one room; the functions in the first layer are defined as algebraic sums over the fuzzy sets. In the second layer, building on the results of the first, transitions between situations are handled by mapping the contexts from the first layer to symbols of a formal language and matching a current context trajectory with episodes defined by the applications. The authors plan to implement these concepts in a middleware in future research.

VTT Finland (FI)

In [KKP^{+}03], another approach oriented on mobile devices is presented, which applies sensors embedded into the device instead of the environment and which aims to recognize contexts. By applying a naive Bayes classifier to a set of domain-specific features, they achieved a high accuracy in context recognition for their selected scenarios. The applied sensors were an accelerometer, light intensity, temperature, humidity, skin conductivity and a microphone with a range of audio features. These highly optimized features were used with two Bayes networks, one for the audio context and the second for the other features, constructed with expert knowledge. They applied a supervised approach with an assistant who labeled the data during the experiment and assigned the context classes which have been defined in advance to the experiment, namely seven classes for audio and two indoor/outdoor classes. Due to the use of very specialized features and hand-crafted Bayes network structures, the authors acknowledge that generalization of this approach might be limited. We strictly try to avoid applying expert knowledge for the learning algorithms. However, some of the described audio features derived from the microphone data could prove useful for future research.

COORDINATE

The COORDINATE project [HKKJ02] builds upon previous experiences with the PRIORITIES project, which aims to predict when users will return to their offices if they are currently away. COORDINATE is a network context prediction server based on a central database that provides predictions about presence and availability. While PRIORITIES used Support Vector Machine classifiers for estimating the urgency of messages and considers features extracted from those messages, COORDINATE applies Bayesian network techniques for handling multiple sensors like audio, video, computer activity, calender information and position information. The Bayesian network structure is learned from the recorded data and is used to predict the likelihood of people attending meetings and to estimate the interruptibility of meetings. The authors used parts of the system time for classifying the current situation not unlike the way we experimented with time features (cf. Section 4.5.6), but derived more domain-specific extracts of the system time, namely weekdays, weekends, morning, lunchtime, afternoon, evening and night. They discovered that these extracts describing the time of day are valuable for the prediction. This project is a network-centric infrastructure approach and thus not suitable for embedded systems, but the use of Bayesian networks with automatically learned structure is interesting for prediction in general.

3.2 Pioneering Projects

In this section, the most important pioneering or initiating projects that had an influence on the development of context awareness (as defined in this thesis) are listed. They describe an important part of the history of context computing and help to understand how research on context computing evolved. These earlier projects were characterized by a strong influence of the respective application area, with specialized sensing technology and optimized algorithms — only the Cooltown project started to shift the focus from applications to protocols. However, they laid ground for many of the recent projects described in the last section, which could build upon the collected experiences.

Active Badge

The Active Badge system [WHFG92] is a wearable, personal badge equipped with a microprocessor, an infrared sender and a button. Every 15 seconds, it sends a beacon containing its unique id to receivers distributed in the environment, allowing to gather location information at a central server. It used infrared as primary communication medium, which can be embedded in small devices with limited battery life. These badges have been used at ORL for pioneering applications in pervasive computing, like a building-wide notification system, and have since then been deployed at the University of Kent, Imperial College, London, Lancaster University, the University of Twente, Xerox PARC, DEC research laboratories, Bellcore and MIT Media Lab. The largest system is still in use at Cambridge, with over 200 badges and 300 sensors in the environment (cf. `http://www.uk.research.att.com/ab.html`). As one of the first building-wide location system that can be said to be unobtrusive (the badges are small enough to be worn comfortably), the Active Badge system had a significant influence on the development of subsequent location systems and already defined upper boundaries regarding the size and power consumption of worn devices; future, massively deployed systems should definitely not be any larger or have less battery life. With the early Active Badge installations, even issues like privacy in practical applications or acceptance by test subjects could be tackled due to the large number of badges in use.

Smart Badge system

Inspired by and building upon the experiences with the Active Badge System, the Smart Badge system has been developed [BHMJ97]. Similar to the Active Badge, a Smart Badge has an infrared transceiver and can be worn, but it additionally integrates tilt and heading sensors. Because infrared receivers are also available in addition to infrared senders, the Smart Badge can sense its environment (the spatial proximity of other Smart Badges) and transmit this sensor information bundled with its unique id to the infrastructure, where it is gathered at a network server. This usage of spatial proximity was one of the first in the pervasive computing research field.

Xerox ParcTab

At the Xerox Palo Alto Research Center (PARC), the ParcTab system has been developed, which is a palm-sized PDA with touch screen, complemented by an infrared communication infrastructure with room-sized cells. Most of its applications are executed on remote hosts and thus depend on the communication infrastructure, which also handles location tracking. One of the applications that have been implemented on the ParcTab is a remote control system to control lights and temperature for the current location, others include the better known Forget-me-not system [LF94]. The ParcTab can be seen as a more complicated version of the Active Badge that includes a limited user interface for arbitrary applications. As for the Active Badge, infrastructure support is necessary, but this project still inspired many subsequent publications focusing on context aware handheld devices.

Cooltown

The Cooltown project in Palo Alto [KMS+00] is still one of the most prominent pervasive computing installations. Its main goal is to provide a "web presence" for people, places and things. To this end, things (i.e. devices) are equipped with web servers and URLs are used as the primary way of addressing information throughout the systems. By periodically sending URLs in infrared beacons and sensing those URLs, location awareness is provided to users in the form of location-aware web services. Their use of HTTP as communication protocol and WLAN for physical communication links allows arbitrary clients to access the infrastructure and thus make use of the web presence of other people, places and things. Example applications that have been implemented include the Cooltown museum and bookstore, which allow to retrieve information about real-world items that broadcast URLs, or the Cooltown conference room, which gives access to projectors, printers or whiteboards via URLs. This project has affected research on context awareness in two areas: Firstly, its distinction of physical entities into people, places and things has become one of the most often cited classifications of context aspects, although newer definitions are less location-specific. Secondly, the usage of HTTP and URLs as principles for referencing and sensing shifts the focus in research from applications to protocols. Using standard, well-known and established Internet protocols and only enhancing them by dynamic sensing technology might be an important direction for future context-aware systems.

Neural Network House

Learning user's habits has previously been explored by Michael C. Mozer in The Neural Network House [Moz98], which is able to predict occupancy of rooms, hot water usage and likelihood that a zone is entered in the next few seconds using trained feed-forward neural networks. The context information in the project was again mainly comprised of location, but additional state information from rooms like the status of lights or the temperature set by inhabitants were used. While this project is one of many "smart house" projects, it was one of the first to include prediction of user actions. Although this was done via feed-forward multi-layer perceptrons, trained with standard back-propagation (cf. e.g. [Zel94]) with the known limitations, it showed that prediction of user locations can help to save resources and support users by learning their behavior and automating simple tasks.

3.3 Summary

The last sections gave a survey of pioneering and more recent projects in context computing and their relation to this thesis. Judging from the recent developments of context computing research projects, future context computing middleware will most probably need to standardize in two areas: one standard protocol for communication between involved components (HTTP-based transfer might be appropriate) and a standard representation for modelling context in a structured way (to allow sharing between devices and interpreting context by applications). However, we are not contributing to any standard protocol or model, as we are currently concentrating on embedded systems that perform all necessary computations locally. As can be seen from Table 3.1, TEA and the research at the University of Helsinki are most closely related to the focus of this work, which is on context prediction on embedded systems. Both cope with limited resources and aim at applying adaptive algorithms which do not necessitate a dedicated training phase for learning user habits, but stay flexible during the whole usage period of the device. We can build upon the results from TEA, which is an architecture for online context awareness, and extend it by context prediction and creating an open, extensible framework for context-aware

	TEA	Context Toolkit	Neural Network House	CIS	Location Stack	Solar	MavHome	SOCAM	Activity Compass	Univ. Helsinki	Univ. Queensland	WearNET	COORDINATE	Present thesis
Type	D	I	I	I	I	I	I	D	D	D	I	D	I	D
Software/Framework														
cross-platform		X		1)		X		X			X			X
open/extensible		X		X	X	X	X	X	X		X	X	X	X
intermediate layer	X	X		X	X	X	X	X		X	X	X	X	X
Publicly available		X												X
Limited resources	X							X		X		X		X
Context recognition														
adaptivity	X		X				X			X				X
unobtrusiveness	X									X				X
multiple simple sensors	X	X	X	X		X		X		X	X	X	X	X
deals with uncertainty				X	X			X	X				X	2)
deals with missing values				X				X		X				X
Prediction														
general context														**X**
sequence			X		X		X		X	X			X	2)
periodic pattern							X						X	2)

1) only for clients using the context information
2) supported by core framework, but depends on the algorithm modules

Table 3.1: Properties of related work

applications. Other research like the projects at the ETH Wearable Lab, the MIT Media Lab or the University of Helsinki show the applicability of various sensing technologies and thus ease an selection of appropriate sensors. Projects like the Neural Network House, MavHome, Activity Compass or again the work from the University of Helsinki showed the application of a range of prediction algorithms to specific problems or application areas and are also a basis for our research on general, abstract context prediction. Especially the PRIORITIES and COORDINATE projects delivered interesting discussions about using Bayesian networks for prediction, which should be considered when selecting prediction methods.

In Table 3.1, the properties and features of a selected set of projects, namely those that constitute context computing middleware or deal with prediction, are summarized. The type describes if the project is device (*D*) or infrastructure (*I*) based; this distinction between direct and indirect context has been introduced in [GSB02] and also suits our overview. We declare a software framework to be open and extensible if it supports to easily create new applications on top of it, i.e. if it has been built with support for arbitrary applications in mind, and examine if an intermediate layer acts as an abstraction between sensors and applications, i.e. applications do not need to deal directly with device-specific sensing technologies. Furthermore, we review if the developed software is publicly available by being offered for free download and if it explicitly deals with limited resources of embedded systems. As for the context recognition aspects, we found five properties to be distinguishing among projects: Adaptivity describes that not only a single training phase or expert-driven definition of rules, but constant learning during the lifespan of the system is being applied and the system thus adapts to changed user behavior. Unobtrusiveness stresses that the amount of human supervision should be kept to a minimum by using unsupervised learning techniques. It is further interesting if a system uses a combination of multiple sensors and is able to deal with uncertainty in sensor samples and missing sensor values. For the prediction aspect, currently only this thesis seems to consider predicting general, abstract context identifiers at a high level, but the techniques of sequence prediction and periodic pattern prediction are also used in other projects, albeit at a lower level.

As can be seen from the summarizing table, most of the considered projects are extensible and utilize an intermediate layer, which should therefore be considered as an established approach. Only the Context Toolkit is both cross-platform and publicly available, which are most probably the reasons for its popularity. However, it does not explicitly deal with uncertainty or missing values or support adaptivity within the framework itself. This gives rise to either an extension of the Context Toolkit in these areas or the development of another framework designed to tackle them. Because the focus of the present thesis is on context prediction, a new framework designed specifically with this aim is developed. Table 3.1 shows that a few projects in the area of context computing already utilized prediction methods and should therefore be investigated more closely for research on general context prediction. Especially the MavHome and COORDINATE projects and the work at the University of Helsinki lay a solid ground for further considerations on the use of prediction methods in context computing.

The distinguishing aspects of the work presented in this thesis are context prediction and the combination of an open, cross-platform framework with the concentration on limited resources and unobtrusiveness.

Chapter 4

Context Prediction

Context Prediction is an important aspect of Context Computing that aims at inferring future contexts from past (observed) contexts. In [Fer03a, Fer03b], a general architecture for context-aware applications has been defined. As already shown earlier in Fig. 2.1, sensors and actuators are the interfaces between the physical and the virtual world. In a more detailed view, context computing is situated at the input of pervasive computing applications and processes sensor data to derive high-level information about the current situation, which is then used by applications to act on the environment (cf. Fig. 2.3). As can be seen in Fig. 4.1, context-aware applications will usually need to be designed in a component oriented style. Starting from low-level sensor data, various transformations are applied to infer high-level context information; this step is usually termed context recognition from the application point of view. On top of this context information, statistical and machine learning algorithms can be used to monitor and extrapolate the context history with the aim of learning and applying this knowledge to predicting future contexts. Finally, applications may use past, current and predicted future contexts to control actuators, typically in terms of rules that are dependent on the high-level context information.
Here we are mainly concerned with the context prediction component and, as a prerequisite, also context transformation parts. In this chapter, we introduce the detailed architecture for context recognition and prediction that has been developed within the scope of this thesis. It aims to infer high-level context information from a multitude of heterogeneous sensors. Context rules then describe the influence of those current and future contexts on the actuators, and consequently the behavior of the whole system. The execution of such rules or other means to act on the environment by controlling actuators is outside the scope of this thesis and left to applications built on top of the presented architecture.

4.1 General Concept

A common approach is to regard a system as a black box and define its input/output behaviour. In Fig. 4.2, the inputs and outputs of the function f that denotes the context recognition and prediction parts are shown: the system obtains low-level sensor data by sampling sensors at time intervals and produces high-level context information as user-defined, descriptive labels for context classes. To be able to identify quality criteria for f, we first need to define the input and output data in more detail.

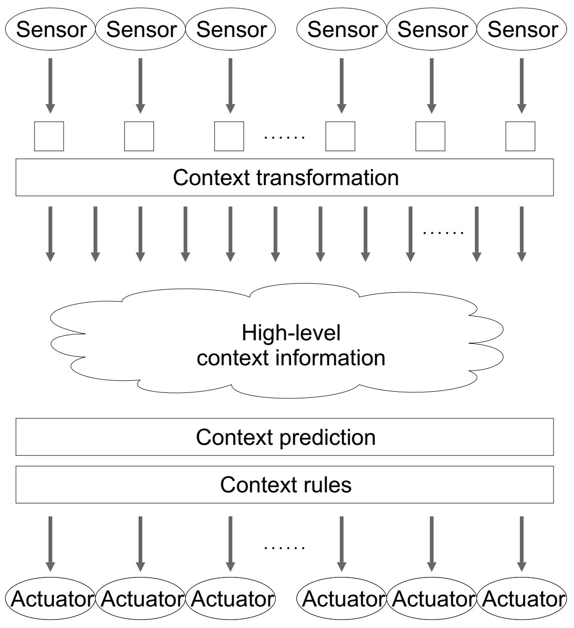


Figure 4.1: General architecture of context-aware applications, after [Fer03b]

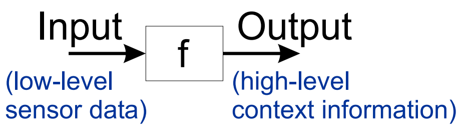


Figure 4.2: Black-box overview of the context prediction architecture

4.1.1 Input Data

The possibly multiple input data streams are heterogeneous and complex, exhibiting the following characteristics:

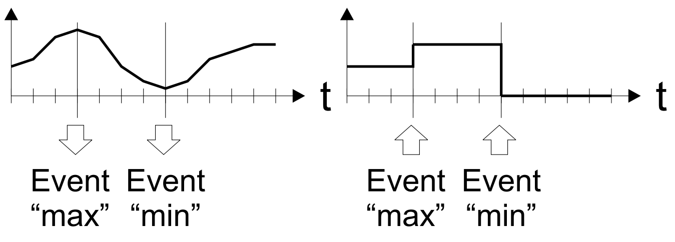
- They are *time series*. A (single-dimensional) time series can be defined as a mapping from points in time to data values:

  $$x : T_0 \to X$$

  where T_0 denotes the set of sample times and X is the set of possible sample values. In [BD02], the distinction between *discrete* and *continuous time series* is explained:

  > *A time series is a set of observations x_t, each one being recorded at a specific time t. A* discrete-time time series *[...] is one in which the set T_0 of times at which observations are made is a discrete set, as is the case, for example, when observations are made at fixed time intervals.* Continuous-time time series *are obtained when observations are recorded continuously over some time interval, e.g. when $T_0 = [0,1]$.*

  For context prediction, we are primarily concerned with discrete time series, because current sensing technology usually demands sampling at discrete time steps to produce values suitable for



(a) Time series to event stream (b) Event stream to time series

Figure 4.3: Conversion between time series and event streams

automatic processing in computer systems. When dealing with sensors, sampling in analog-digital converters will always produce discrete time series. By applying appropriate transformations, the set of sample times can be mapped to integers, i.e. $T_0 \subset \mathbb{Z}$.

- The input is *multi-dimensional* with a possibly large number of input dimensions. Each physical sensor can yield one to many time series, which are combined to an input vector $\overline{x_t}$ for each sample time t.
- Sample values from different sensors can be heterogeneous and of different types. As we mainly work on feature values instead of sensor values in the context recognition part, this heterogeneity is dealt with after a first pre-processing step. It is discussed in more detail below in Section 4.5.1.

An additional type of input data that could be exploited for context prediction are *events*. An event is an occurrence of some external incident and is defined by its time of occurrence and its value. The main difference between samples and events is that sampling is typically controlled by some process integrated in or associated with the system, yielding sample values at known times, while events are generated externally and can not be controlled by the system. An interesting concept would be to analyze the delays between event times and use this information instead of sample values for inferring higher-level information. This would constitute a different type of coding input data in a way similar to Spiking Neural Networks (SNNs) [May02], but is currently not considered as a viable alternative, given the current sensing technology. However, a discrete time series can be converted to an event stream by looking for (known) events in terms of patterns and transmitting the time series data surrounding the pattern within the generated event [RC99]. Examples for very simple patterns would be (local) minima and maxima, as depicted in Fig. 4.3. In the opposite way, an event stream can be converted to a discrete time series by maintaining a state variable that is updated with each incoming event and returns the last known value at each discrete sample time.

4.1.2 Output Data

The output of the system should characterize:

- The current context, optionally represented by a user-defined label (string).
- Predicted future contexts, possibly for different time horizons.

Some applications might need a list of possible current and future contexts with a likelihood measure instead of only a single, “best matching” context. The architecture should be flexible enough to support both output modes, but a list of rated contexts can not be defined as mandatory output; a number of prediction methods that could be used for context prediction are only capable of computing a single context that is deemed to be the most likely one at the respective, future point in time.

4.1.3 Approach

In our chosen approach, which has first been presented in [MRF03b], common patterns in the sensor readings are detected by classification methods. These patterns can then be interpreted as “states” of a state machine that act as context identifiers. A user context is therefore abstracted to these states, whose internal data structures relate sensor readings to abstract identifiers. A related approach has been taken in [HP98], where “affective states” of a user were recognized from a number of sensors. Although this state-based interpretation makes it more complicated for applications to query for specific aspects of a context (e.g. location) instead of the context identifier, it allows to monitor and record the state trajectory of this state machine. When a user advances from one context to another, sensor readings will change and another state will become active, reflecting the context change. Thus, interpreting the context changes as a state trajectory allows to forecast the future development of the trajectory, and consequently to predict the anticipated contexts. To be more concise, within the scope of this thesis forecasting is understood as the computation of the probability distribution for all possible contexts for a future point in time. With discrete, single-step forecasting, this accounts to estimating the probability with which each of the possible contexts will become active in the next time step. For clarity, only the term context will be used in the remainder of this thesis, which is similar to a state in our interpretation.
For the whole system, represented by the function f, we can define some quality criteria for evaluating implementations:

- As context awareness needs to be embedded into a wide range of devices, including small and mobile ones, the system has to cope with limited resources, namely processing power and memory.
- To keep the system unobtrusive, explicit user interaction must be kept to a minimum.
- Context recognition and prediction must be performed online close to real-time.
- The system must cope with varying dimensionality of input data.
- Because real-world sensor data will be used, missing values must be dealt with.
- A reasonable trade-off between computational complexity and accuracy should be found with respect to the application requirements.

By following these criteria, we can subsequently select an architecture and a set of methods to design and implement the system.

4.2 Architecture

Following current best practice in designing an architecture, we use a multi-layer architecture with simple interfaces between the layers. Such architectures have been used successfully in different areas, most notably in networking with the ISO/OSI network layer model, and are also applied in context computing as discussed above. Our approach to context prediction, shown in Fig. 4.4, is based on five steps, separated by simple interfaces: sensor data acquisition, feature extraction, classification, labeling and prediction. In these five steps, recognized current and predicted future high-level context information is inferred from low-level sensor data. Due to the exact definition of the interfaces between the steps, they are mostly self-contained and can be exchanged independently. All of the feature extractors, the classification algorithm and the prediction algorithms have been realized as dynamically loadable shared libraries which can be selected during run-time as proof of concept of the flexibility.

A key issue for the quality of recognized and predicted context is the significance of acquired sensor data; a broad view on the current context, which is usually desired for most applications, requires a multitude of sensors with ideally orthogonal views of the environment. To achieve such a broad view, it seems reasonable to use multiple simple sensors, each capturing a different aspect of the overall context. However, many of the sensors which are currently available to information appliances in personal or mobile computing do not produce numerical output and it is not easily possible to extract numerical features without altering the meaning. Examples for such sensors are Bluetooth or wireless LAN adapters, which can capture parts of the network aspect, the GSM cell id, which defines a qualitative location aspect, or the name of the currently active application (or application part), which captures a part of the activity aspect of a user or device context. A method to cope with this heterogeneity of features has been developed for the general case, independent of the used classification algorithm.

As already mentioned, the general idea for recognizing and predicting context within our architecture is to detect common patterns in the user's or device's situation and abstract those patterns to context identifiers. Thus, unsupervised classification methods are an ideal instrument for our architecture. In the scope of this thesis, the term clustering is used as a special form of classification, namely unsupervised classification. When the current context has been recognized in terms of an abstract context identifier, in our architecture by means of clustering, it can be interpreted as the state of a state machine. The trajectory of this state can then be monitored and used for learning the user behavior and consequently be predicted, yielding future context identifiers for use at the application level. One major advantage of this approach is that predicted future context identifiers can be handled like current ones, and thus context-aware applications can automatically become "future-aware".

The main characteristics of this architecture and the following overview of the different steps have also been presented, in an earlier version, in [May04]. In its current state, it has been optimized towards mobile and embedded systems, typically with restricted GUI capabilities, because this is the more challenging area. The different processing steps can be regarded as filters, transforming input values to output values. No central context repository is necessary, every step is independent and performs online data processing. Thus, the architecture is well suited for resource limited information appliances, but does not exclude the integration of complex modules with high demands on computational resources. When available, powerful processing or storage components can be utilized. Although modules from different steps, e.g. a classifier and a predictor, might use extensive data storage facilities like a central database server, they can do so independently. The interfaces between the distinctive steps are vectors attributed with time stamps; time is passed implicitly from one step to the next. For debugging, evaluation and monitoring purposes, the complete history of these vectors can be logged in a general way and thus allows offline processing if necessary. During normal operation in a device, no central history is necessary as each module maintains its own, internal data model. Typically, this will not cause any duplication of data because the modules in different steps will construct data models with different meanings.

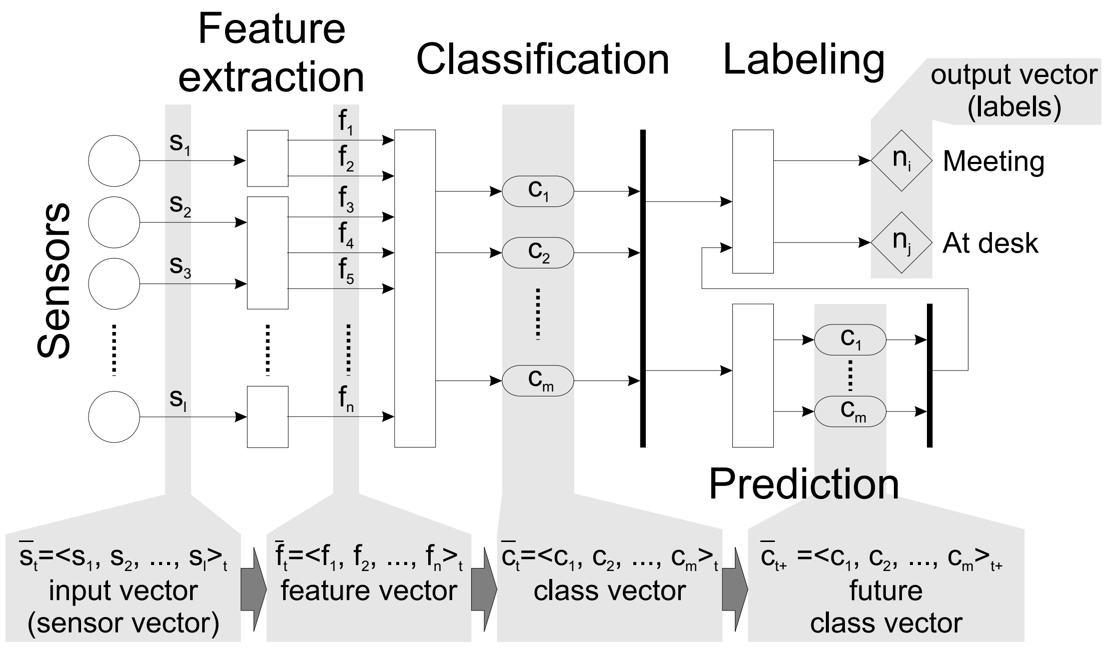


Figure 4.4: Architecture for context prediction

There is an additional, very important issue of context computing that influenced the design of — and is addressed by — this online, filter-type architecture: privacy. Although this thesis is not primarily about privacy, associated concerns need to be addressed starting with the first design considerations of any context-aware system, or user acceptance will be a serious problem. This has already been pointed out early in [BBL$^+$00] from an application point of view:

> *Privacy issues underlie most context-aware applications: the more the application knows, the better in can perform, but the more it knows, the more invasive it is of privacy. Privacy concerns have killed many potential applications already, and will continue to do so.*

In a more recent survey, this has been further elaborated [MR03]:

> *In an environment full of sensors that are keeping track of everybody and everything that is happening, privacy becomes an important issue which can not be added at the end of the development process. Privacy must be taken into account from the beginning.*
>
> *[...]*
>
> *There are two main approaches towards ensuring privacy. The more common approach is to try to make sure that as little information as possible gets outside the system, the second interesting alternative is to restrict the amount of information being acquired and stored to the absolute minimum right from the beginning.*

Our approach is the latter one, to store only as much information permanently as absolutely necessary. By allowing context recognition and prediction to be embedded directly into personal mobile devices, a further step towards broad acceptance is taken by this architecture. The fact that users may feel more comfortable with “smart” personal devices than with a “smart” environment that captures contextual information has also been suggested independently in [MR03]:

> *Another alternative would be to let the environment stay “dumb” and use smart mobile devices or wearable computing equipment to gather and process contextual information, and let these devices instruct the environment to perform certain tasks. It would also allow people to feel to be more in control of what is happening. A person would be able to go totally offline by simply switching off their mobile device or by not wearing their wearable computer.*

Although the architecture supports possibly arbitrary context-aware systems and certain modules can record detailed contextual information of users, it is important to note that personal mobile systems that run in online mode with only minimal permanent storage facilities are expected to achieve better user acceptance.

Main advantages of this architecture with regard to the criteria defined in Section 4.1.3 are that it allows an online operation without explicit training phases, that it is unobtrusive due to the use of unsupervised methods and that it is able to cope with limited resources because no central context repository is required. Furthermore, the modularized architecture allows classification and prediction algorithms to be exchanged easily, which simplifies the evaluation phase for specific applications.
To derive context from raw sensor data, the following steps are applied, which are depicted in Fig. 4.4:

1. Sensor data acquisition: Sensors, e.g. brightness, microphone or IEEE 802.15 Bluetooth and IEEE 802.11b Wireless LAN (*WLAN*) network interfaces, provide data streams (time series) of measurements. Usually some physical values like the incoming RF signals are the base for the measurements, but more abstract sensors like the currently active application can also be utilized. In [Sch02a, p. 78f], sensors are classified as physical or logical, which is not done in this architecture, mainly because the general aim is to provide uniform access to arbitrary sensing technology. Within this work, a sensor is defined as *any entity that can provide measurements of the environment a device is situated in*. Values are sampled at discrete, but not necessarily equidistant time steps $t \in \mathbb{N}_0$, whose frequency should be set to the maximum desired sample frequency of all used sensors. As this is highly application specific, no universally valid distance between sample time steps can be determined.
   A *sensor vector* $S_1 \times S_2 \times \ldots \times S_l$ defines sensor readings $\langle s_1, s_2, \ldots, s_l \rangle_t \in S_1 \times S_2 \times \ldots \times S_l$ for points in time t.

2. Feature extraction: From raw sensor data, information can be extracted by domain-specific methods, yielding multiple data values per sensor which are called *features* F with samples $f \in F$ as a function of time. During feature extraction, the available data is deliberately simplified, transformed or even expanded, allowing for a better interpretation. Usually, simple statistical parameters like the mean $\overline{x}$, standard deviation σ or higher moments are used as features for time series of numerical data. For nominal and ordinal data, alternative methods must be explored.
   The set of features is called a *feature vector* $F_1 \times F_2 \times \ldots \times F_n$, which defines samples $\langle f_1, f_2, \ldots, f_n \rangle_t \in F_1 \times F_2 \times \ldots \times F_n$ for points in time t in the multi-dimensional *feature space*. The number of features n, i.e. the dimensionality of the feature space, is defined by the accumulated set of features that can be extracted from the available sensors.

3. Classification: The task of classification is to find common patterns in the feature space, which are called *classes*. Because a feature vector should possibly be assigned to multiple classes with certain *degrees of membership* (the "probability" that the feature vector belongs to a class), soft classification/soft clustering approaches are utilized. These approaches map a feature vector of n different features to degrees of membership $c \in C$ with $C := [0;1]$ of m different classes: $F_1 \times F_2 \times ... \times F_n \to C^m$. The classes c_i for $i = 1 \dots m$ are regarded as the detected user context and are identified by a simple index in the class vector.
   The *class vector* C^m defines class degrees of membership $\langle c_1, c_2, ..., c_m \rangle_t \in C^m$ for points in time t. The number of classes m should be determined by the applied classification method, based on the given feature space, and is usually not fixed. During run-time, m can increase when new classes are found.

4. Labeling: To ease the development of context-aware applications and for presenting detected contexts to the user, descriptive names should be assigned to single classes or combinations of classes (cf. [LL01]). Labeling maps class vectors to a set of names: $C^m \to N$ where N is a set of user-defined context labels (strings).
   A *context label* $n_t \in N$ describes the currently active context at time t.

5. Prediction: To enable proactivity, our approach is to forecast future contexts. Thus, the prediction step generates anticipated future class vectors from current ones: $C^m \times \mathbb{R}^+ \times \mathbb{R}^+ \to C^m$.
   A (future) class vector defines degrees of membership for each class: $\langle c_1, c_2, ..., c_m \rangle_s = p(\langle c_1, c_2, ..., c_m \rangle_t, t, s)$ for points in time t and s with $s > t$.

It should be noted that, with the above definition of the interfaces, a *flat* context model is assumed. Flat, in this case, means that there are no defined relationships between context classes; all contexts are situated at the same level of abstraction. This assumption stems from the chosen approach of examining context syntactically, disregarding domain specific knowledge about the interrelationships of related or nested contexts. By exploiting such knowledge, context can be analyzed semantically on different levels of granularity, e.g. the finer contexts "writing", "reading", "meeting" and "giving a lecture" could be nested inside the less detailed "at work". Undoubtedly, adding such domain specific knowledge to a context-aware system can improve the results by, on the one hand, helping to resolve possible ambiguities and, on the other hand, allowing applications to work with contexts on different levels of granularity. However, this knowledge, represented by relationships between context classes, has to be introduced into the system by experts for each application domain and is consequently in conflict with our central aim of unobtrusiveness. By changing the definition of class vectors, i.e. the interface between the classification and the labeling and prediction steps, it would be possible to support a *hierarchical* model of context, where *context aggregates* can be built from *context atoms* by composition (cf. [Fer03a]). This more flexible context model would allow a more powerful definition of relationships between contexts. Unfortunately, it is yet unknown if a hierarchy of contexts can be constructed automatically in an unsupervised manner solely from the gathered sensor data, as it is possible with a flat context model. Although hierarchical clustering methods allow the construction of a hierarchical model of the input data, they are typically used in a highly interactive way for manual inspection of data. To allow a broader range of classification methods to be used within this architecture and for simplicity of the defined interfaces, the following considerations generally assume a flat context model with the above definition of class vectors. In the present work, we restrict ourselves to a syntactical analysis of

context instead of a semantical one, accepting the limitations of this approach in favor of unsupervised and unobtrusive operation.
In the following Sections 4.4 to 4.8, the five blocks are described in more detail.

4.3 Focus

This thesis concentrates on a general approach to context prediction that is suitable for a wide range of applications. Domain-specific heuristics might achieve higher accuracy for special applications, but are specific to the application area they have been developed for and can usually not be applied easily to other areas. Thus, the concept and architecture are probably better suited for applications utilizing many different sensors than for applications with a single sensor and well-known, highly optimized, domain-specific methods.

4.4 Sensor Data Acquisition

As context computing is directly coupled with the available sensors, a short survey of state-of-the-art sensor technology is given to motivate the selection of appropriate ones. It has already been recognized in a number of research projects (e.g. [BGS01, GSB02, TIL04, SDA99, LJS$^+$02, LRT04]) that context awareness based on multiple simple, energy-saving and cheap sensors can be more reasonably embedded into end-user consumer devices than when being based on powerful but large or expensive sensors. Additionally, a multitude of different sensors can possibly provide a broader view of the environment than a sensor of one category might be able to. With advances in sensor technology and systems integration, it might be possible to have even more complex sensors like CCD video cameras in unobtrusive, body-worn objects like buttons (cf. [BOST04]). However, for context computing, the use of off-the-shelf sensors, i.e. sensing devices which can be utilized by end users without special adaptations, should be preferred due to broad and cheap availability, leading to a broader scope of applications.
For context awareness in information appliances, sensor fusion does, at the current state of research, not seem appropriate, because the available sensors typically capture different, mostly orthogonal aspects of the user or device context (cf. Section 2.1.3). Fusing of sensors would offer a more complete view on the environment, but necessitates some level of redundancy and a common model, which is currently not available for context descriptions. A possibility for exploiting multiple similar sensors, which can obviously be fused, is to exchange raw sensor or feature vectors between devices in spatial proximity. If two or more devices have similar sensors, their samples can be merged to obtain more robust results due to redundancy. This has been proposed independently in [MHH03], similar to our recommendation in [MRF03b]. Sharing perceptions between nearby, possibly mobile devices demands spontaneous ad-hoc peer-to-peer communication techniques to allow a flexible formation of communication links whenever similar sensors are detected in range. We have proposed an ad-hoc peer-to-peer middleware in [FHMO04a, FHMO04b], which has been optimized for highly dynamic interaction between mobile devices and can be used to share sensor streams.

4.4.1 Sensing Technologies

A vast diversity of sensing technologies is currently available from different fields, but only a subset can currently provide meaningful and exploitable measurements to context computing. In this short survey, we restrict ourselves to sensor technology than could potentially be embedded into information appliances in some form. The following list was inspired by [Sch02a], but has been adapted to our focus of research on embedded systems. It identifies families of sensors; we concentrate on which information that is relevant to context computing can be gained. For details on specific implementations of the respective sensor families, we refer to [Sch02a].

Light (brightness) Light intensity sensors for measuring the brightness of ambient light are small, simple senors which are commonly available. They are currently used in many consumer devices to control the intensity of the display back light, but could be used for additional purposes, presumed that they are accessible to middleware components.

Vision Cameras, e.g. CCD, measure the same physical property as simple brightness sensors, but combine many sensor cells to achieve a spatial resolution. They are becoming widely available in small form factors and usable resolutions and thus get included in end-user devices like smart phones or PDAs. As they constitute the upper range of sensors regarding their expressiveness and data rate, using cameras for context computing demands significantly more processing power, but can yield a large number of features about the environment: ambient light intensity, motion of the camera (i.e. the device or user), motion of objects in front of the camera, dominant color, contrast, etc. Video cameras have already been successfully applied to recognizing high-level context information [CP98].

Audio Microphones in various forms are also common in current devices, mostly targeted for the frequency range of human speech in mobile phones or PDAs, but also capable of recording frequencies outside that range. Audio processing is also demanding in embedded systems and sometimes assisted by special-purpose digital signal processors (DSPs). It is also possible to extract many different features from audio data to assist context recognition (e.g. [CSP98, KKP$^+$03]).

Acceleration Accelerometers have a number of advantages for inferring activity, like moderate power consumption and price, that allow them to be distributed over the body and thus allow to distinguish activity at different granularity levels. They have been used extensively in context computing (e.g. [CCLG02, LWJ$^+$04]).

Location For location sensing, there is an important distinction in the resolution, i.e. granularity, of different approaches. While one class of systems is cell-based, yielding a coarse description of the location e.g. on the granularity of rooms, buildings, or city blocks, the others provide location data in some coordinate system. Most location systems are infrastructure based, including the global GPS/Galileo (coordinates), land-based GSM (usually cell-based data) or high-resolution near-range systems based on infrared, ultrasonic or radio sensing (coordinates). Current, standard networking technology like WLAN or Bluetooth can also be used to provide cell-based and, with restrictions in accuracy, coordinate information. Another important technology is Radio Frequency Identification (RFID), which is primarily an identification technology that is currently being massively deployed in a wide range of applications. RFID can be used either for cell-based location sensing when locations of reference units are known or, with recent enhancements, for computing coordinates of tracked units via triangulation.

Orientation Gyroscopes, magnetic field sensors or tilt sensors can be used to infer 3D orientation information. However, for directly inferring high-level context information, orientation information is too fine-grained. It is nonetheless possible to combine it with other sensors for extracting features from a sensor group.

Proximity Proximity can be seen as a special form of location, namely location relative to the current device or user, but there are some distinct properties in proximity sensing. Classical proximity sensing technologies are based on radar, microwave or ultrasonic but are less common in current off-the-shelf devices due to size and power constraints. Although some of the location sensing technologies are also suitable for direct proximity sensing, or are basically proximity sensing technologies, like Bluetooth, WLAN or RFID, there are additional methods for sensing proximity with simpler means. Touch sensors or body contact (cf. [Zim96]) sensors can be used to detect direct contact, i.e. zero distance or maximum proximity. Particularly touch sensors can be realized with a minimum of technical effort and can thus be implemented in arbitrary mobile devices to detect if they are held or worn.

Environmental conditions Temperature, humidity and air pressure sensors are also rather simple, electro-mechanical sensors with low power consumption. Humidity sensors are less common in end-user consumer devices, but temperature and air pressure sensors are available in more sophisticated watches or mobile GPS receivers and can indicate a variety of situations: e.g. temperature can indicate if a device is in-doors or out-doors, if it is worn near the body etc. while air pressure indicates the altitude.

Force Force sensors measure a mechanical force and are available in vastly different forms, ranging from strain gauges to load cells. Typically, they are realized as piezoresistors whose resistance increases with mechanical strain. There are also field effect transistors that exploit the piezoresistive effect and which can be integrated with CMOS technology. Although there have been some experiments with load cells (e.g. [SLS$^+$02]), they are generally less common in off-the-shelf devices.

Bio-sensors From medical care, a number of bio-sensors are available that could be used to detect emotional aspects of context (cf. Section 2.1.3). Practical considerations require that such sensors must be very easy to apply and that they can be worn comfortably, like skin conductance and heart rate sensing integrated into a watch. This usually precludes elaborate cabling from the physical sensors to a processing unit and gives rise to Personal Area Networks (PANs), e.g. realized by Bluetooth connectivity.

Identity A way to determine the identity of a user is important not only for authentication purposes in terms of computer security, but also for general context-aware systems. Without a notion of the identity, properties of a user profile like interests or past behavior could not be used for adapting the system. An impressive range of sensing technology has been developed primarily for identity recognition, including iris scanning, fingerprint sensors, speaker recognition, face recognition, DNA analysis and other so-called “biometric” systems. For closed environments where there is less chance of identity theft and which have weaker requirements on security, other means of recognizing user identity can be used, most probably in a less obtrusive way. Examples are RFID tags, visual markers, bar codes, infrared badges or even personal Bluetooth-enabled devices identified by their unique MAC address.

State change The simplest possible form of physical sensors are switches, with the main advantage of requiring no power to be operated. Switches can be built in diverse mechanical forms, such as contact or ball switches, tilt sensors or standard buttons, but can also be integrated into connectors to indicate when a connector (like power) is plugged into the device.

In addition to these families of physical sensors, the internal state of a device can also provide important information about the current context, e.g. the currently active application is a strong indicator for the current user activity.

4.4.2 Requirements

There are a few issues that need to be considered when embedding sensors into devices for the purpose of context computing. First of all, there is a trade-off between power consumption and data rate respectively accuracy of a sensor. Simple sensors with no or low power consumption typically exhibit limited accuracy or data rate. Examples are the simple light intensity sensors in contrast to CCD video cameras or touch sensors in contrast to RFID-based proximity sensing. Another practical issue is the size of a physical sensor. Some application areas have upper bounds on the size of devices, like the area of mobile communication which defines clear limits for the maximum weight and size of mobile phones that are acceptable to customers.
As already indicated by these issues, the requirements on sensing technology are highly specific to application areas and it is difficult to define general ones. Instead, the context sensing layer should be flexible enough to deal with arbitrary sensors and not impose any additional restrictions. Nevertheless, care has to be taken when using remote sensors that they are still within a close range to the place where context awareness is aimed for. When remotely connected sensors or shared perceptions of remote devices are utilized, it has to be ensured that their measurements still represent the context of the device which performs the context recognition. In this regard, “close” does not necessarily correspond to immediate spatial proximity but is defined by the application, e.g. the air pressure is usually almost identical within a whole city block (at street level), so that remote sensors within that area can be used. For the internal state of a system, the notion of “close” is even independent of the geographical location, e.g. a group of people logged into a CSCW environment will share an approximately similar application and activity context although they can be geographically dispersed.

4.5 Feature Extraction

Feature extraction is probably best known from image processing (e.g. [Bov00]), general pattern recognition, robotics and speech recognition. The task of feature extraction is to reduce the complexity of raw sensor data by computing a number of properties of the signal that describe it. By applying domain- and sensor-specific methods, the raw sensor data is transformed into more meaningful values, which are typically represented as real-numbered feature vectors. Usually, the representation of sensor data in the associated feature space is more compact, saving computational resources in the subsequent classification. There exist some common features that can be used to differentiate one- or two-dimensional signals and are applicable to all types of such signals. Examples are simple statistical functions like minimum, maximum, mean and median values or the standard deviation. Such simple features have been applied successfully to context recognition with physical sensors that yield real-numbered values (e.g. [Lae01]). However, more advanced features are specific to certain signal types and can only be extracted with domain-specific knowledge. Examples of such highly domain-specific features are histogram difference or scene change categories in video indexing [Bov00, section 9.1], ridge endings or ridge bifurcations in fingerprint classification [Bov00, section 10.5], anatomical features in face recognition [CGY96], mel-scaled filter-bank coefficients in sound classification [CSP98] or WLAN access point MAC addresses in range and GSM cell id for recognition of cell-based location context [MRF03a]. It is already well documented that feature extraction can vastly improve the results [Bov00, page 411]:

> *The quality of feature selection/extraction limits the performance of the overall pattern recognition system. [...] Thus feature extraction is the most crucial step.*

Therefore, domain-specific knowledge about sensors can — and should — be exploited to select an appropriate set of features to be used in this step and thus improve the overall results. Although this selection is crucial for the whole context prediction process, it does not include any higher-level modelling of interrelationships between contexts as discussed above in Section 4.2; it is limited to selecting and applying a specific set of features suitable for the chosen application area and can thus be done more easily for new application domains where context models are not yet known.

4.5.1 Types of Features

Compared with feature extraction from video or audio data streams, feature extraction in context computing faces an additional complexity due to the simultaneous utilization of different families of sensing technologies. As previously described in [MRF03a], some of these sensors do not yield single, numerical values but data with a more complex structure. Some important information that can be extracted from these sensors is categorical (nominal) and non-atomic, e.g. the list of Bluetooth or WLAN MAC addresses which are currently in communication range. Nonetheless, it could provide useful information for determining the current user context. The specific example of recognized Bluetooth MAC addresses, which is described by a flat but a priori unbounded “list” feature, describes a significant part of the networking and additionally a part of the location context. Non-numerical features can provide valuable information for context recognition and prediction and should thus not be neglected. Due to this mixture of different sensor types, the feature vector $\langle f_1, f_2, ..., f_n \rangle_t$ formed by an arbitrary combination of domain- and sensor-specific features is highly heterogeneous. We can distinguish the following types of features by their possible sets of values:

- *Nominal* (categorical, qualitative): values of a set F on which no order relation has been defined. Nominal features only need to define an equivalence relation, which is reflexive, symmetric and transitive. A special case are binary features with $F = \{0, 1\}$. This type of features poses only minor requirements and is thus in practice applicable to arbitrary kinds of features.

- *Ordinal* (rank): values of a set F with a defined total order relation $\leq$ in addition to the equivalence relation. A binary relation R on a set F is a total order if and only if it is reflexive, antisymmetric, transitive and for any pair of elements $a \in F$ and $b \in F$, $\langle a, b \rangle \in R$ or $\langle b, a \rangle \in R$. Due to the additional requirement of a defined order, this type of features allows to determine the median of a set of values.

- *Numerical* (quantitative): values of an ordered set F with defined $+$ and $\cdot$ operations. Numerical features are an algebraic field, which demands closure of F under $+$ and $\cdot$, associativity for $+$ and $\cdot$, commutativity for $+$ and $\cdot$, distributivity of $\cdot$ over $+$, existence of additive identity, existence of multiplicative identity, existence of additive inverses and existence of multiplicative inverses. Typical examples for such fields are integers, rational numbers or real numbers. We can further distinguish according to the density of values in the set between *discrete* ($F \subseteq \mathbb{Z}$) and *continuous* ($F \subseteq \mathbb{R}$) features. This type of features allows to compute statistical moments like the mean (average) or variance of a set of values.

- *Interval*: intervals instead of single values, i.e. $F = \{\langle a,b\rangle \mid a,b \in G\}$ where G is another set, e.g. $G = \mathbb{R}$. An equality relation can be defined trivially between intervals, as can be operations. Intervals are well known in *interval arithmetic* [Kea96] and can be used to determine bounds of values. For intervals based on $\mathbb{R}$, an order relation can also be defined. For real-world sensor data with known accuracy bounds, intervals are well suited and can often be used as a direct but more powerful replacement to numerical values.

The above feature types are all *atomic* (in the same sense as atomic attributes in the well-known first normal form of relational database design, cf. e.g. [Wik]), which means that elements of the respective sets can be expressed with a fixed number of variables (two for interval types). But some sensors yield *non-atomic* values like the list of Bluetooth MAC addresses in communication range, which are nominal values. Such list features are sets $F \subseteq \mathcal{P}(G)$ where G denotes the basic set and $\mathcal{P}(G)$ the power set of G, e.g. the set of all possible MAC addresses.
If non-numeric values should be used as input to a classification algorithm which can only work with numerical inputs, they need to be coded. The standard procedure is to transform all non-numerical into numerical input dimension by assigning one input dimension for each possible feature value, i.e. the *one-of-C* method. E.g. for the WLAN ESSID or the currently active application (represented by strings), the single, nominal dimension can be split into as many binary dimensions as there are different values of the respective feature, where, at each time step, only one of those dimensions is set to 1 and all others must be set to 0. This has been applied successfully to categorical data with a bounded set of values (e.g. department, color, nationality), but is problematic when the set of possible values is not known in advance or too large to be coded with binary inputs (e.g. WLAN MAC addresses would need 2^{48} input dimensions to cover all possible values). Therefore, coding categorical data as numerical inputs does not seem to be the best solution and a better method should be found.
In our architecture, a feature type is defined by the feature extractor that is producing the respective features. All feature extractors are independent of each other. They typically operate on different sets of sensors, but it is also possible to use multiple different feature extractors on the same sensor time series. As no standard interface for the variety of raw sensor data streams has been defined, such cases have to be handled by the respective feature extractors in a sensor-specific way. However, because each feature type is defined by its extractor, the necessary transformation of the internal type to the external interface expected by the classification algorithm can also be implemented independently and in a domain-specific way. In the following, the external interface to the classification step, which defines the requirements on each feature, is explained in more detail.

4.5.2 Requirements

The transformation of arbitrary feature types to a common interface for classification algorithms is an abstraction of the feature details and thus allows all heterogeneous features to be unified in a multi-dimensional, homogeneous feature space that can be classified by commonly used, well-established algorithms. One of the issues with such an abstraction is that the range of some sensors can not be determined a priori, e.g. Bluetooth or WLAN interfaces, when used as a proximity sensor, yield one binary value for each already known peer, with the number of known peers increasing over time. Such

a non-atomic, growing list of values can not be mapped to a single numerical type while preserving the semantic ordering. Common classification algorithms depend on a defined order of the given input values to achieve a reasonable distinction into classes, but a mapping would discard the connections between values of the internal feature type. Thus, simply mapping any type to a numerical type and then executing the clustering on the mapped values would impair classification accuracy.
A comparison of different classification methods, ranging from clustering algorithms to neural networks, showed that only two operations are necessary on a feature F: a distance metric and an adaptation operator (this matches the results of other research [NB01]). With these two operations, common supervised and unsupervised classifiers can be easily applied to any feature which defines them.
In the following, we will define both operations in more detail, as already presented in [MRF03a].

4.5.3 Similarity Measure

The first operation that needs to be defined on each feature, i.e. on each dimension of the feature space formed by feature vectors $\langle f_1, ..., f_n \rangle \in F_1 \times F_2 \times ... \times F_n$, is the similarity measure or *distance metric*.

$$\begin{aligned} d \quad : \quad & F \times F \rightarrow [0;1] \\ & \delta = d\left(f_1, f_2\right) \end{aligned}$$

defines the distance between two samples of the feature F, normalized to $[0;1]$; the normalization is not necessary, but eases the classification step. A general distance metric has to fulfill non-negativity, identity, commutativity (symmetry) and the triangle inequality. We would like to note that the range of values in F can change (increase) during run-time, thus the distance of two given samples can also change due to the normalization. Although this is no problem with the classification methods we have studied, others might need modifications.

4.5.4 Adaptation Operator

Additionally, the second operator adapts/modifies a point (in one dimension) and is necessary for supervised and unsupervised learning.

$$\begin{aligned} \alpha \quad : \quad & F \times F \times [0;1] \rightarrow F \\ & f' = \alpha\left(f, g, a\right) \end{aligned}$$

modifies the sample $f \in F$ towards $g \in F$ by a learning/adaptation factor $a \in [0;1]$. It is further defined that $\alpha\left(f, g, 0\right) := f$ and $\alpha\left(f, g, 1\right) := g$, i.e. the minimum and maximum adaptation cases.

4.5.5 Examples for Feature Dimensions

This section gives some example definitions of both operators for a selection of the features we are currently using in our experiments. They show numerical, binary and list feature types.

1. *Signal strength*: The Bluetooth, WLAN or GSM signal strength is a numerical, continuous variable; therefore, we can apply the L1 (Manhattan) metric:

$$d\left(f_1, f_2\right) := \frac{|f_1 - f_2|}{F_{max} - F_{min}}$$

and

$$\alpha(f,g,a) := f + (g - f) \cdot a$$

for samples $f_1, f_2, f, g \in F$ with maximum and minimum values F_{max} and F_{min}.

2. *Associated to access point*: This is a binary variable with values $f \in \{0,1\}$; the distance operator can thus be simplified to the equality relation: (NOT XOR in boolean algebra):

$$d(f_1, f_2) := \begin{cases} 1 & if \quad f_1 = f_2 \\ 0 & if \quad f_1 \neq f_2 \end{cases}$$

A simple adaptation operator could be defined as

$$\alpha(f,g,a) := \begin{cases} f & if \quad a \leq 0.5 \\ g & if \quad a > 0.5 \end{cases}$$

which only sets it to one or the other value, depending on the current value of the adaptation factor a. A better, although more complicated variant is to implement the operator with state so that it internally sums up a and only modifies the feature value after a reaches a certain threshold.

3. *Number of MAC addresses in range*: The number of Bluetooth peers or WLAN access points in range is a numerical, discrete variable; we define

$$d(f_1, f_2) := \frac{|f_1 - f_2|}{F_{max}}$$

and

$$\alpha(f,g,a) := \begin{cases} \lceil f + (g - f) \cdot a \rceil & if \quad f \geq g \\ \lfloor f + (g - f) \cdot a \rfloor & if \quad f < g \end{cases}$$

for a number of already detected, different MAC addresses F_{max}. As explained earlier, F_{max} can increase during run-time when new, previously unknown addresses are found.

4. *List of MAC addresses in range*: The list of Bluetooth peers or WLAN access points in range is a nominal, non-atomic variable. Due to the non-atomicity, there are various ways for coding; but each value $f \in F$ can be seen as a set of addresses. For a list of addresses, we apply the Hamming distance (the number of different addresses)

$$d(f_1, f_2) := |(f_1 - f_2) \cup (f_2 - f_1)|$$

when f_1 and f_2 are sets. The adaptation operator can be arbitrarily complex; our current operator changes, according to the adaptation factor a, a randomly selected fraction of the different addresses in f and g to the addresses in g by adding and/or removing addresses in f. In practical implementations, it is advantageous to code the list of MAC addresses as a bit vector, where addresses that have been gathered so far correspond to bit positions, which reduces the distance metric to the computation of the bit-sum

$$d(f_1, f_2) := \sum_{i=0}^{k-1} \begin{pmatrix} 1 & if & f_1^i \neq f_2^i \\ 0 & if & f_1^i = f_2^i \end{pmatrix}$$

between bit vectors of increasing length.

It can be easily verified that the formal requirements on both operations are fulfilled by the above definitions.

4.5.6 Time Features

One important aspect of context is generally the time, e.g. indicating if it is work or spare time, if there are holidays or if the user is probably asleep. Various extracts of the system time in terms of a time stamp could be used as independent features as well, e.g.:

- season
- month
- day of the week
- day of the month
- hour
- minute
- second

We speculate that these additional input dimensions might allow the clustering algorithm to find user habits, i.e. temporal patterns, more easily. To evaluate this speculation, we conducted a short experiment, feeding the implemented clustering algorithm with the extensive data set described in Chapter 6, one time with additional time-based features and one time without. The results in terms of the final classification error clearly indicated that adding time-based features was disadvantageous for the clustering algorithm, at least for this data set. Thus, we decided not to use time features for further experiments with a similar setting, but they should be re-evaluated for other application areas. Due to the module-based architecture and its consequential implementation, switching features on or off for the classification algorithm can be done easily via configuration options.

4.6 Classification

The purpose of classification in general is to classify input vectors into a set of classes and it is usually used to decrease the dimensionality of input data. There are two kinds of classification, depending on the training data: *supervised classification* to separate data points into known classes and *unsupervised classification*, also called *clustering*, to find previously unknown classes, solely based on the given input data. Since we aim for unobtrusive operation of the whole system, supervised methods that demand an a priori definition of context classes or continuous questioning of the user can not be applied. Thus, we apply unsupervised classification methods, with the known disadvantage that "correct" classification results are not known and, as a consequence, the evaluation and comparison of different methods is more difficult.

4.6.1 Introduction

As depicted in Fig. 4.4, classification is the third step in our architecture and builds directly upon the results of the feature extraction step. Its task is to find patterns in the feature space, which is spanned by the feature vectors. Patterns in the feature space are accumulations of points that can be found by unsupervised classification methods, or, as defined in [Nel01, p. 142]: *a cluster can be defined as a group of data that are more similar to each other than data belonging to other clusters*. When certain regions of the feature space are well populated, this indicates situations, described by their corresponding sensor values and the derived features, which the user or device regularly is in. However, as real-life situations are always slightly different even on repetition of the same tasks, the features extracted from sensor readings will also be varying. Therefore, points in the feature space will not be equal, but will be distributed over some region within the feature space. A classification algorithm should detect such regions of accumulation and separate them from other regions that might belong to different situations and thus to different context. The classes found automatically by the classification algorithm are regarded as the different contexts.

4.6.2 Requirements

Context recognition by classification poses some requirements on the utilized clustering method, making many well-known techniques unsuitable for this area. Because the intended context recognition should be performed unobtrusively, the algorithms used for computation have to be mostly black-box, with no tuning or parameterization being required by the average user. Therefore, the main requirement is to yield useful results with no user intervention at all. It has to be noted that it will be possible to find algorithms yielding significantly better results for any given scenario, when human experts can intervene to tune the algorithm parameters. This thesis does not intend to produce perfect recognition of current and prediction of future context, but to do it with a minimum amount of explicit user interaction. A subset of the following requirements was first presented in [MRF03b], but they are described in more detail here:

Online, life-long learning For information appliances, there is no dedicated training phase like for a speech recognition engine that needs to adapt only once to a speaker. Since such a device has to be operational at all times, learning has to be done continuously and in an unsupervised way. Batch algorithms which iterate over the whole data set at once can not be applied. In [Ham01], this problem has been taken up:

> *A shortcoming from the biological, as well as the technical, point of view originates from the artificial separation of a lifespan into a learning and recognition phase.*

For any practical application, learning has to be performed online and almost in real-time; it is not feasible to log training data, in our proposed architecture in terms of time stamped feature vectors, and process it offline. Therefore, the classification algorithm must immediately use new data as soon as it is received. Because of limited storage capability, it is infeasible to store all past feature vectors. Each new data point must be immediately incorporated into the internal data model so that it can be safely discarded before the next sample time. This means that an online classification algorithm, when applied in an offline manner, has a run-time complexity of $O(n)$ — every algorithm with a higher complexity will most probably not be usable in an online, i.e. incremental, way.

Adaptivity Learning must never stop; as the user's habits will change over time, classes must always adapt to new feature values. This demands that all parameters of the classification algorithm are either constant, i.e. chosen at implementation time without the need for modifying them during run time, or self-adaptive. Continuously increasing or decreasing parameters like a "learning rate" of many common neural network algorithms like the Kohonen SOM would be limited after a certain time of learning and thus prevent an algorithm from running constantly for arbitrary periods of time without interruptions.

Furthermore, it is computationally infeasible to re-compute the internal data structures, including all classes, whenever an input (feature) vector is presented to the algorithm. Therefore, the algorithm must adapt existing classes to slowly changing input patterns like long-term trends, while short-term changes should form new classes instead of modifying existing ones. All parameters have to be adaptive, e.g. a learning rate should exhibit a long-term, adaptive changing behavior instead of a hard-coded monotonous decrease.

Variable Topology Because the number of classes can not be determined a priori for the general case, the internal structure and, in the case of connections between classes, the internal topology, must be able to adapt dynamically. A good summary of the advantages of so-called growing networks, which specialize on a variable internal topology, is given in [Fri96]: there are fewer parameters to define and it is possible to interrupt the self-organization process, i.e. learning, and to continue it at a later time since there is no distinction into different phases. The former advantage describes that the internal structure does not need to be specified in advance, but is formed during learning, which is important for devices that should adapt to their user. The number of classes for matching input data should be handled dynamically, increasing and decreasing it during run-time without recomputation of existing classes.

Clusters with arbitrary shapes The algorithm should be able to find clusters with arbitrary shapes, including "holes", where an area enclosed by data points belonging to a specific cluster does not belong to this cluster itself. As real-world situations can vary in arbitrary ways while still constituting the same high-level context, the clustering algorithm should be able to adapt to such feature vectors that are dispersed but still belong together.
In the possibly high-dimensional input space, the algorithm should be able to find clusters in sub spaces of this input space, as for some context, certain dimensions might not contribute at all to the characterization of the high-level context. With arbitrarily shaped clusters, sub-spaces can be found when a cluster extends over a whole dimension of the feature space. When the algorithm is unable to deal with either arbitrary shapes or explicitly with sub spaces, handling a large number of dimensions becomes increasingly difficult, as the separation of classes in a high-dimensional space easily leads to overfitting and thus poor generalization. This well-known "curse of dimensionality" (e.g. [Fri97a]) describes the exponential growth of hypervolume of a function with its dimensionality and has also been acknowledged in [Lae01] for the area of context recognition via clustering.

Soft classification Context classes are not mutually exclusive, but more than one context can be active at the same time (e.g. "at home" and "sleeping"). Therefore, the output of the classification step has been defined to specify the degree of membership (cf. [Nel01]) of each context class, i.e. a measure on how well the classes match the current context, much along the same lines as the "fuzzy membership" described in [MBS93]. The difference between hard and soft classification is that the former assigns each input vector to exactly one class while the latter computes a degree of similarity to each output class. Consequently, we utilize soft clustering algorithms in this architecture.

Noise resistance Owing to the noise that is intrinsic to physical sensing technologies, all algorithms in this architecture have to cope with erroneous sample values. In unsupervised learning, where the true output value, which in the case of clustering means the correct class memberships, is not known, it is difficult to distinguish noise from different classes. A sample value that is far off the boundaries of an

automatically formed class could either mean a sudden switch to another class, which we interpret as the transition to another context, or it could be an erroneous sensor value. Ideally, a classification algorithm should detect noise and discard it or assign it to a special class (this method of assigning known noise to a special class is used regularly for supervised classification).
A special case are missing values, which occur when a sensor is unavailable at sample time. Missing values are also considered as noise, but can be more easily detected than erroneous sensor values even in unsupervised learning.

Limited resources Any algorithm that should possibly be executed on an embedded system has to work within tight resource limitations. These resource limitations mainly include storage capability and processing power, but other factors like limited battery life, which prohibits long calculations, need also be taken into account. Aside from the run-time memory requirements, the amount of persistently stored data should also be minimal to limit the overall memory consumption.

Simplicity In our case, the algorithm should perform as few distinct operations on feature vectors as possible. As the feature extractors have to provide the necessary operators (cf. Section 4.5.2), a multitude of operators complicates their implementation. A simple internal structure of the automatically formed clusters eases their interpretability, which is desirable as to present users a human readable form of the context classes. Ideally, expert users could be able to tune the class parameters to match their common contexts best, but this would not only demand to interpret internal structures of the classification algorithm, but also to modify them, which may be difficult for more complex algorithms. Furthermore, the number of parameters that need to be specified for an algorithm should be as low as possible, because it is difficult to optimize them when the feature vectors, and thus the shape of the feature space, are not known in advance.

Privacy For privacy reasons, It should be impossible to deduce exact, past user behaviour from context classes or the internal data structures. This implies that long-term time series of audio, video, etc. should not be recorded and stored, even if the storage capabilities would allow to do so. A responsible handling of user privacy is crucial to wide acceptance of context-aware systems.

4.6.3 Survey

In this section, a short literature survey of classification algorithms that might be suitable for the purpose of context recognition is given. Supervised algorithms have not been considered for the classification step, as they would counteract our striving for unobtrusiveness. As pointed out in [TIL04], users are usually not willing to continually provide labels for their current context, which are a prerequisite to applying supervised methods. Based on the above requirements, this literature study has the aim to select the most promising candidates for implementation and a later quantitative comparison.

K-Means The k-means clustering algorithm is one of the simplest, yet best known clustering techniques, sometimes also called c-means. Its name stems from the fact that the “k” or “c” denotes the fixed number of clusters that the algorithm will compute, which has to be specified a priori. By an iterative batch training, it minimizes the following cost function [Nel01]:

$$I = \sum_{j=1}^{k} \sum_{i=1}^{N} \mu_{ji} \left\| \underline{u}(i) - \underline{c}_j \right\|$$

where k is the number of clusters, N is the number of samples, i.e. the size of the training data, and $\mu_{ji} = 1$ when the data sample $\underline{u}(i)$ belongs to cluster j and $\mu_{ji} = 0$ otherwise. As can be seen, it simply minimizes the sum of all quadratic distances of each data sample to its associated cluster center, which is also called cluster prototype in some publications. Various extensions or modifications to k-means have been proposed, e.g. for better performance [ARS98] or for dynamically growing the number of clusters [DWM02]. K-means can be used in an online mode and is, due to its simplicity, very fast and copes well with limited resources, but dues not fulfill our core requirements of soft classification or variable topology. As there is no other parameter than the number of clusters, it could even be seen to be adaptive.

Fuzzy C-Means (FCM) As a direct extension of the k-means algorithm, fuzzy c-means minimizes the cost function [Nel01]:

$$I = \sum_{j=1}^{k} \sum_{i=1}^{N} \mu_{ji}^{v} \left\| \underline{u}(i) - \underline{c}_j \right\|$$

with $\sum_{j=1}^{k} \mu_{ji} = 1$, but u_{ji} are no longer required to be equal to one or zero, but can assume some degree of membership in $[0;1]$ and thus make FCM a soft clustering approach. The power of v determines the fuzziness of the clusters in the interval $[1;\infty]$, which is usually chosen to be 2. Lower values enforce degrees of membership close to 0 and 1, i.e. a sharper separation. In addition to being an online algorithm, still simple in terms of internal structure and fast, FCM fulfills the requirement of soft classification, but fails on the variable topology and on the adaptivity aspects. The number of clusters k as well as the fuzziness v need to be determined a priori by the developer, which can not be done satisfyingly without prior knowledge of the data set.

Neural Gas (NG) The Neural Gas algorithm is another extension to the k-means clustering algorithm, which takes a “neighborhood ranking” into account [MBS93]. By not only adapting the “winning” cluster center to a new data point but all clusters depending on their proximity to the new input, NG copes better with nontrivial data distributions and generally reaches a lower cost function than the original k-means. In each step, as a new data point is presented, the cluster centers are ranked according to their distance to the data point, which makes NG also a soft clustering approach like FCM. When the neighborhood ranking is limited to only the winning cluster, i.e. the nearest one, NG is equal to the classical k-means [DWM02]. NG fulfills the same requirements as FCM, still requires an a priori definition of the number of clusters and also necessitates a training parameter to be chosen, in case of NG a decay constant for monotonously decreasing a learning rate. However, due to the ranking of cluster centers, a sorting of a list of N prototype vectors has to be performed, making NG slower than k-means and FCM by a factor of $\log N$, but with the gain of a lower cost function and thus better approximation of the data samples.

Growing Neural Gas (GNG) The Growing Neural Gas algorithm is a further enhancement of NG and belongs to the family of Cell Structure algorithms, which also includes Growing Cell Structures (GCS) and Dynamic Cell Structures (DCS). GNG differs from NG in two important ways: it addresses the requirement of a variable topology by allowing the network to grow clusters as necessary and it forms a graph between the cluster centers, which reflects the topology of the input data distribution. This topology is used to adapt and insert clusters, in contrast to NG, where clusters are adapted based on their distance to the new data point. Because k-means, NG and GNG all minimize the same cost function [DWM02], their classification results can be easily compared and the trade-off between computational cost and accuracy can be evaluated quantitatively. As GNG extends NG to also fulfill the requirement of a variable topology, and can, due to the constructed topology, cope with arbitrarily shaped clusters, it is the best candidate so far and is described in more detail below in Section 4.6.4.

Kohonen Self-Organizing Map (SOM) The Kohonen Self-Organizing Map is a variant of the Vector Quantization (VQ) technique, also developed by Kohonen. VQ is again quite similar to k-means in online mode, updating the the "winner" cluster towards the new input data. Although the terminology is different, as the clusters are named neurons for VQ, the basic concepts are very similar. The SOM extends VQ roughly like NG extends the basic k-means clustering: a *p*-dimensional topology is established between the fixed number of clusters and, in addition to the winner neuron, also neurons in the neighborhood are modified by new data points. This neighborhood function, most often used with a 2-dimensional topology, as well as a global learning rate are decreased during learning, which violates the requirement of adaptivity. Because the SOM is able to preserve a topology in the input data distribution and map it to the typically 2-dimensional output topology, it is often used for visualizing high-dimensional data. For the purpose of recognizing context by classification, the mapping to a lower-dimensional space is not necessary and might even be counter-productive due to the information loss in the topology. In its standard formulation, the number of neurons, i.e. clusters, also has to be determined a priori, failing the requirement of variable topology, but some variants that overcome this issue have been proposed, e.g. the Growing Grid (GG) [Fri95a]. Such variants fulfill the core requirements like GNG, but are still less flexible in their growing process. Therefore, GNG seems to be a better candidate for the classification step.

Spiking Neural Network (SNN) A recent branch of neural networks, the Spiking Neural Networks, have also been applied to unsupervised clustering. SNNs differ from Artificial Neural Networks (ANNs) in the type of information they handle. While ANNs traditionally work on real-numbered input vectors which model the firing rate of natural neural networks, SNNs explicitly take time into account by processing so-called spike trains (cf. [May02]). These spike trains better model the firing of neurons in natural neural networks like the human brain and it has been recognized that SNNs might be better suited for fast information processing than ANNs. Encoding information in single spikes — in their relative times of occurrence — is one of the possible coding schemes in SNNs called "temporal coding". The advantage of using this temporal coding for implementing well-known network structures like the SOM (cf. [Ruf98]) or RBF networks (cf. [BKP02]) is that adaptation of neurons, i.e. clusters, can be done locally. This gives the possibility of very efficient implementations of such networks directly in VLSI, and thus of high network performance at low processing power. A multi-layer RBF network with temporal coding has been shown to slightly outperform classical k-means and the SOM on Fisher's Iris data set [BKP02]. In general, SNNs are an interesting type of neural networks, but only recently, methods for the evolution of the network topology have been proposed (e.g. [UPRS03] evolves SNNs with Genetic Algorithms). With the current state-of-the-art in SNNs, the network structure needs to be defined a priori, again violating the requirement for variable topology.

Adaptive Resonance Theory (ART) Adaptive Resonance Theory is a series of networks for unsupervised classification and has been developed to overcome the "stability/plasticity dilemma" (cf. [Gro76]), i.e. the trade-off between convergence of the learning algorithm to some optimal solution and adaptation to new input data that is present in classical networks like the SOM. ART networks solve this dilemma by introducing the so-called "vigilance" parameter, which controls the boundary between inserting a new cluster for representing new data and adapting old clusters. They are similar to vector quantization, but implement a growing approach like NG or GNG. For new data samples, the "winner" cluster is only adapted when its distance is lower than the vigilance parameter, else a new cluster is inserted at the place of the new input point. Obviously, this vigilance parameter is crucial to the accuracy of the classification, and it must usually be determined by trial-and-error [Nel01], violating the requirement of adaptivity. Additionally, ART networks are very sensitive to noise, making them less suitable for handling real-world sensor data.

IncrementalDBSCAN (IDBSCAN) The IncrementalDBSCAN algorithm is an optimization of DBSCAN for better performance, with a run-time complexity of $O(n)$ instead of $O(n^2)$ [EKS$^+$98]. DBSCAN is a well-known, density-based clustering algorithm from the field of data warehouses, which is applicable to any data set that is embedded into a metric space. A metric space only demands the definition of a distance function for any pair of data points, while e.g. the BIRCH algorithm is applicable only to Euclidean vector spaces (cf. [EKS$^+$98]). The DBSCAN algorithm finds clusters in a data set according to their density and is also able to separate noise from clusters, and IDBSCAN is an incremental variant of it, allowing to use the algorithm in online mode. The authors state that it can be proven that IDBSCAN yields the same results as DBSCAN and thus it is only a reformulation. There are two parameters that have to be specified for (I)DBSCAN: a neighborhood radius Eps and the minimum number of objects in a cluster $MinPts$. These parameters again make it difficult to apply IDBSCAN to context computing, but the merging and splitting of clusters, described below in Section 4.6.4 as an extension to GNG, is also an issue for IDBSCAN and it might thus be interesting if some of its concepts could be used. According to [AGGR98], this algorithm is unable to find any clusters in subspaces of the input space (a 5-dimensional cluster in a 10-dimensional input space was tested in the paper). Unfortunately, even if the run-time complexity of IDBSCAN is linear, it seems to need to store previous data points in the cluster definitions, which means that the memory requirements will be too large for embedded systems.

Other interesting algorithms like e.g. BRIDGE, which is merging k-means and DBSCAN [DLX01], BIRCH, which uses a hierarchical data structure [ZRL96], OPTICS, which is an interesting variant of DBSCAN that eliminates the need to estimate its Eps parameter [ABKS99], DBCLASD, which does not require any input parameters at all [XEKS98], CLIQUE, which is able to find clusters in sub spaces and generates minimal cluster descriptions [AGGR98], naive Bayesian classifiers or Support Vector Machines (SVMs) have not been considered because there does not seem to be a reasonable way to use them in an unsupervised or online mode. SVMs are a technique for classification that recently received much attention, as they outperform most other algorithms on a wide range of problems [Bur98]. However, they are supervised two-class classifiers and the "unsupervised" variants are mostly one-class classifiers that can detect abnormal outliers. Extensions to support multi-class classification basically map the problem to multiple two-class classifiers. Additionally, SVMs are computationally intensive and currently not well suited for embedded systems with limited computational resources.

	K-means	FCM	NG	**GNG**	SOM	SNN	ART	IDBSCAN
type	P	P	P	1)	P	P	P	P
approach	DI	DI	DI	**DI**	DI	DI	DI	DE
online	X	X	X	**X**	X	X	X	2)
adaptivity	X			**X**	3)			
variable topology				**X**		4)	X	X
arbitrary shapes		3)		5)				X
soft classification		X	6)	**X**	X	X	X	
noise resistance			X	**X**	X	X		X
limited resources	X	X	X	7)	X	X	7)	
simplicity								
parameters	1	2	2	**6**	4	n/a	1	2
operations	2	2	2	**2**	2	n/a	2	1
interpretability	X	X	X	8)			X	
privacy	X	X	X	**X**	X	X	X	

1) hybrid: aspects of partitioning and hierarchical clustering
2) incremental run-time complexity, but problems with storage capacity
3) variants exist
4) might be possible, depends on exact network model
5) clusters are spherical, but the additional concept of meta clusters allows arbitrary shapes
6) in standard formulation: only ranking of classes
7) due to growing potentially more memory necessary
8) the cluster center can be interpreted in a similar way to k-means, but the additional meta clusters are more difficult

Table 4.1: General overview of algorithms for unsupervised clustering of sensor data

Summary

A comparison of the discussed algorithms and their compliance with the presented requirements is listed in Table 4.1. In addition to the requirements, we also examine the type of an algorithm, i.e. if it is partitioning (P) or hierarchical (H), and its basic approach, i.e. if it is distance (DI) or density (DE) based. The three aspects of simplicity, namely the required number of parameters, the required number of operations and the interpretability of the internal data structures, are listed explicitly for a more detailed presentation. As can be seen, GNG fulfills our stated requirements best. Although it requires a large number of parameters, results have been shown to be largely robust for parameter values within a wide range. Therefore, the parameters can be considered constant for a wide range of application areas and need not be tuned by application developers.
A quantitative comparison of the well-known k-means and SOM algorithms with GNG also showed that GNG compares favorably, owing to its design that fits our given requirements better. This comparison is presented in more detail in Chapter 6. Based on the results of this literature survey, the GNG algorithm has been implemented and used for most experiments within our framework. It might improve the overall classification quality when multiple classifiers are used in parallel and their results are combined at the output of the classification step. A simple method would be to only take the ranking of classes into account when combining the results, to avoid the issue of different distributions of the degree of

membership scores between different classification algorithms [HHS92]. However, this requires the classes to be comparable between the applied classification algorithms, which is typically only the case for supervised classification. We do currently not see a method to apply this combination to classes formed independently by multiple unsupervised classifiers. In the following, this thesis concentrates on GNG and presents it in more detail, along with necessary extensions to apply it in the classification step.

4.6.4 Extended Lifelong Growing Neural Gas

GNG has been introduced by Bernd Fritzke in 1995 [Fri95b] and shares a number of properties with the conceptually similar Dynamic Cell Structure algorithm, which has been developed independently by Jörg Bruske and Gerald Sommer [BS95]. In 1998, Fred Hamker extended GNG to support life-long learning, addressing the Stability-Plasticity Dilemma (the original work was published as [HG98], but extended in [Ham01]). The resulting algorithm has also been called Lifelong Growing Neural Gas (*LLGNG*). The main difference between GNG and LLGNG is that the latter uses local error counters at each node to prevent unbounded insertion of new nodes, thus allowing online learning for arbitrarily long periods. For the historical development of GNG and LLGNG we refer to [MBS93, Fri95b, BS95, Fri97b, HG98, Ham01]. They have been repeatedly shown to outperform classical approaches like k-means, RBF or SOM in some applications, e.g. in Mackey-Glass time series prediction [MBS93], pattern classification and data visualization [Fri96] and on spectrometer data [DWM02]. GNG also showed advantages over standard Multi-Layer Perceptrons in supervised clustering for cancer diagnosis, but had problems with other data sets [HH97]. Although context recognition data sets have not been investigated in previous publications, the wide range of problem areas where GNG has already been applied successfully suggests favorable results also in the area of context computing.
For continuous classification of feature vectors to recognize user or device context, it is important that the applied classification algorithm is self-adaptive. Fred H. Hamker stated quite clearly why this is important for tasks that involve adaptation over arbitrarily long periods [Ham01]:

> *Thresholds such as a maximal number of nodes predetermined by the user as well as an insertion criterion dependent on the overall error or on a quantization error are not appropriate, simply because appropriate figures for these criteria cannot be known in advance.*

Although parameters like the typically annealed learning rate for the SOM can be made self-adaptive, others like the output grid size have to be pre-determined. LLGNG has been designed to cope with long periods of learning.
GNG (and consequently LLGNG) shares its basic structure with many neural networks, including Kohonen SOM [Koh95]. It consists of a number of inputs fully connected to a number of competing output nodes, also called *clusters* (cf. Fig. 4.5). The output units are mutually connected (but not necessarily fully), forming a neighborhood function. When applied to our classification step, inputs are the features extracted from various sensors, which can be highly heterogeneous. The output units then represent the recognized classes in the feature vectors, which are used for assigning labels. Those labels do not necessarily have to be fully connected with the clusters, even a one-to-one mapping might be appropriate if the clusters provide sufficiently high-level information. So far, this applies to the majority of neural networks used for classification, or, more specifically, clustering purposes.
Each output node in GNG/LLGNG consists of an associated position, usually termed *reference* or *prototype* vector, weighted edges to other nodes and local error counters. The basic principle is to insert

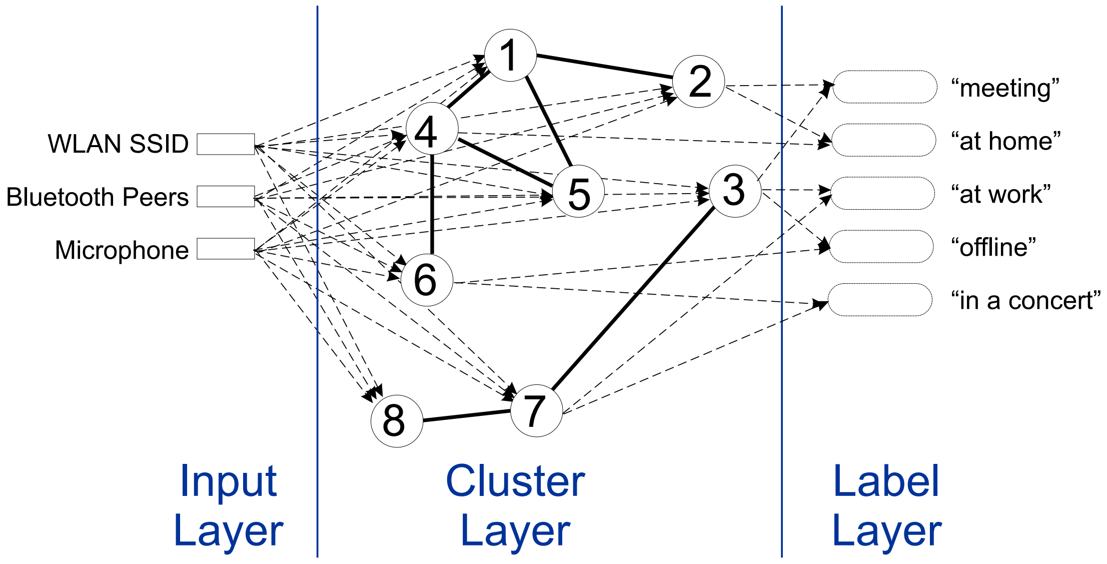


Figure 4.5: Basic structure of GNG and LLGNG

new nodes (clusters) based on local error, i.e. to insert nodes where the input distribution is not well represented by clusters. For classification of features to recognize context, this is an important property because it ensures independence of the usually unknown input distribution; GNG can be seen to combine aspects of hierarchical and non-hierarchical clustering. Additionally, edges are created between the "winner" node, which is closest to the presented sample (in this case a feature vector) and the second nearest one. Other edges of the winner node are aged and removed after some maximum age. The resulting data structure is a cyclic, undirected, weighted graph of nodes and edges, which is continuously updated by competitive Hebbian learning.
In LLGNG, nodes and edges are created and removed and weights are adapted based on local criteria. The following algorithm is summarized from [Ham01], for further details we refer to it. For each new input (feature) vector presented to the algorithm:

1. The node b which is most similar to the input vector and the second best node s are determined by calculating the distance of each node with the defined distance metric.
2. An individual quality measure is calculated for b and all neighbors based on their local error counters.
3. The individual learning rate is calculated for b and all neighbors, which depends on each quality measure computed in the previous step and the age of each node, with younger nodes featuring a higher learning rate.
4. b and all neighbors are moved towards the new input vector according to their respective learning rate.
5. After a certain number of adaptation steps, the node q with the highest value of the insertion criterion is located and a new node is inserted between q and a neighbor. The new node is connected to q and the neighbor with new edges and the edge between q and the neighbor is removed.

6. After a certain number of adaptation steps, the node *d* with the lowest value of the deletion criterion is located and removed, including all of its edges.
7. For the node *b*, the error counters are updated and the age and insertion threshold are decreased.
8. For all edges of *b*, the age (i.e. their weight) is increased. The edge between *b* and *s* is assigned an age of zero or created if it does not exist.
9. All edges that are older than a certain threshold are removed. If a node loses its last edge, the node is removed as well.

In the following, the extensions to LLGNG developed for this work are presented:

Coping with heterogeneous feature vectors

To ease the implementation of heterogeneous features, we reduce the set of heterogeneous features to a minimal subset of abstract features providing meaningful implementations of the necessary two operations *getDistance* and *moveTowards* (cf. [MRF03a]). We ended up with an AbstractString, an AbstractStringList and NumericalContinuous, NumericalDiscrete and Binary features as base classes for the actual implementations. The NumericalContinuous, NumericalDiscrete and Binary features were simple to implement, because they map from a numerical to a numerical space by implementing *getDistance* and *moveTowards* as the Euclidean distance metric and thus need no special considerations. The string based features are more difficult, as they are nominal in type.

Meta clusters

In the standard formulation of GNG and LLGNG, the edges in the internal graph are only used for three purposes:

- For adapting neighbors (adjacent nodes) of the winner: The neighbors of the winner node (those which are adjacent to it) are moved towards the presented sample, with a smaller adaptation rate than the winner node itself.
- For inserting a new node between the winner and its neighbor: A new node is inserted between the winner node and its neighbor with the highest quality measure (which is a local criterion) and immediately connected to both by new edges.
- For removing nodes which have lost all edges due to edge aging: Those units which have lost all edges due to edge aging are removed.

Additionally, the local insertion and removal criteria in LLGNG depend on the neighbors. As can be seen, the edges are only used for the insertion, removal and adaption of nodes, but not for analyzing the graph structure itself.
One of the problems with using standard, unsupervised clustering algorithms for context recognition is that the automatically created clusters are usually too fine-grained for mapping them to high-level context information like “in a meeting” or “at home”; for defining simple context-based rules on the application level, we would like to achieve this granularity. Therefore, we introduce the concept of meta clusters by explicitly using properties of the generated graph structure. To the best of our knowledge, this information has neither been used for GNG nor for the LLGNG variant in previous publications.

The idea of meta clusters is motivated by the fact that GNG/LLGNG preserve the topology of the input distribution when mapping the output space. Thus, structures in the feature space will manifest themselves in the cluster space. As already mentioned, the internal GNG data structure is a graph. The GNG graph, after some learning time, usually consists of multiple components distributed over the cluster space. These components consist of two or more connected nodes and are perfect candidates for high-level context information, because they cover arbitrarily shaped areas in the high-dimensional, heterogeneous cluster space instead of only RBF-type shapes that single clusters cover. This structure within the cluster space actually represents a hierarchical context space as discussed in Section 4.2, but restricted to two layers: clusters can be seen as *context atoms*, while meta clusters can be regarded as a first level of *context aggregates*. Although this automatically constructed hierarchy can — in the present form — only span two layers, it demonstrates that more complex models of context can be used within the classification step. As the class vector interface assumes a flat context model for simplicity, the hierarchy needs to be mapped to a flat vector.
In our extended version of LLGNG, each component is simply assigned a unique meta cluster id and this id is used for the mapping to high-level context information. Fig. 4.5 shows two meta clusters formed by 8 clusters. For performance reasons, the meta cluster ids can not be recalculated after each step, but have to be updated during online learning. When starting with two adjacent nodes in the initialization phase, we simply assign the first meta cluster id to this component and cache the id in each node. During online learning, insertion and removal of edges will lead to the following cases:

- Inserting a new node: Since a new node will only be inserted between two existing ones, its meta cluster id is set to the id of the connected nodes.
- Inserting an edge between two nodes with the same id: No change is necessary.
- Inserting an edge between two nodes with different id: Due to merging two components, one of the ids will be used for the resulting component, overwriting the other one. When both ids have already been assigned to high-level context information, the merge is prevented by not inserting the edge.
- Removing an edge: If the previously adjacent nodes are no longer connected, a meta cluster split has occurred and a new meta cluster id must be allocated for one of the two components.

Normal adaptation of clusters has no influence on the graph structure and can thus be ignored for the handling of meta clusters. Further (performance) optimizations for meta cluster handling in our extension include a caching of meta cluster ids and an incremental check if two nodes are still connected after removing an edge. We also do not perform the complete meta cluster split procedure when one of the components consist of a single node, since this (invalid) node without edges will be removed at a later pruning stage anyways.

Additional performance optimizations

A few additional optimizations have been done, but they are aiming at enhancing the run-time performance and do not introduce new concepts. By storing the nodes in a splay tree instead of a linear list, analyzing neighborhood information before performing expensive operations on single nodes, and caching internal computations we were able to achieve a speedup of over 1.9 compared to the straightforward implementation. These optimizations do not noticeably increase the required memory, but halve the processing time.

4.7 Labeling

Although unobtrusiveness is generally desirable for context-aware applications, in the end the success of context computing will depend on applications that suit the user's needs. As argued in [BBL$^+$00]:

> *There will be no pure context-aware applications, since context-awareness on its own is not something a user needs. Instead many see context-awareness as an enabling technology to help other applications perform better.*

The labeling step has the task to provide applications with high-level, descriptive, user-defined labels for the otherwise automatically detected contexts. Therefore, it is the only step in this architecture that requires user interaction and thus is critical with regard to our general aim of unobtrusive operation. Nonetheless, a completely unsupervised operation of the whole system would not be suitable for practical applications, since the primary aim of context computing is to make interaction with computer systems easier and more intuitive. It is important to involve users in the process of adapting applications to the current context (cf. [Eri02, BD03]) as this can also make applications more trustworthy. A simple yet effective way is to let users name common contexts by assigning descriptive labels to the context classes discovered by the classification step. In the opinion of the author, this constitutes a good compromise between unobtrusiveness and user control, as only those contexts that are important to the user will need to be labeled and the necessary interaction can be implemented in various ways appropriate to the specific device and application. For example, an automatically recorded history of past contexts could be presented to the user upon request to allow labeling with hindsight, or a discreet notice could be shown on the device screen whenever a new, unknown context is encountered to allow an immediate naming. If information appliances without such a common user interface are to be enhanced by context computing, then interactive labeling becomes more difficult. However, the focus of this thesis is not in HCI, so we leave conducting user studies to develop appropriate ways of interaction for the labeling itself for further research. Instead, we assume an assignment of labels to contexts as given and concentrate on techniques for learning the given mapping. On the other hand, user studies on the practicability of labeling at this stage of context recognition might give new insight into the acceptance of context computing in real-world applications and should be covered by future work. The mapping of context classes to labels corresponds to a connection between implicit and explicit context, which has been explained in Section 2.1.2: implicit context is generated as the output of the classification step, but explicit context allows users to exert control over context-aware applications. We think that both aspects of context are necessary for seamless user interaction.

When an output, represented as a string, and an input vector, represented as degrees of memberships for context classes, are assumed to be given for the labeling step, the underlying problem constitutes a supervised classification problem and the task is to learn the classification from a given data set. In this case, the question arises why a second mapping should be used, given that the first mapping, the classification step, could in principle be made supervised and thus directly generate the user-defined class labels. There are two reasons for purposely splitting classification and labeling: Firstly, when only using supervised classification, learning of common patterns in the feature space could not start before the user has assigned the first labels and it would thus be impossible for an information appliance to continuously monitor and learn from its environment. Unsupervised learning in the classification step effectively decouples finding common patterns in sensor values from the necessary user interaction to bridge the gap between implicit and explicit context; only by this decoupling can we support an unobtrusive operation where the times in which labeling takes place are controlled by the user. With a direct

mapping from feature vectors to labels, a user would continuously need to assign labels when new contexts are to be trained. In [TIL04], it has already been documented that users did not adhere to constant naming of the current context in an experiment. Secondly, automatically learned context classes describe common situations and can be, even without user-assigned labels, helpful to applications. Furthermore, the prediction step builds upon the results of the classification step and not on the labeling step, because labeling enforces an information loss. Thus, a devision into two steps makes sense, even if the labeling step is simple.
As already pointed out in [MRF03b], the complexity necessary for this step mostly depends on the quality of the classification step. If classes are long-term stable, i.e. previously learned classed are not overwritten by different new ones, then a simple 1-to-1 mapping of classes to labels is sufficient. However, if the used classification algorithm overwrites older classes in order to learn new contexts, then the degrees of membership of all classes will need to mapped to labels. In [LL01], a simple k-means clustering is used as a second step, i.e. for labeling, after clustering features. For each class, represented by a winner neuron of the used SOM, k-means was applied to the input vectors of the SOM (which correspond to feature vectors in our architecture) to avoid overwriting of labels. This additional complexity is necessary because of the shortcomings of the SOM, and one of the reasons for selecting GNG for our experiments. The same issue of overwriting already learned knowledge has been mentioned in [Ham01]:

> *While a gradual interference is unavoidable, the sudden and complete erasure of previously well learned patterns - a catastrophic interference - severely limits life-long learning.*

According to this, even a second layer of classification, embedded into the labeling step, is expected to produce worse results than a simple mapping of labels on top of a more stable clustering, which has overall less complexity. It is still an open issue if the classification quality will facilitate the use of a simple 1-to-1 mapping or if a more complex n-to-1 mapping from the whole class vector to labels is necessary with our selection of algorithms.

4.8 Prediction

The core part in our architecture is the prediction step, whose aim is to predict future context classes based upon the observed history. As briefly mentioned in Section 4.1.3, our approach to enable a prediction of future context is to interpret the context classes recognized by the classification step as “states” of a state machine. This interpretation is motivated by observations of real-life situations: when considering fairly high-level contexts like “at work”, “in a meeting”, “giving a presentation”, “at home”, “in a car” or “at the beach”, people are in many occasions engaged in only one at a time and advance from one to the next. Of course, this only holds true for high-level context descriptions we aim for. Activities like “writing an email”, “talking” and “sitting” are often done simultaneously. This issue points out the dilemma of representing context at different levels of abstraction; a hierarchical model of context as discussed in Section 4.2 would allow to represent contexts composed of other, most probably lower-level contexts in a recursive way. With the current interface to the prediction step, it is possible for arbitrary detected context classes to be active at the same time, even if no structural information about relationships or dependencies between those contexts can be communicated. The prediction step thus has access to the activation of each automatically detected context at each time step and can construct a model of the interrelationships. However, many prediction methods are univariate and can only cope with a single time series. As we aim to provide high-level future context information to applications, a simplifying assumption is that applications can also benefit when they are only provided with the most likely current and future contexts instead of a complete probability estimation over all contexts. This simplification allows to consider a single trajectory of the respective best matching contexts at each time step and use it as a base for context prediction. A similar state-based interpretation of user actions has been chosen independently in [RC03].

When assuming a state machine as the underlying model, it is possible to monitor the trajectory of active states, i.e. of active context classes. Recording this history and deriving the internal model from the observed trajectory then allows to extrapolate into the future and predict the development of the state trajectory, resulting in a prediction of future context classes.

An entirely different approach to context prediction would be to analyze and predict all sensor or feature time series independently. This would allow to exploit domain-specific knowledge for prediction, like geographical maps for location prediction [PLFK03] or calender information for activity prediction, to complement the domain-independent mathematical prediction models and thus achieve higher accuracy for predicting those time series. For a number of aspects of context and thus for features, stringent constraints can be specified that help to narrow the field of possible future values and help to interpret the results of a prediction on those features. However, this approach would mean to pursue slightly different aims; it would allow to respond to queries about future sensor or feature values, i.e. about future aspects of the whole context, while we aim to predict the whole context. It would certainly be possible to use the predicted and interpreted future values of all features as an input to the classification step and thus also compute a future context class, but there is an issue that severely handicaps this approach. As it will be varyingly difficult to interpret the predictions of different feature time series, the overall prediction accuracies of the different dimensions in the feature space will also diverge. While predictions of some dimensions like location-based features might be interpreted accurately due to well-known methods in the respective areas, predictions of others like audio-based features will most probably be meaningless for the longer term. Therefore, the combination of those predicted feature values in terms of the recognized context class produced by the classification step is also expected to have a low overall predictive accuracy. The reason is that all interrelations between the feature values from different aspects of the whole context are discarded when predicting future values. Feature values are, at least in our assumed area of embedded systems with integrated sensors, clearly *not* independent of each other due to a common underlying cause, namely the situation, or context, that generated the sensor readings. Discarding this statistical dependence accounts for an information loss. Although we expect this approach to be inferior, we have not yet performed any quantitative comparison, which is subject to future research.

As can be seen in Fig. 4.4, the prediction step obtains its input directly from the classification step and provides its output to the labeling step, which can then be used to map predictions to future context labels. Therefore, the input and output interfaces of the prediction step are equivalent, both being a vector of degrees of membership of context classes. This allows the prediction to be unsupervised, like the classification step, and also allows to predict more than a single future "best matching" context, because more information is available than after the labeling step. To be precise, the aim of the prediction step is to generate future class vectors which have the same semantic meaning as the current class vector provided by the classification step.

4.8.1 Requirements

With the same reasoning as in the classification step, a list of requirements on prediction methods for our purpose is defined before a more detailed evaluation is presented. Again, the following list has previously been published in [MRF03b], but is shown in more detail below. The most important requirement is still that the methods should be unsupervised and work without domain-specific knowledge or definition of rules by experts. For our purpose, it is preferable to obtain usable results with no explicit training and user interaction than to obtain a high prediction accuracy with parametrization of each system by experts.

Unsupervised model estimation As mentioned, the internal model topology and the prediction method parameters need to be estimated automatically without human interaction or explicit definition of input/output behavior by experts. For some methods that depend on an internal structure, this includes not only to find suitable parameters, but also to find the model itself, as far as it is specific to the application area. If it is not possible to find a model structure a priori for arbitrary applications and contexts, then it must be learned while the system is in use.
This requirement is valid both for embedded systems, which typically have only a very limited way of user interaction, and for services implemented within an infrastructure. Even if the recomputation of predictive models was handled by infrastructure components instead of the end user devices themselves, e.g. by a GSM service provider for the mobile phones connected to its network, human supervision of the specific model selection processes would not be possible due to scalability issues. Human resources are generally limited, requiring the use of unsupervised methods.

Online learning For information appliances in real-world scenarios, it is infeasible to switch between artificially separated training and prediction phases or even to store enough history for a batch training on extensive data sets. Therefore, the algorithm should continuously adapt its parameters during normal operation, incorporating new class vectors as soon as they arrive. The reasoning for this is the same as for the online learning and adaptivity requirements on classification methods: devices should be operational at all times and explicit training phases impair their functionality. An exception to this is to store only the recent history in detail, which could be used to optimize and/or evaluate the quality of the predictions (by splitting the history in a training and a test set). This is discussed in more detail below in a separate requirement.
For completeness, it should be noted that this requirement could be lifted when not aiming for context prediction embedded into information appliances. If the recomputation of prediction models is deliberately outsourced to services located in the infrastructure or to dedicated training systems like in [Ore02], then batch training becomes feasible. Although this thesis primarily aims at making information appliances capable of predicting context without any infrastructural support because this is the more challenging area, the following survey of prediction methods also includes methods that rely on batch training. On the one hand, some interesting and otherwise suitable prediction methods can not be used in an online manner and, on the other hand, context prediction with the general approach presented in this thesis can also be advantageous for non-autonomous devices. In the detailed survey presented in Sections 4.8.4 and 4.8.5, it will be discussed for the specific prediction methods if they can be used in an online mode and thus for information appliances or not.

Incremental model growth When new classes are detected in the classification step during run-time, new dimensions will be added to class vectors. The prediction algorithm must be able to incrementally increase its internal model topology without requiring a complete retraining. As mentioned, recreating the internal model from scratch will in many cases be impossible, as the complete context history can not be stored in embedded systems because of memory requirements and for reasons of privacy. It is currently unclear if shrinking of class vectors during run-time is also necessary or if "dead" classes that were not activated for a certain time frame could simply receive a minimum probability estimation for becoming active.

Confidence estimation The algorithm should be able to compute an estimation of the correctness of the predicted context along with the forecast itself. This estimation can be used by applications as a confidence measure to determine if the prediction should be relied on for certain actions. In [HKKJ02] the authors also suggest to use confidence estimates for prediction and give the following reason:

> *Uncertainty plays an even more central role in reasoning about* future *states of people; even perfect knowledge about a user's current activities and intentions does not typically extinguish uncertainty about the feature.*

It is therefore important to supply applications with a measure of how certain a computed prediction is, so that applications can treat uncertain predictions differently than more certain ones.

Automatic feedback The prediction engine should continuously estimate the next class vectors and evaluate its estimations by comparing with the real class vectors when they are available. This can be done even if the history is not recorded in detail, because only the next prediction needs to be stored until the time of the prediction to allow a comparison between the predicted and the true contexts. Furthermore, if some part of the (recent) history is stored in detail, this data can be used to retrain the algorithm by splitting the history in a training and test set, either when the user starts some "optimization" tool or in idle phases. The difference to storing the complete history and thus enabling the use of batch-only training methods is that with this method, the necessary storage memory can be limited without impairing the prediction accuracy.

Manual feedback If some action that has been carried out automatically due to a forecast is reverted/canceled by the user, this forecast should receive a penalty to make it less likely the next time. This methodology is known as reinforcement learning in machine learning and is a reasonable way to introduce user feedback unobtrusively.

Long-term vs. short-term The used method should ideally be suitable, e.g. parameterizable, for different forecasting horizons, i.e. predicting context in the near future with high confidence, but also being able to predict later context, most probably with lower confidence.

The developed framework is flexible with regard to the used algorithms, as they only need to implement the input and output interfaces and fulfill the requirements to be usable. Although we have currently not implemented a "meta predictor" as in [CYEOH$^+$03] which combines the results of multiple prediction algorithms running in parallel, this would be a worthwhile task for future work. Unlike the results of the classification step, the output classes for the prediction step are clearly defined. Thus, combining multiple results by merging the ranking of most likely future context classes seems reasonable for multiple prediction algorithms in our approach. In [HHS92], a few computationally simple methods are described that could act as a basis for further research.

4.8.2 Aspects of Prediction

There are a few aspects to time series prediction that need to be considered. In this section, we examine those that are important to context prediction.

Periodic patterns People are creatures of habit; we live with many recurring events, like day and night, weekdays and weekends, lunch time, dinner time, seasons, holidays, birthdays and anniversaries, etc. It is thus one of the most important aspects of time series prediction to find, analyze and model such recurring events, which we call *periodic patterns*. Although many of those patterns will occur with common period lengths like days, weeks, months or years, the prediction method should not necessarily be restricted to locating only patterns in such intervals. There might also be other, nonetheless regular patterns at less common intervals, e.g. taking medication or feeding every few hours.

Sequential patterns The second form of habit is to do certain tasks in an established way, always in the same or a slightly to a situation adapted but still very similar sequence. We call those habits the *sequential patterns*, which are characterized by a list of sub-elements that compose a whole sequence. In [GC03], sequential prediction has been defined as: *given a sequence of events, how do we predict the next event based on a limited known history.* A prediction algorithm should identify patterns in the time series even if they are not completely equal, but similar.

Long-term trends As habits can change over time, the prediction algorithm has to cope with such changes and adapt to the modified habits. When sudden and drastic changes in some habits occur, we do not speak of changed habits but of the formation of new ones. Such new habits should definitely be learned *without* forgetting the old habits, because they may only be temporary, e.g. while attending a conference or while being on holidays. *Long-term trends* are slow, gradual changes that should be followed by the internal model in terms of modification of learned habits. Although we already demand adaptivity from the applied classification algorithm with the aim that long-term trends in the contexts themselves are tracked, the habits consisting of periodic or sequential patterns on top of the context classes can also change. A prediction algorithm developed for online and unsupervised learning is usually also adaptive, because new knowledge is automatically incorporated into the internal data structures, also allowing to track long-term trends. However, not all algorithms automatically take trends into account, so special considerations might me necessary depending on the used method.

4.8.3 Options for Prediction

With current prediction methods, two possible ways for predicting class vectors can be identified: *continuous time series prediction* on individual classes and *categorical time series prediction* on the best matching class trajectory. The former tries to predict the future development of the degrees of membership of each context class, i.e. the probabilities that a certain context will be active at future points in time. This approach inherently produces future class vectors of degrees of membership for each context, because the semantics of the vector elements is not modified. For the latter approach, an integral view is taken by considering only the “best matching”, i.e. the highest ranked context class at each time and analyzing the trajectory of context classes to predict future best matching contexts. This does not impose that only a single output value can be computed for each prediction; it is actually desirable to give an estimation for each possible context, thus implicitly fulfilling the requirement of confidence estimation.

By creating a complete vector of degrees of membership this way, the output interface for the prediction step is satisfied in syntax and semantics. However, this is not an option for some categorical time series prediction algorithms, as they can only create a single value on predictions, based on the past categorical time series. For such algorithms, the single value can be trivially extended to the required output interfaces by setting the respective predicted class to maximum, and all others to zero degree of membership.
In the following two sections, a survey of appropriate methods for continuous and categorical time series prediction is given.

4.8.4 Continuous Time Series Prediction

Continuous time series prediction tries to find a model of a system by observing its output in terms of time series. The prediction problem can be defined as a mapping from a number of past samples of the time series to a future sample value, typically with discrete time steps. For the general case, this should not be confused with a sequential supervised learning problem [Die02], although time series prediction can be expressed by supervised learning by taking both the input vectors, which are assumed to be known for supervised learning, and the output vectors, which are assumed to be unknown, from the same time series but splitting it into past and future. This approach is known as the *sliding window method* [Die02, section 3.1]. In time series analysis, it is generally assumed that the input data includes a systematical component and random noise. This noise complicates the identification of the patterns and it is thus usually desirable to filter it out before analyzing the data. In the following, only a selection of methods is presented that are either simple and well-established or seem appropriate considering our requirements on prediction algorithms.

Statistical tests First of all, there are a few statistical tests that can be applied to time series data to obtain a rough assessment of the underlying system that is generating it. Most of the time series analysis methods only deal with stationary data, i.e. data without apparent trends or seasonality, which are discussed briefly in the next two paragraphs. If a time series turns out to consist of independent and identically distributed (IID) random variables, then no further analysis needs to — or rather can — be done, besides estimating the mean and variance of this random distribution. A few statistical test for checking the hypothesis that a time series is IID have been developed, including the sample autocorrelation function, the portmanteau tests, the turning point test, the difference-sign test, the rank test and others. We refer to [BD02] for a detailed introduction into the topic of time series analysis. For practical time series analysis with limited resources, we recommend the turning point and difference-sign tests as well as a test based on the sample auto correlation function, as they can be implemented very efficiently in an online way with an upper limit on the memory requirements. In the developed software framework, those tests have been implemented for online operation, i.e. it is not necessary to have the complete time series available to perform the statistical tests, but they can be used for continuous estimation during run-time. Only when one or more of the tests for IID fail is it reasonable to perform further analysis and try to find a model of the time series for prediction. In the following, we assume that the observed time series, i.e. the degree of membership of a certain context class, is not IID, i.e. not randomly generated, but follows some systematical pattern that can possibly be discovered and predicted.

Trend analysis In the *classical decomposition* of continuous, time series, data is represented by the model [BD02]:

$$X_t = m_t + s_t + Y_t$$

where m_t is a slowly changing component called the *trend* of a time series and s_t is called *seasonal component* with a known period length d. The remainder of this model Y_t, also called the *residuals*, is assumed to be stationary and can be modeled by one of the known prediction techniques like ARMA. In this classical decomposition, the trend can be estimated with a number of techniques like smoothing with finite moving average filter, exponential smoothing, smoothing by elimination of high-frequency components or polynomial fitting or it can be eliminated by differencing repeatedly. For a detailed discussion of the different techniques, we again refer to [BD02]. For our current experiments, we do not explicitly take trends into account, but have implemented an averaging predictor that can also be used as a trend estimator and for smoothing.

Seasonality analysis The seasonal component in the classical decomposition is defined to fulfill $s_{t+d} = s_t$ and $\sum_{j=1}^{d} s_j = 0$, i.e. to have a known period length and a mean of zero. For estimating the seasonal function, it can be approximated by linear combinations, and it is possible to eliminate the seasonality by differencing with a lag of d. Both options require a known period d, which can e.g. be found by brute force methods. However, it is difficult to estimate the seasonality in an online way without having access to the whole time series at once. Therefore, exact seasonality analysis seems to be restricted to batch training approaches and can not by used in embedded systems. An estimation of the lag, i.e. the period, can be taken from the sample ACF.

ARMA ARMA is probably the best known and most often applied method for stationary, continuous time series prediction. It is a combination of autoregressive (AR) and moving average (MA) models with discrete, equidistant time steps, consisting of linear terms. The univariate output, describing the predicted next sample, depends on the present and the last q inputs of the system and the last p outputs of the system. Before analyzing a time series with an ARMA model, it needs to be stripped from its trend and seasonal components, as ARMA only deals with stationary data. Although it uses solely linear combinations, it has been applied successfully in a wide number of application areas, e.g. for optimizing distributed time warp simulations by predicting event arrival times [Fer99]. The Durbin-Levison and the Innovations algorithms allow to estimate the AR and MA parameters of an ARMA process given a time series [BD02]. After parameter estimation, it can be used to successively predict the next sample value of the time series. A slight variation is the ARIMA model, which explicitly includes a number of differencing steps for making a time series stationary before estimating the AR and MA terms, and consequently needs integration steps for predicting a sample value, hence the letter "I" in the name. Before the parameters can be found, the orders of the AR and MA components need to be estimated, which is assisted by the sample and partial ACF and rarely needs to be larger than two for AR as well as MA, but often depends on expert knowledge and trial-and-error. Thus, ARMA violates our requirements of online learning and incremental model growth, but can still be applied to settings with non-autonomous devices.

Generalized Linear Models (GLM) In [FK96], an interesting generalization of ARMA with online update of the model has been recently presented. The authors presented a recursive estimation method for time series analysis with GLMs based on so-called canonical link models, which have the advantage that the existence and uniqueness of estimators is guaranteed. By recursively computing the regression coefficients of the estimation function, it can be used in an online mode, thus fulfilling our requirement in contrast to the standard ARMA formulation. This is an interesting work an should be examined more closely in future work when linear models are to be implemented as prediction methods within our architecture.

Artificial Neural Network (ANN) Artificial Neural Networks are a family of methods following the principle of combining multiple simple *neurons* into a network structure to realize a desired behavior. An artificial neuron is a simple unit which computes a linear, weighted sum with an additional output function. The most important neural network type is the Multi-Layer Perceptron (MLP) with a strict feed-forward architecture of three layers: the input layer is defined to assume the values of the input vector and does not perform any further computation, the hidden layer is fully connected to the input layer and usually uses the sigmoid function as output function, and the output layer with a number of neurons corresponding to the dimensionality of the output vectors is again fully connected to the hidden layer and applies a linear output function. Such a MLP can be shown to be a universal function approximator [Kol57], i.e. to be able to represent arbitrary, multi-dimensional functions. The aim for a MLP generally is to learn the correct mapping from input to output vectors given a training data set of corresponding inputs and outputs. As the network structure and the involved output functions are typically defined a priori, learning the mapping accounts to finding a combination of weights for the neurons in the hidden and the output layers that realizes the desired function. The well-known back-propagation learning algorithm solves this problem for MLPs. This approach has the advantage that the exact function does not need to be known nor does it need to be expressible in a closed mathematical form, because it can be learned from a given training data set, but it has the disadvantage that the internal behavior of a neural network can not be analyzed analytically, but can only be simulated. For finding a corresponding output vector to a given input, one has to calculate the outputs of the hidden layer neurons and subsequently of the output neurons, which makes reasoning about the weight representation difficult. For a thorough introduction of MLPs and the back-propagation learning algorithm, see e.g. [Zel94]. In an extensive comparison with 16 time series of different complexity, it has been shown that MLPs can outperform ARMA models for time series prediction in many cases [TF93]. For better comparability, the authors have chosen the number of input units for the MLP to be equal to the number of AR terms in the ARMA model for the respective time series, which is an interesting approach to selecting the structure of a MLP and could be used as a basis for future research. An additional advantage is that MLPs can, due to their multi-dimensional output, be easily constructed for multi-step prediction, whereas with ARMA models this can only be done iteratively with consequently limited accuracy. Furthermore, MLPs are, owing to the use of sigmoid functions in the hidden layer, capable of predicting non-linear processes which is not possible with ARMA. The main issue of neural networks and more specifically MLPs is that they are also not suitable for adaptive, online operation and thus not appropriate for embedded systems. Because the network structure needs to be determined a priori, they fail the requirement of incremental model growth and because the back-propagation learning algorithm is a batch algorithm by design, they violate the requirement of online learning. In typical experiments, back-propagation needs hundreds of epochs (an epoch is a presentation of *all* training examples to the network) to achieve good results. Thus, online learning with only one epoch does not seem reasonable for this type of neural network. However, for scenarios where batch training is acceptable, MLPs are a viable option. A method for estimating the confidence of predictions from neural networks has been developed and shown to produce good results on a complex time series [WN94].

Support Vector Machines (SVM) Support Vector Machines are a newer technique for machine learning and are suitable for both pattern recognition and regression estimation, which can be applied to time series prediction. The basic concept of SVMs is that data that is non-separable in its original space can be mapped to another space where it is separable by a linear hyperplane. This hyperplane is chosen so that the distance between the classes that should be separated is maximized. In the final solution of the separation problem, the hyperplane is defined only by those data points that lie closest to it, i.e. those points which are nearest to the other class and are consequently called *support vectors* because changing them would also change the hyperplane and thus the solution of the training. If all other points from the data set were removed or moved within the same class, training would still yield the same hyperplane. For a detailed introduction of SVMs, we refer to the excellent tutorial [Bur98]. In contrast to ANNs, SVMs always find a global solution and are therefore resistant to complex problems with local minima. They also successfully address overfitting, which is a serious problem for almost all methods from the family of ANNs. Although SVMs have started over two decades ago, they currently receive increasing attention because they have been shown to yield high accuracy and outperform competing methods for many problems, including time series prediction [MSR$^+$97, MOG97], in which we are currently interested. As in many machine learning techniques, most of the practical problems can not be solved analytically but need to be dealt with by numerical optimization. SVM training generally depends on constrained quadratic optimization and, with current algorithms, SVMs bring forth extreme demands for processing power and are thus difficult to use for embedded systems. For our purpose of context prediction, SVMs can in principle be used in the same way as ANNs for continuous time series prediction, as there are variants for regression. But there are other variants that could be used to predict the next best matching context class when given the current and possibly past class vectors with degrees of membership, which corresponds to categorical time series prediction. Very recently, a method for online training of SVMs has been presented [AC03] as well as online time series prediction with SVMs [MTP03], but these recent developments have so far not been evaluated with regard to usability within our architecture. They are inherently unsupervised, but incremental model growth seems difficult even in the online variant.

4.8.5 Categorical Time Series Prediction

The approach of using continuous time series prediction independently on each of the context class degrees of membership has a significant disadvantage that is not unlike the reason why we chose not to predict sensor or feature values directly. Although some high-level contexts might be statistically independent, most are presumed to be dependent. An example would be the two mutually exclusive contexts “giving a presentation” and “in a concert”. The knowledge — or strictly speaking the estimation with a high confidence — that one of the contexts is active is a strong indication that the other one is not. As already mentioned in the introduction to the prediction step, we expect people to be engaged in only one high-level context during most of the time, assuming a strong mutual dependency between all context classes. This favors the second option for prediction, namely categorical time series prediction, where the trajectory of the best matching context classes is considered. Although an integral model over the degrees of membership of all context classes would not discard any available information at this stage and at the same time consider the statistical dependence, this would necessitate a multi-dimensional time series prediction. This can be performed with universal approximators like ANNs. Universal models like ANNs could be the focus of future research by concatenating past values of multiple time series into a single input vector, thus mapping the multi-dimensional problem to a single-dimensional one. In the following, we examine single-dimensional categorical time series that only take the best matching

context for each time step into account.
Many of the categorical time series prediction methods assume the Markov property, which supposes independence between time steps. A first order Markov model assumes that the next state depends solely on the current state, and consequently any relationship between separated states must be communicated via intermediary states [Die02]. For context classes, this assumption is invalid for the general case. E.g. when being in the context “walking in the corridor”, the probability that the next context will be “at work” should be different when the previous context was “at home” from the case when the previous was “at work”. In the first case, the user is probably just coming to the office in the morning, while in the second case he might be leaving his office for lunch and it is thus less likely that he will immediately return. However, models based on the Markov assumption have been used successfully for many application areas where the assumption is also invalid, so they should not be excluded from our survey. The list of models included in this survey is also not exhaustive, but should include the most common ones as well as those that have been deemed well suited for this purpose.

Central tendency predictor The most simple prediction method is to predict the central tendency, i.e. the average, value considering a window of the n last values of the time series. A general average can be found for any type of time series and is often chosen to be the arithmetic or geometric mean for numerical and the median for ordinal series. For categorical time series of best matching context classes, which are of nominal type with no order relation, the average over a time window can be defined to be the value that has been seen most often, i.e. the peak in the histogram. Central tendency predictions can be computed online in an unsupervised way and trivially support model growth. Furthermore, they can be computed incrementally with constant run-time complexity and consume minimal memory. Thus, they fulfill all of our core requirements, but mark the lower end of computational complexity. Obviously, central tendency predictors can neither deal with periodic nor with sequential patterns, but are restricted to smoothing the time series.

Finite state machine Another simple method is to apply finite state machines that can be used to realize state predictors as described in [PBTU03b] for location prediction. Such state predictors are essentially specialized central tendency predictors that remember only one state state to predict, but have a certain “inertia” for switching states. E.g. a 2-state predictor only switches its prediction to a new state if it has been observed at least twice successively and remains in its current state otherwise. As the general central tendency predictors, arbitrary n-state predictors are unsupervised, online, cope trivially with model growth and can be implemented very efficiently.

First order Markov model After the central tendency predictor, the next model with regard to complexity is the first order Markov model. For each context, the frequency of each successor, often called transition probability, is recorded and the successor with the highest frequency is predicted for the next step. This has been proposed in [LC00] for controlling the context recognition process. When a transition from the last recognized context to the current one was regarded as unlikely by the first order Markov chain model, the current recognition was supposed to be discarded. In the same way, such a Markov model could be used to predict the most probable context for the next time step. First order Markov models are more complex than central tendency predictors, but still fulfill all core requirements including model growth; the transition probabilities for a newly created context class can be safely initialized with zero until transitions into this new context occur. The run-time complexity for online training and prediction is still low enough to consume negligible processing power compared to classification.

Month	Day of month	Hour	Minute	Frequency
7	10	23	15	0.001
7	11	9	00	0.001
7	11	15	30	0.002
...	...	...	...	

Table 4.2: Example for a time-dependent transition probability table

Higher order Markov model When the underlying process suggests a dependency between pairs of states that are not directly adjacent in the time series, then the first order Markov models can be extended to not only depend on the current state for predicting the next one, but to take a number of past states into account. This number of states defines the order of the higher order Markov models. Such models are, in contrast to first order Markov models, capable of exploiting relationships between temporally distant states. Subject to their order that defines the maximum distance between states for which a relationship can be recognized, higher order Markov models can in principle account for sequential as well as (relative) periodic patterns in the time series. Active LeZi is a specific algorithm that constructs such models.

Active Lempel-Ziv (ALZ) The Active LeZi prediction algorithm has been developed for the MavHome project and uses information theoretical principles to construct higher order Markov chains [GC03]. Based on the Lempel-Ziv LZ78 compression algorithm, is has been designed as an online and unsupervised algorithm. Incremental model growth is also possible, as the internal data structures are dynamically updated when new values in the time series are observed. Therefore, ALZ fulfills our core requirements, but is slightly more expensive than simple first order Markov chains both in run-time and memory complexity. By extracting phrases from the time series and incorporating concepts from Prediction by Partial Match (PPM) algorithms, ALZ constructs weighted Markov models of different orders and combines them into a common probability distribution.

Full model All models for categorical time series prediction that have been examined so far only realize sequential prediction, but no *absolute periodic patterns* are explicitly taken into account. In [HKKJ02] it is also argued that taking the time of the day, in this paper in domain-specific extracts like weekdays, weekends, morning, lunchtime, afternoon, evening and night or the weekday into account is valuable for prediction. By adding a dependency on certain aspects of the system time to the transition probabilities, a full statistical model can be obtained. For first order Markov models, instead of only recording a single transition probability from each observed context to every successor, a time-dependent probability distribution could be recorded. Table 4.2 shows an exemplary probability distribution for the transition from a high-level context “A” to a high-level context “B”: On July 10 at 23:15, a transition from A to B occurred as well as on July 11, 9:00. On July 11 at 15:30, the frequency is higher because a transition from A to B occurred at the same date and time a year before. Such a complete model allows to aggregate frequencies for different periods, e.g. hourly, daily, weekly or monthly and thus compute different transition histograms based on different periods. The context cube [HLA$^+$04] could provide an effective means of storing the context history along with the history of raw features and/or sensor values. However, it is currently aimed towards large context repositories and an implementation with upper memory bounds seems difficult. An aggregated daily histogram, i.e.

by only considering the hour attribute, of transitions from A to B might show peaks at 9:00 and 15:30 and thus indicate a high probability for predictions within these time frames. For higher order Markov models like the ones constructed by Active LeZi, transition tables need to be recorded for each phrase of context classes, at each level of the model. A model calculation shows the amount of memory necessary for recording such a complete probability distribution: We assume that 30 distinct context classes have been recognized by the classification step and that only a first order Markov model is used. For each possible transition between any two contexts, i.e. 30×29, a table with the attributes month, day, hour and minute with a granularity of 5 minutes will be stored and the frequency will be expressed as a float value with 4 Bytes. Then, for each table $12 \times 31 \times 24 \times 12 \times 4 = 428544$ Bytes will be consumed, amounting to an overall memory usage of roughly 355.56 MBytes. This is a worst case model, because for certain pairs of contexts the transition probability tables will be sparse and the overall memory is consequently expected to be significantly lower for practical applications. On the other hand, this model calculation only considered a first order Markov model, which does not seem appropriate for more complex sequential prediction. By considering higher order Markov models, the number of transition tables grows exponentially with the model order. Although the full model will attain the highest prediction accuracy that can be achieved with the observed history of the time series, it is not practically applicable for embedded systems with limited memory.

Hidden Markov Model (HMM) Hidden Markov Models are to be considered *the* standard model for time series classification in many areas, e.g. speech recognition, handwritten character recognition, gesture recognition or segmentation of DNA sequences. In context computing, they have been applied at various levels, like recognizing location based on audio and video [CMP00], task prediction [RC03] or action recognition [LBVW03]. So far we have only considered "exact" models that record, i.e. count, occurrences of observed transitions and calculate probabilities from those transition frequencies. HMMs are a more powerful model, but do not necessarily produce the same model when trained with identical training data. The reason for this issue is that during the estimation of the model parameters, the typically used method, called forward-backward algorithm, leads to local maxima for this optimization problem. Many practical problems have been found not to be solvable by simple Markov chain models, where the observations are mapped to states. HMMs thus add an additional hidden layer, hence the name, which contains a Markov chain model that indirectly influences the HMM output via a separate probability distribution. To detail, a complete HMM consists of a set of states, a state transition probability distribution, an observation symbol probability distribution and an initial state distribution. When training a HMM model from a data set, the number of states and the connections between the states need to be determined a priori and all three probability distributions are adjusted by training on the data. For a detailed introduction into HMMs and the associated training procedures, we refer to the standard tutorial [Rab89]. HMMs have been applied to many problems in time series analysis because there exist mature methods for parameter estimation and simulation, building on a solid theoretical foundation. However, as already mentioned, the parameter estimation is an optimization problem for which no analytical solution is currently known. The training method is an iterative batch approach and we are currently not aware of any online method that would allow to use HMMs in an online way in our architecture. Incremental model growth is also not supported, as the number of hidden states and the alphabet of observation symbols need to be defined before training and can not be modified afterwards. The major requirement of unsupervised operation is violated by the need to determine the number of hidden states a priori. These issues restrict the HMM method also to scenarios where an autonomous operation of embedded systems without infrastructure support for recomputation is not strictly necessary. Various extensions of the basic HMM have been proposed to address state durations [LT99] or to deal with the representational capacity of HMMs by adding structure to the hidden states. Especially the Factorial HMMs (FHMMs) have been shown to outperform standard HMMs on time series with complex structures [GJ97] and could be beneficial for our purpose, while Coupled HMMs (CHMMs) [BOP96] might be less suitable because unsupervised model estimation does not seem possible. The issue of selecting appropriate numbers of hidden states has been tackled in [LB00], thus giving rise to unsupervised operation. By applying the Bayesian Information Criterion (BIC) [Hec95], HMMs become a viable method for batch training approaches, but still seem unsuitable for context prediction embedded in information appliances.

Bayesian Networks (BN) Bayesian (Belief) Networks are a superset of HMMs, Kalman filtering and other probabilistic models [Mur02]. A BN is in general a directed acyclic graph (DAG) of nodes, which represent the random variables, and edges, which represent the direct influence of one variable on another. Within this structure, each variable is independent of its non-descendants given its parents, and the network can thus be completely parameterized by specifying local models for each variable. In the case of discrete variables with a finite set of values, a standard representation of these local models are tables with conditional distributions. Due to the graphical representation of probabilistic dependence, BNs have been widely successful for problems where prior knowledge exists and can be incorporated into the model. Additionally, the dependence information in BNs has a causal semantics and is thus often intuitive for some applications [Hec95]. There exist various approaches for learning the parameters of a BN from data, including the possibility to learn the model structure even from incomplete data sets with missing values [Fri98] and to perform an online update of the structure [FG97], which allows BNs to fulfill the requirement of online learning. However, the number and the structure of hidden variables used for describing influences on the observed variables also have to be specified a priori, which leads to the same issues in unsupervised operation as present with HMMs. Dynamic Bayes Nets (DBNs) [MM99] introduce time dependencies into the model and can be used to analyze time series. By discretizing the state variables of a DBN, one obtains a HMM model with the standard properties. Consequently, the forward-backward algorithm used to train HMMs can be seen as a generalization of the belief propagation from BNs. Other types of DBNs deal with continuous variable distributions, which is not necessary for categorical time series prediction and thus not considered within the scope of this survey. A method for performing online learning of DBNs has been described in [Mur02, Chapter 6], including the possibility to learn the structure of the network, thus fulfilling the requirement of unsupervised operation. For the problem of context prediction, the more general class of DBNs does not offer any immediate benefit over its special case of HMMs. As formulated in [RN95]:

> *The difference is that, by decomposing the state of a complex system into its constituent variables, the DBN is able to take advantage of sparseness in the temporal probability model.*

For the categorical time series prediction approach to context prediction, the assumption that the system advances from state to state implicates that there is only one discrete random variable. Therefore, the special case of HMMs is sufficient for the assumptions stated earlier and is used as the representative for DBNs in the remainder of this thesis. A more practical advantage of HMMs is that, being the older model, there exist more mature software tools and libraries that can be readily used for context prediction than there are for general DBNs.

transition to	1	2	3	4	5	6	7	8	9	10	11	12	13
B	0	0	1	1	0	1	2	2	1	0	0	0	0
C	1	0	0	0	0	0	0	1	1	2	1	2	1

Table 4.3: Example for a probability distribution table for duration prediction

Episode Discovery (ED) In [CYEOH+03], Episode Discovery has been presented briefly as a data mining algorithm that is capable of finding significant episodes in a time series, where a significant episode was introduced as a related set of events that occurs at regular intervals. It is based heavily on [SA96], but has been modified to cope with unordered in addition to ordered sets of events. In a sliding window, ED searches for episodes using the minimum description length principle and describes episodes with a so-called regularity factor, which can be daily, weekly or monthly, to achieve shorter episode descriptions. An online variant has been developed and tested on artificial test data [DCB+02] and, based from the description, it seems to be unsupervised and support model growth, but failed to discover periodic patterns that were occurring multiple times a week [HC03]. Although the algorithm is online and unsupervised, the description does not suffice to re-implement it and examine if incremental model growth would be possible.

Duration predictor A new categorical time series prediction algorithm is currently developed within the scope of our work for future experiments on context prediction. The basic idea is to build on the durations in which the context classes are active and record not only the single transition probabilities between states, but the complete probability distributions describing how long a specific context has been active. Table 4.3 shows such an exemplary transition table for a context "A" with the recorded transitions to contexts "B" and "C". As can be seen, a transition from A to B is more likely when A has been active for around 7 time steps, while a transition to C is more likely when it has been active for longer than 10 time steps. This additional information helps to decide on ambiguities, when the aggregated transition probabilities to different contexts are similar, i.e. when context B has been observed after context A roughly the same number of times than C has been observed after A. As this discrete probability distribution can not be used directly for each time step, a smoothing filter is applied before searching for peaks in the histogram. The difference to recording the full model as described above is that only relative times instead of absolute ones are stored, saving memory but also allowing to take time into account, at least for sequential prediction. By exploiting the probability distributions, it should be possible to predict the number of time steps that will most probably be spent in the current context and thus to predict the time of the next context transition. Prediction of absolute periodic patterns, i.e. patterns interrelated with fixed time periods like days or weeks, is not directly possible with this technique, as there are still only durations relative between adjacent states available. This method for categorical time series prediction is optimized to context prediction, where longer sub-sequences of equivalent contexts are observed in the given categorical time series. It is currently work in progress, but preliminary evaluation results are presented in Chapter 6.

4.8.6 Summary

For reasons described in the introduction of the prediction step, we currently favor categorical time series prediction for its integral view on the problem over continuous time series prediction on the individual context classes. As also stated in [MRF04b], we have currently not selected a specific algorithm for the prediction step because our architecture is open for arbitrary algorithms that can be adapted to suit our interface. Based on the presented literature survey, which accounts to a qualitative comparison of possible candidates based on their features, we found the simple Markov chain models more suitable than HMMs for online, unsupervised usage in embedded systems, mostly due to the supervised and batch training method of HMMs. ALZ is a specific algorithm for constructing multiple Markov chains of different order and unifying them in a combined model and seems to be suited well. A quantitative comparison of ALZ with naive central tendency predictors and HMMs is presented in Section 6.4. Experiments with continuous time series prediction comparing ARMA and ANNs are also shown in this evaluation. For continuous time series, the online variant of regression with SVMs, also called Support Vector Regression (SVR), is a very promising candidate and should be evaluated in even more detail in future work. Various types of BNs might also be suitable, as there exist methods for learning not only the model parameters as is possible with HMMs, but also the model structure. This would allow for unsupervised learning but also needs further investigation. At the moment, the aspect of periodic pattern prediction is missing from our analysis because of limited support by prediction methods for tackling this issue in an online manner. Even the Episode Discovery algorithm, whose presentation by the authors suggests that it can locate periodic patterns, actually locates sequential patterns. It was capable of finding patterns with a common distance like daily, weekly or monthly, but is reported to have failed on patterns that occurred multiple times a week. An example of a pattern that should be detected and predicted by a suitable algorithm could be "each Monday at 13:15". The well-known algorithms that are capable of detecting arbitrary periodic patterns are aimed at databases where the full time series is available. In such a setting, multi-pass approaches are feasible, but in the field of context computing, the involved systems typically deal with data streams and therefore require a one-pass approach. An algorithm that is claimed to work with a single pass over the data set has very recently been proposed [EAE04]. Its authors state that it is the first published algorithm which can detect periodic patterns with an unknown period length with a time complexity of $O(n \log n)$ when n denotes the length of the time series. With this time complexity, it is currently unclear if a single pass is indeed sufficient, particularly because the first phase of the algorithm comprises an expansion of the categorical input to a binary time series. For this expansion, the set of symbols from which the categorical time series is constructed needs to be known in advance, which can not be guaranteed for an online mode of operation. Periodic pattern prediction is currently not solved for context computing with the requirements defined above, but is an important area for improving the accuracy of context prediction. The newly published algorithm could offer some possibilities for tackling this problem and lays ground for future research in this direction.

It might be necessary to use multiple different prediction methods concurrently and merge their results to generate a reasonable forecast of the context trajectory. However, combining results of multiple predictors requires at least a ranking of predicted context classes by each of the methods and can benefit from a more detailed confidence estimation. At the moment, the possible improvement in prediction accuracy by combining multiple methods is unknown. A detailed investigation of this issue should be subject to future research.

Chapter 5

Framework

A software framework has been developed to perform practical experiments with context prediction. Although the primary motivation for the development was a proof-of-concept implementation of the presented architecture for context recognition, it has been developed with flexibility in mind and can now be used to support arbitrary applications. The internal structure and details on the framework are presented in an accompanying diploma thesis [Rad04].

5.1 Concept

Figure 5.1 shows an overview of the implementation, which closely resembles the architecture presented in the last chapter and shown in Fig. 4.4. Reflecting the need to work with sensors, the framework needs direct access to the specific hardware and is therefore situated between the hardware (but on top of the operating system) and applications that utilize context information. To achieve a high level of flexibility both during development and for application deployment, the core framework is kept as small as possible and is strongly based on a modularized concept. It basically only manages the information flows between

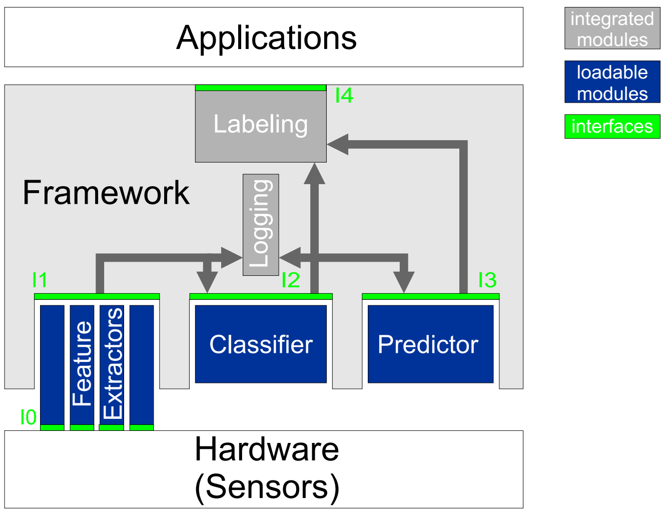


Figure 5.1: Conceptual overview of the framework

the defined steps and provides interfaces for modules to implement these steps. The modules then realize the functionality of the system, but are independent of other modules and thus easily exchangeable.
Three different hot-spots have been defined for adding modules: *feature extractors*, *classifiers* and *predictors*. All of these module types are realized as dynamically loadable libraries on those platforms that support them, and a common framework configuration specifies which modules should be loaded upon startup, i.e. during run-time. It is easily possible to use different features, classification or prediction algorithms by only modifying this configuration and without changing any code. In Fig. 5.1, the respective interfaces that need to be implemented by the loadable modules are `I1` for feature extractors, `I2` for classifiers and `I3` for predictors. The labeling step has currently not been realized as a module, but is a part of the core framework. Interface `I4` is therefore not designed for implementation by additional modules, but is used directly by applications to query high-level context information. Additionally, the framework provides a common facility for logging and replaying of previously recorded logs at two different interfaces in the architecture, namely `I1` and `I2`. This component allows for decentralized logging of feature streams and identified contexts locally at each device, but centralized context data repositories are not being considered. If a combined log of all feature streams and identified contexts of different devices is desired, the individual local logs need to be transferred and merged by other means. The integrated replaying component has been designed for a complete reproduction of the recorded feature streams so that new classification methods can be evaluated offline on arbitrary data sets. It does not provide random access to feature logs or past recognized contexts that could be utilized by classification or prediction methods. Although a structured way of logging to and querying a central context data repository could be added to the currently available logging and replaying component, this was not one of the design goals for the framework. Its current aim is to support context recognition and prediction locally at each device, and support for coordination of multiple networked devices is an issue for future research.
In the following, the framework components are described briefly (for a detailed presentation see [Rad04]).

5.2 External Interfaces

As expected, the framework has two external interfaces that it uses to communicate with other systems. In Fig. 5.1, they are denoted as `I0` and `I4`. At the input side of the framework, `I0` connects directly to the available sensors, while at the output side, `I4` provides applications with recognized current and predicted future context.

5.2.1 Sensors

The wide range of sensors that is currently available dictates a similarly large range of different interfaces for software parts to communicate with those sensing technologies. Although highly desirable, it is consequently hard to define common interfaces for accessing raw sensor data. Sensor data can be either event- or stream-based and different sensors require significantly different sampling rates and yield heterogeneous values of various atomic and non-atomic data types. This diversity makes it very challenging to define a common interface that (a) is flexible enough to deal with all peculiarities of arbitrary sensors, that (b) is easy to use and efficient in terms of the overhead for accessing sensor data, and that (c) allows structured access to the sensor data on a semantic instead of a purely syntactic level. It is obviously possible to define interfaces that fulfill (a) and (b), e.g. TCP streams that simply forward all sensor data in the native format of the respective sensor to a central location. However, such an interface does not provide a standard, structured method of accessing this information at the receiver side other than implementing an interpreter for each sensor stream independently. It is also easy to define interfaces that fulfill (b) and (c) by limiting the range of possible sensor values to either events or streams and defining a common sampling rate over all sensors. Interfaces that fulfill (a) and (c) can too be defined, but it is unlikely that they would be easy to use on both sides of the interface: for the wrapper modules that provide access to some sensor via the common interface and for the framework that accesses the sensor data.

Therefore, this framework takes a pragmatic approach to interfacing with specific sensors by not defining a common interface at this level. Instead, feature extractor modules access the respective sensors independently and implement a common interface after the feature extraction step, denoted as `I1`. As discussed in Section 4.5, the selection of features is highly domain-specific and correlated with the set of available sensors, which too suggests a combined implementation of sensor data acquisition and feature extractor modules. In contrast, the Context Toolkit uses a clear separation between sensors (wrapped by "widgets" in the Context Toolkit), a combination of sensors (called "aggregators") and components that extract higher-level information from sensor data (called "interpreters"). This is clearly a more flexible approach and can be easily distributed over multiple devices because the communication between the components is based on HTTP. For the framework presented in this chapter, a more lightweight approach is used that does not depend on a network protocol internally and causes practically no overhead in transferring data between its components. Because context widgets are roughly comparable to our notion of feature extractors, an integration of both approaches is feasible. For automatic context recognition and prediction as proposed in the present thesis but within a networked environment instead of locally on individual devices, the Context Toolkit could be used to provide feature values for further processing within this framework. Such a combination of both frameworks would support highly flexible but still unobtrusive context-aware applications and is an issue for future research.

The external interface `I0` at the input side of the framework is thus, in the current implementation, not a common interface, but describes an arbitrary number of different, sensor-specific interfaces.

5.2.2 Applications

For applications, the framework provides context identifiers either in the internally constructed numeric format or in terms of context labels defined by the user. Currently, only a pull mechanism is implemented whereby arbitrary applications can poll for current and future contexts. This can either be done via a direct method call when the framework is linked to the application or via a standard protocol for a loose coupling. This protocol is also used for connecting user interface applications to the labeling step and is described in more detail below in Section 5.4.4. When necessary, a push mechanism can be trivially implemented within the current framework, which would allow applications to subscribe to specific current or future contexts and receive appropriate notifications.

The external interface `I4` at the output side of the framework is thus a method call interface accompanied by a standard protocol for loose coupling of the labeling step.

5.3 Platforms

Portability was one of the primary aims during development. Therefore, the framework core is completely platform independent and should compile and run on any system with an ANSI C++ compatible compiler. There are some small support components that need to be adapted to specific operating system APIs, mostly multi-threading support for sampling sensors at different sample frequencies and dynamic loading of libraries for handling modules at run-time. These platform dependent parts have, at the time of this writing, been implemented for:

Win32 This includes all Microsoft Windows operating systems starting with Windows NT 3.1, but support for various sensors demands newer hardware APIs that are available since Windows XP.

WinCE Windows CE provides a subset of the Win32 API and is typically used on PDAs and smaller tablet PCs or handheld PCs. Newer smart phones also use a variant of Windows CE as operating system.

POSIX Multi-threading and dynamic loading of libraries have also been implemented for the POSIX standard, but have only been tested on Linux with i386 and ARM processor platforms.

Symbian OS Many current smart phones are based on the Symbian operating system, to which the framework also has been ported.

It should be noted that the platform dependent components mentioned above are small and can usually be implemented with minor effort for other platforms. However, the interfaces that allow access to sensors, i.e. to the hardware layer, are in practice significantly different and require adaptations in the feature extractor modules.

5.4 Architecture

All major steps of our architecture, explained in detail in the previous chapter, have been implemented within the framework, directly mapping the formally defined interfaces to interfaces in the OOP sense. The sensor data acquisition step is not a separate part of the framework because it is closely coupled with the feature extraction step and the discussed problem of external interfaces arises. Due to this close dependency and because, from a software point of view, sensor data acquisition is usually limited to using some hardware-specific API, the framework integrates it with feature extraction into a single part. This does not prohibit to create feature extractors that rely on external sources of sensor data like context widgets implemented by the Context Toolkit.
Of the four main parts of the framework, three are implemented as loadable modules and one is more loosely coupled. This allows the framework to be extended in all four parts by either creating new modules that can be loaded into the respective hot-spots or writing applications that can be coupled with the framework. It would be feasible to define a loose coupling of all components via standard protocols, and thus providing the possibility of distributed operation where different parts of the framework are executed on different but interconnected devices. A subscription-based event or streaming interface could allow classifiers to subscribe to feature streams and predictors to subscribe to recognized contexts provided by classifiers. Applications would be free to subscribe to events or streams at all levels. The necessary additions to the current framework are moderate and could be implemented solely within the framework core and without modifying the currently defined modules. However, the main issue with such an enhancement is to still allow the framework to be executed locally on resource limited devices. This implies to make the loose coupling optional and retain the current method call interfaces with their advantage of marginal overhead for those cases where distributed execution is not necessary. A worthwhile compromise would be an integration with the Context Toolkit as mentioned above.

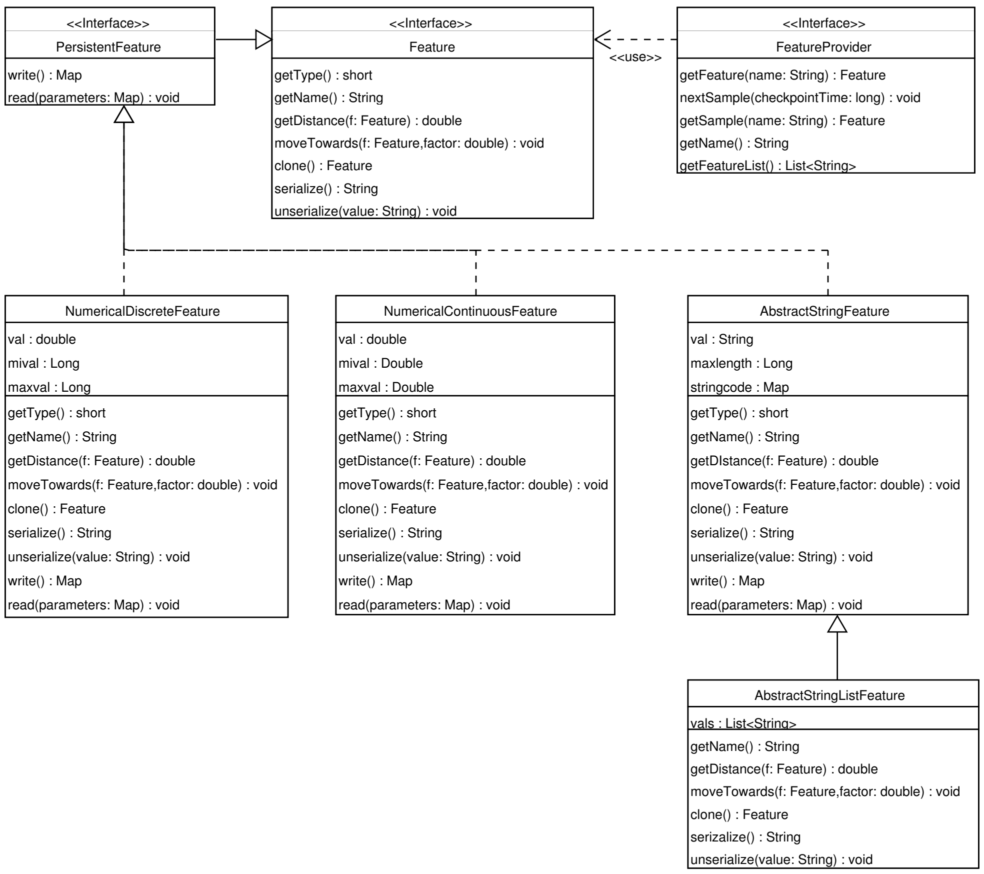


Figure 5.2: Interface for feature extractor modules

5.4.1 Feature Extractors

Feature extractors are our chosen abstraction of sensors and offer a more general and higher-level access to sensing technology. A feature extractor provides a number of heterogeneous features as defined in Section 4.5 and is usually responsible for handling a specific sensor type. Fig. 5.2 shows the basic interfaces intended for use by feature extractor modules and situated at `I1` in Fig. 5.1. They are central to the framework and are responsible for its flexibility in dealing with heterogeneous feature values. The operating system or vendor-specific APIs used by feature extractors for accessing hardware sensors are depicted as `I0` in the framework overview. No assumptions about the communication with sensors are being made. Consequently, it is the responsibility of a feature extractor module to ensure that it can deal with the data rate of its associated sensor and that it fulfills the domain-specific quality criteria like latency or computational accuracy. It is not necessary for feature extractors to manage extensive data buffers, because the framework periodically polls current feature values from the providers. Typically, only a single buffer for the most recent feature value needs to be maintained.
`Feature` is the connecting interface between feature extractors and the input part of classifiers. Its main methods are `getDistance` and `moveTowards`, which are the implementations of the two methods necessary to deal with an heterogeneous feature space and described in more detail in Section 4.5.2. In

addition to these primary methods, a few additional ones are necessary for managing feature extractor modules: `getType` and `getName` return the type (boolean, nominal, ordinal, numerical discrete or numerical continuous) and a unique name, respectively, `clone` creates a complete, if necessary deep, copy of the object and is used by classifiers, and `serialize` and `unserialize` allow to save and restore feature objects in a generic way. This generic serialization allows an implementation of a common logging and replaying part that works with arbitrary feature extractors and is described in more detail below in Section 5.5. `PersistentFeature` is derived from `Feature` and extends it by the methods `write` and `read`, which allow objects to save and restore static data, i.e. data that is not contained in the feature objects but is global to all feature objects of the same type. Examples for static data are minimum and maximum values that need to be persistent over restarts of the framework. A pool of persistent data is managed automatically by the framework in a XML file, whose schema is presented in [Rad04], for all registered feature extractors. For representing a whole sensor with possibly multiple features, `FeatureProvider` defines three primary methods: `getFeature` and `getSample` each return feature objects by their unique name with randomized values and values representing the current sensor readings, respectively. `getFeature` is used for the initialization of classification algorithms which depend on random input vector prototypes, and the random initialization is obviously dependent on the internal data of the specific feature implementations. Although `getSample` returns feature objects that represent the sensor readings, it is defined to return a snapshot of the time that has been passed to the last invocation of `nextSample`. The reason for this separation is that different sensors usually require different sampling intervals, and `nextSample` is therefore used as a means of synchronization over all feature extractor modules.
These basic interfaces are sufficient to work with arbitrary, heterogeneous features. However, a few common types of features occur in many cases, motivating the implementation of the defined methods for the standard types. `NumericalDiscreteFeature`, `NumericalContinuousFeature`, `AbstractStringFeature` and `AbstractStringListFeature` implement all necessary methods for features that can be represented as a single integer value, a single real value, a single string, or a list of strings, respectively. To use these types of features within feature extractor modules, it is sufficient to override the method `getName`, returning a unique name of the implemented feature, and to provide appropriate static variables, which need to be specific to each feature and can not be shared among all features of the same type.

Available Modules

In the current framework, a number of feature extractor modules have been implemented for the most common off-the-shelf sensors available in mobile devices. Only a few of them could be realized in a platform independent way, like the virtual time feature extractor, and most need at least some platform dependent code for accessing the operating system APIs to the respective hardware interfaces. Currently, the following feature extractors are available:

Active window An important aspect of context is the activity of the user, and when interacting with current computer systems, the application that is currently in use strongly suggests certain activities. This feature extractor only provides one feature: the name of the currently active application or application window, depending on the platform. It has been ported to Win32, WinCE and POSIX/X11.
Because the name of the currently active application or application window is a simple string, this feature is derived from `AbstractStringFeature`.

Audio In addition to activity and location, environmental sounds can also give an indication on the social aspect of context, e.g. if a user is talking or laughing. Typically from a microphone, this module samples an audio stream and extracts the average loudness, the number of peaks and a configurable number of FFT coefficients within a time window. By using a cross-platform library, it runs under Win32, WinCE and Linux.
This module provides multiple features: the average loudness is derived from `NumericalContinuousFeature`, the number of peaks is derived from `NumericalDiscreteFeature` and all FFT coefficients are also derived from `NumericalContinuousFeature`.

Bluetooth The social aspect of context is also partially described by the set of people that are in spatial proximity, and location in terms of cell-based location information can be trivially inferred when devices with a known location are in communication range. If a Bluetooth adapter is available in a device, it can be used to inquire for other devices in range. This information is provided in two different features, as a list of MAC addresses, and aggregated as the number of devices in range. We currently have implementations for Linux Bluez, Widcomm and Digianswer Bluetooth APIs. The list of addresses is derived from `AbstractStringListFeature`, and the number of addresses in range from `NumericalDiscreteFeature`.

GSM Location information is also available from mobile phone networks, again in terms of cells. When compared to Bluetooth, GSM has larger cell sizes and consequently a lower accuracy for location tracking, but is far more widely available. In most cities, multiple GSM cells can be reached and therefore allow for deriving location information with an accuracy of a few hundred meters, while in less populated areas a single GSM cell can cover over 30 kilometers and therefore can only provide very rough location information. For every device that is capable of connecting to a GSM adapter via a (physical or virtual) serial port and accessing it with AT commands, a generic feature extractor has been realized to provide the current GSM cell identifier. This currently works under Win32, WinCE and POSIX and includes mobile phones connected via Bluetooth to the respective device (cf. [MRF04b]).
Because the GSM cell id can be conveniently represented as a single string, the single feature provided by this module is consequently derived from `AbstractStringFeature`.

Network The network aspect of context describes the services available to the user, the bandwidth and other network parameters and is important for applications that rely on communication or infrastructure. Currently only for POSIX, this feature extractor can provide a list of all active IP addresses reachable within a given subnet, representing the network aspect of the device context. Although this leads to a slight information loss, the list of IP addresses is interpreted as a list of strings for simplicity; this allows to derive the single feature from `AbstractStringListFeature`.

Power Another indication for the device location can be its connection to a base station or charger. For Win32, WinCE and Linux with APM or ACPI, this feature extractor provides a single, binary feature that indicates if the device is plugged into its charger.
Because this is currently the only binary feature, is is not derived from any of the standard feature types, but is implemented as a special class. Due to its inherently simple design, it does not need persistent static data and therefore implements `Feature` instead of `PersistentFeature`.

Time Time is obviously one of the most important parts of context information. From the system clock, various features like day of week, hour etc. are provided.
This is also a special feature not derived from a standard type, but in this case it is derived from `PersistentFeature` to save and restore the minimum and maximum timestamps.

Video The video feature extractor is currently in development and will provide a number of simple features like the average brightness or the movement and is intended for devices like smart phones that have a built-in, simple CCD camera.

WLAN Wireless LANs can also provide location information, but additionally contribute to the network aspect of context. For location information, the accuracy of WLAN is between Bluetooth and GSM based location tracking. If a device is equipped with a WLAN adapter, the list and number of access points or ad-hoc WLAN stations in range, the current device mode (client, ad-hoc or master), the current ESSID, the MAC address of the access point currently associated to and the signal level to this access point are provided.
The list and number of access points, ESSID, access point address and signal level features are derived from `AbstractStringListFeature`, `NumericalDiscreteFeature`, `AbstractStringFeature`, `AbstractStringFeature` and `NumericalContinuousFeature`, respectively. Like the power feature, the device mode feature implemented in this module is a simple class not derived from the standard features and without support for persistent data.

WLAN access point For information appliances with infrastructure access, remote access points can be queried for their list of WLAN clients, which can be used as an alternative to a locally connected WLAN adapter for deriving a list of devices in range.
Similar to the network module, this feature represents the list of WLAN client MAC addresses as a list of strings and is consequently derived from `AbstractStringListFeature`.

Creating new Modules

For creating new feature extractor modules, it is necessary to implement the `FeatureProvider` interface in a dynamically loadable library and export a global method named `getProvider` that returns a single, static instance of the feature provider on each invocation. The implementations of the `getFeature` and `getSample` methods can return an arbitrary number of different features, as long as each feature is identified by a unique name. All of the methods specified by `FeatureProvider` are defined to be non-blocking. If a feature extractors needs to poll for sensor values, this should be done in a separate thread that is started during initialization. For implementing the respective features, the standard types described above can be used in most cases.
Because integrating new sensors is done in terms of implementing feature extractor modules, with the current framework it is not possible to directly re-use features implemented by different feature providers, i.e. for different sensors. However, in practice the lack of modularization at this level turns out not be an issue, because the available standard feature types typically limit the creation of a new feature provider to wrapping the respective API for accessing the sensor; the effort for applying standard features on the sensor data is marginal.

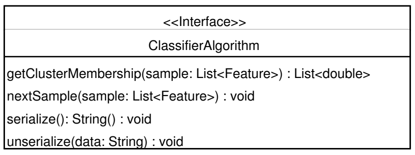

<<Interface>>
ClassifierAlgorithm
getClusterMembership(sample: List<Feature>) : List<double>
nextSample(sample: List<Feature>) : void
serialize(): String() : void
unserialize(data: String) : void

Figure 5.3: Interface for classifier modules

5.4.2 Classifiers

Classifiers are the second modularized part of the framework and are applied to the feature vector data generated by the used feature extractor modules. In Fig. 5.3, the simple interface defined for classifier modules is shown. It is situated at `I2` in the framework overview and, in contrast to the feature extractor interface at `I1`, includes both the input and the output of classifiers. Similarly to the `Feature` interface, there are only two main methods defined in the `ClassifierAlgorithm` interface that need to be implemented by each classifier. `nextSample` constitutes the input interface and is used to feed new feature vectors into the classification algorithm, which is expected to immediately incorporate this new data into its internal structures as demanded by our first requirement. As can be seen from the interface definition, the feature vector is passed in terms of a list of objects implementing the central `Feature` interface instead of a simple numerical vector. This definition allows the classification algorithm to work on a heterogeneous input space with arbitrary types of features, avoiding a mapping to a specific homogeneous vector space that typically implies an information loss. The second main method `getClusterMembership` defines the output interface of classification algorithms and is used to retrieve the class vector as defined in Section 4.2, i.e. a vector of class membership values in the range $[0;1]$, for a given feature vector. As for feature extractors, the methods `serialize` and `unserialize` are used to save and restore the internal data structures and state of a classifier in a general way.
Classifier modules are expected to meet the requirements that have been defined for the classification step in Section 4.6.2.

Available Modules

At the time of this writing, only a single classifier module has been implemented within the framework:

LLGNG Building on the results of the literature survey presented in Section 4.6.3, the Lifelong Growing Neural Gas classification algorithm has been implemented as a classifier module. The advantages of LLGNG with regard to the defined requirements outweigh its complexity when compared to other, simpler methods. In addition to the standard formulation of LLGNG, the concept of meta clusters has been implemented and is used for computing the class membership vector. In this implementation, class memberships are not returned for the individual clusters but for meta clusters defined by the internal topology of the LLGNG neighborhood. Various performance optimizations (as briefly described in Section 4.6.4) are responsible for the computational efficiency of the current implementation.

Creating new Modules

Implementing new classifier modules requires to implement the `ClassifierAlgorithm` interface and create a dynamically loadable library that exports a global method named `getClassifier`. Similarly to

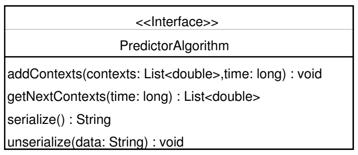

<<Interface>>
PredictorAlgorithm
addContexts(contexts: List<double>,time: long) : void
getNextContexts(time: long) : List<double>
serialize() : String
unserialize(data: String) : void

Figure 5.4: Interface for predictor modules

feature extractor modules, this method should return a single, static instance of the classifier. Loading a new classifier module into the framework during startup can be done trivially by specifying it in the central configuration, which will cause the respective dynamic library to be used. Additionally. the configuration allows an easy masking of features, so that some features are generated within the framework and included in all feature vectors but not used for classification. This allows to provide the selected features to the logging part or directly to applications even if they are known to be unfortunate for the classification step (cf. Section 4.5.6), but has to be considered in all classifier modules by ignoring those elements of the feature vector that are marked accordingly.

5.4.3 Predictors

The third hot-spot for loading modules into the framework are predictors, denoted as `I3` in the framework overview. Fig. 5.4 shows the `PredictorAlgorithm` interface, which also defines both the input and the output of predictors. Following the style of the other interface, there are again two main methods: `addContexts` is used to update the internal models of prediction algorithms with new class vectors, given in the same representation as returned by classifiers, and `getNextContexts` returns a predicted future class vector in the same format. For both methods, a time stamp must be passed that specifies, in the case of `addContexts`, the time when a class vector was generated or, in the case of `getNextContexts`, the time for which the prediction should be computed. This interface closely reflects the more abstract one presented in Section 4.2, but has been defined explicitly for online operation of predictors. Again, the methods `serialize` and `unserialize` are used for saving and restoring the data models constructed internally by prediction algorithms.

Available Modules

In contrast to the single classification module, multiple prediction modules have already been implemented. The reason is that the literature survey presented in Sections 4.8.4 and 4.8.5 did not result in one algorithm that shows clear advantages over other alternatives. Additionally, the trade-off between computational and memory complexity and prediction accuracy can most probably not be decided for the general case. A set of available predictors allows to select an appropriate one for the intended application area. At the moment, the following algorithms for prediction have been implemented either as helper functions or as complete predictor modules:

Statistical tests A number of statistical helper functions have been implemented for testing the hypothesis that a specific time series was created by an IID process.

ACF An approximation of the sample autocorrelation function has been implemented that can support a parameter estimation for other predictors.

Central tendency predictor For comparing more complex prediction algorithms with a “baseline”, a simple average predictor for categorical time series has been implemented with a sliding window approach.

ALZ The Active Lempel-Ziv algorithm has been implemented as one of the most promising candidates for categorical time series prediction.

HMM Although it could currently not be implemented in a completely online way, a HMM prediction module has been written which applies a similar method for predicting categorical time series as described in [RC03]. This module is based on the open source GHMM library available at `http://ghmm.org/`.

Duration predictor A preliminary version of the duration prediction algorithm described in Section 4.8.5 has also been implemented, but is at the time of this writing still in development to improve prediction accuracy.

Besides the statistical tests, currently no continuous prediction techniques have been implemented, but categorical time series prediction is favored for the reasons given in Section 4.8.5.

Creating new Modules

For creating new predictor modules, it is again necessary to implement the respective interface, in this case `PredictorAlgorithm`, and create a dynamically loadable library that exports a method to return a single, static instance of the predictor object, in this case called `getPredictor`. In many cases, the underlying prediction method will be unable to compute complete future class vectors for arbitrary points in time; some methods are only capable of predicting a single, best matching future class instead of the whole probability distribution (i.e. categorical time series prediction methods), and most of the well-known methods are based on time steps with a fixed step width, which makes exact prediction at arbitrary times impossible. The interface shown in Fig. 5.4 includes only the most flexible variant of `getNextContexts`, which can naturally be used for all types of prediction methods. When only a single, best matching class can be predicted, then this class can be set to the maximum probability in the class vectors and all others to zero, and for methods using fixed time steps internally, the returned prediction can either be interpolated or set to the nearest time steps. However, for an easier or more accurate way of dealing with these issues, additional variants of `getNextContexts` have been defined that suit certain types of prediction methods better but are no longer general. These variants can be called directly by applications that query for context predictions, but then those applications depend on the type of prediction method. In the general case, a new predictor module only needs to implement the most flexible variant of `getNextContexts` shown above, but may choose to implement additional ones for simpler access or increased accuracy.

5.4.4 Labeling

The labeling step of our architecture is not within the main focus of this research, as most issues are attached to HCI. Therefore, it has currently not been realized with modules in the same way as the other steps, but by using standard protocols for communication with the independently developed user interface. This has the advantage that the user interface can reside on another device if the applied protocol is network transparent. For context awareness in information appliances without a powerful input/output user interface like display and keyboard, labeling is still possible by executing the interface on spatially close devices that offer appropriate input/output capabilities.

Available Protocols

In the current framework, two protocols have already been implemented that can be used for communication between the core framework and an external user interface application which is responsible for labeling.

COM As an established standard on Windows systems, COM allows comparably simple and efficient communication between independent applications without requiring any further middleware or third-party libraries. In the current framework, COM is implemented in the core framework as well as the example labeling application.

SOAP For communication between heterogeneous platforms, COM is not suitable due to its tight integration with Windows operating systems. SOAP is a standard for exchanging structured messages in a heterogeneous environment, including support for remote procedure calls, and builds upon XML messages transferred over various protocols. HTTP is one of the transmission protocols that can be used by SOAP, with the advantage of being broadly available in TCP/IP based networks. Within the core framework, a SOAP service has been implemented that can export the current, most likely context via a SOAP interface over HTTP. As SOAP is currently not integrated with operating systems, the open source library gSOAP (`http://gsoap2.sourceforge.net/`) is used for this service. Although SOAP guarantees interoperability with clients using other libraries, it necessitates to include a minimal HTTP server in the framework whenever the SOAP service is needed. This server is running continuously in the background and therefore requires a properly working TCP/IP environment on the respective platform and might lead to security issues when no further measures for protection are taken. Generally, the SOAP service allows a flexible implementation of user interfaces for labeling, but imposes additional requirements on the platform that executes the framework.
The corresponding example user interface application has been realized as a traybar applet that can access the exported context identifiers and provide a mapping to context labels as specified by the user.

Creating User Interfaces for Labeling

For developing custom user interfaces that implement the labeling step of the framework in a suitable way, both protocols integrated into the core framework can be readily used. SOAP is the more flexible variant and is easy to use due to extensive library support. A WSDL schema for the SOAP service is included in the source code distribution of the framework and allows the development of compliant SOAP clients. This approach is suitable for research on the labeling step and for building prototype applications, but causes too high an overhead for on-device context recognition and prediction. When the scope and properties of a user interface application have been sufficiently investigated, an implementation directly on top of the framework methods, avoiding the overhead of one of the protocols, is more reasonable.

5.5 Logging

For evaluating a specific context-aware system and selecting appropriate modules, there is a strong need for logging and offline processing during development of context-aware applications. Therefore, the developed framework supports logging of feature and class vectors in a general way, independent of the specific modules. On application demand, logging can be attached to the feature and context class interfaces at the outputs of the feature extraction and classification steps, respectively. For feature vectors, it is also possible to replay recorded logs for offline processing and tuning of classification methods. Details on the current, preliminary log format have been presented in [MRF04a]. It is ASCII based to allow importing the data into third-party software packages and copes with the heterogeneity in feature values, but has been defined without interoperability with other context computing frameworks in mind. Future formats should be based on XML and support different sample rates as well as event streams, which are both not considered in our preliminary format.

5.5.1 Requirements

As already presented in [MRF04a], the definition and usage of this preliminary log format led to a few insights on context recognition data sets: the format should be

easy to use Experience shows that one of the most important properties is the ability to import the data sets into various software packages for data processing. Although our current semicolon separated format was adequate for first tests, XML is more appropriate for public data sets and benchmarks due to better extensibility and tool support. Powerful tools like XSLT processors allow simple transformations of the log data in the case that a specific tool can not directly import it.

self contained Using different files for storing the actual time series and the meta data about features also turned out to be disadvantageous. Even if the handling of a combined data format (with meta data in a header and time series in a log section) leads to more complex writer and parser code, this is compensated by the clearly simpler administrative handling of a single file for each data set instead of two files belonging together. However, in a few scenarios it was necessary to update the meta data file immediately after writing each log line because a clean application shutdown could not always be guaranteed due to low battery failures. To prevent a corruption of the whole data set with invalid meta data, as it had happened before implementing the synchronous update of the meta data file, it must be guaranteed that the meta data section is always up to date. With a combined XML file, we do currently not have a solution for that problem; the whole file would need to be re-written for each new log entry because the meta data header might change. But this would lead to a significant performance degradation. With the current format, it is sufficient to append a line to the log file and update the (small) meta data file.

open During modification and extension of our logging framework over a period of a few months, the definition of an abstract persistent storage area allowed great flexibility in adding new types of features without needing to adapt the log format; we definitely recommend to let each feature (i.e. each dimension of the feature vector) read and write its own persistent storage in the form of arbitrary XML elements. The ability to add additional elements in future versions without breaking older data sets makes XML generally more future-proof.

An extensive data set has been collected using this general logging component within the framework and was subsequently used to compare classification methods quantitatively by replaying the feature log. More details on the data set are given in [RMF04] and the offline comparison is presented in detail in Chapter 6.

5.5.2 Repository for Public Data Sets

For many research fields, a public repository of data sets significantly fostered research on comparing methods quantitatively. For context computing, no commonly accepted data sets are currently available that could be used to compare the context recognition accuracy with other research projects. A first step towards such data sets has been done by the development of a publicly accessible repository at `http://www.soft.uni-linz.ac.at/Research/Context_Database/`. Although a first proposal was to use a Wiki [LC01] as a common repository, which has proven successful for the Portland Pattern Repository [PPR], it has been jointly decided that, at the time of this writing, a moderated data base is more advantageous. In addition to a structured way of storing meta data along the actual data set, it also allows a moderation, both of which is difficult with a Wiki. For future research, this repository can provide common, accepted data sets for evaluating results in context recognition and prediction. A benchmark data set for context recognition could inspire research on context computing and make results of different research groups comparable.

5.6 Open Issues

Although the framework structure is complete and a number of feature extractor, classifier and predictor modules have already been implemented, it has to be considered preliminary and prototypical at the time of this writing. The module interfaces and data flows are complete, but framework services like logging or labeling still need to be improved with regard to stability and fault tolerance when the framework should be used on embedded systems. It is still an open issue if multiple predictor modules will need to be combined to achieve good prediction results or if a single predictor module is adequate.

Chapter 6

Evaluation

In this chapter, a first evaluation of the developed architecture for context prediction is presented. This evaluation is concerned with the principal methodology of predicting context at the level of abstract context identifiers; it should be seen as a feasibility study by proof-of-concept implementation. Although multiple methods for the classification and prediction of context are compared in the following sections, the purpose is rather to allude to the vastly different results than to select a “winner” among the chosen set of methods. It should be pointed out that the classification and prediction accuracies from this evaluation might not be transferable to other application areas, but should only give an indication on which methods are more appropriate than others for context prediction. For a systematic selection of methods, a variety of data sets from different application areas need to be considered, but this is outside the scope of the present thesis. The evaluation presented in the following serves as a proof-of-concept for the developed architecture and its implementation in terms of a software framework.

6.1 Experimental Design

In most current research projects concerned with building context-aware applications, researchers are both the designers and at the same time the subjects of a usability study or system evaluation (cf. [Sch02a, section 2.4]). This “living laboratory research method” is often necessary, as research methods are still being developed and the necessary hard- and software is typically following the most current developments or is even prototypical. Nonetheless, feasibility studies should be conducted not only within such a “living laboratory”, but should include different test subjects or real-world situations. Some current projects like the activity recognition described in [LWJ$^+$04] already collected test data with multiple, arbitrary test subjects to allow variability in the recorded data. To still have some control over the experiments, test subjects were instructed to follow a detailed script of actions as closely as possible. This methodology has proven successful and thus, for creating a benchmark data set for context recognition, detailed scripts should be used for different scenarios. These scripts can then be followed by independent test subjects for collecting sensor data for context recognition, getting closer to real-world applications. Gathering sensor data with test subjects is an issue for future research.
Although such an approach with multiple test subjects allows a better assessment of the results and is thus appropriate for creating benchmark data sets, a different method is employed in this evaluation; besides the personal resources necessary for conducting such a study, it would counter our aim at unobtrusiveness. When aiming for unobtrusive, background applications with continuous learning

and adaptation, a detailed script that is necessarily limited to a short time span can not be considered a viable approach. It can not cover the challenge of learning common contextual patterns and behavior, which is the main aim of the present thesis. Therefore, a data set collected over a longer period seems more appropriate and is consequently used as the basis for the following considerations. An important property of this evaluation is that the framework described in Chapter 5 was used in its standard mode of operation, running in the background without any user intervention. Although this does not allow to compare the results of the classification step, i.e. the recognition of current context, with known values simply because there *are* no known values, it can show the feasibility of a completely unsupervised approach.

From the possible application areas described in Section 2.2.1, a combination of reconfiguration and alerting seems a worthwhile compromise between added benefit for the user and viability. Short-term context prediction with prediction horizons in the range of a few hours can already offer distinct benefits but is still realizable with the current results. For this chosen application area, the office is an obvious setting because alerting is well understood in this environment and reconfiguration is a frequent issue for many office workers who are involved with computer systems. However, it is not the only setting where these issues are relevant and a more complete coverage of a user's contexts is desirable. In the following considerations, almost all situations in which a specific user interacted with computer systems are included, ranging from office settings and lectures to playing computer games. For this scenario, a number of relevant aspects of contextual information can immediately be identified (cf. Section 2.1.3):

Geographical Location is certainly an important aspect of the user context and is considered important for this scenario. The following sensor technologies could be used to extract location information:

- GPS: This is probably the most prominent location sensing technology and is suited well for most outdoor applications.
- GSM: As a cell-based network, GSM provides location information only on the level, i.e. granularity, of the operator defined cells, but provides good coverage including indoor areas.
- Indoor location tracking systems: Ultrasound location sensors allow very accurate tracking of people and objects with an accuracy of a few centimeters, but require infrastructural support and are therefore only suited for closed environments.
- Inertial sensors: Accelerometers and various rotation and tilt sensors can be used to monitor movement and thus, by integration, infer location information. As a relative location tracking method, it is vulnerable to drifts and thus problematic for long-term recordings.
- WLAN: An increasing number of places are already equipped with WLAN access points, including non-office environments like residential areas (e.g. [LFPR04] shows an exemplary coverage map for Frankfurt) and public places (cf. [Fre]). WLAN access points form cells of coverage that can be compared to GSM cells with regard to location sensing, but provide the possibility for client-based tracking of coordinates when multiple access points are reachable.
- Bluetooth: As Bluetooth devices often have a limited radio range of 10 meters, only cell-based location information in the form of spatial proximity can usually be inferred from Bluetooth sensors.

Physical Physical aspects include a wide range of environmental parameters, but movement of the user or device is a promising candidate for recognizing context. Our declared aim of unobtrusiveness and the long-term measurements prohibit the use of body-worn sensing devices like accelerometers or microphones attached to various body parts (e.g. [LL01, LWJ$^+$04]) unless they are completely integrated into clothing and are available ubiquitously. This hampers the direct monitoring of physical aspects of the user context and necessitates the use of indirect ways: one example is the status of the battery charger, i.e. if some device that is usually carried close to the user is plugged into its charger or not. It can be used as a very rough estimate for the physical context of the device. When connected to an outlet, it can be assumed to be immobile, while the converse does not necessarily hold.

Social An important aspect of daily life is the social context, e.g. if other people are present in the proximity of the user and if a conversation takes places. A variety of sensors can possibly be used to infer social aspects:

- Spatial proximity sensors: Infrared, ultrasound or radio sensors can directly monitor the spatial distance between people, but require additional body-worn devices.
- Location tracking: By correlating absolute location information of multiple people, spatial proximity can be indirectly inferred.
- Video: Scene analysis based on video images can also be used to extract various social aspects, but requires the installation of video cameras in the environment and can be problematic with regard to privacy issues. An alternative is to exploit cameras integrated into mobile devices carried by the user, e.g. mobile phones or notebooks, but thorough scene analysis is significantly more complicated when the camera is mobile.
- Audio: Microphones are broadly available in various devices and can be used to monitor the auditory environment. This audio stream provides indicators for social aspects, like the number of participants or possibly also the level of agitation in a conversation.

Technological The network context in this scenario encompasses connectivity and services available to a user or device. It can be easily captured by gathering the list of reachable WLAN access points, which describes both the general connectivity and the services available to the mobile user at each time.

Activity A strong indicator for the user activity regarding the interaction with computer systems is the currently active application, i.e. the application that is currently interacting with the user. Additionally, activities like typing or talking could also be detected by utilizing the microphone and watching for characteristic features like the number of peaks in a certain time frame.

These aspects of the user context are not complete, but capture parts that are considered important for our chosen scenario of tracking the contexts of computer interactions of a single user. In the following, a notebook computer is used for sampling the sensor values due to its mobility and broad range of hardware interfaces. For geographical location tracking, the GSM cell, WLAN access points and Bluetooth devices are monitored. GPS is not used because it is restricted to outdoor location sensing and the intended application area concentrates on indoor activities. Specialized indoor location tracking systems are also not applied because this would restrict the evaluation to a single, closed location. For capturing physical aspects, only the status of the charger is available. Social aspects are captured by a microphone and spatial proximity sensing via WLAN and Bluetooth; other people in range can — at least in current office scenarios — often be detected by the Bluetooth-enabled devices they carry. An unusually high number of Bluetooth devices in close proximity, where “unusual” depends heavily on the specific user, often indicates a meeting situation. The network context of the mobile device is represented by the list of reachable WLAN access points and the activity aspect by the active application or foreground application window at the respective sample time. All of these aspects can be captured with easily available, cheap sensing technology that also satisfies the size, weight and battery life requirements of typical mobile notebook users. Although a notebook computer is actually a counter-example of pervasive computing technology, it was chosen for practical considerations; it is important to note that all of the mentioned sensors could be easily integrated into other mobile devices like mobile phones or PDAs or information appliances in the environment. Most of the sensors are already available in current off-the-shelf smart phones, which could immediately be used as a direct replacement for the notebook as a sensing device. The use of off-the-shelf technology is preferable to custom hardware developments because of the broader scope, as shortly described in Section 4.4.

The most important aim of this research is still unobtrusiveness. Therefore, the data set was collected by a background application running on the notebook and no user interaction was necessary. Although in a practical application, a labeling of automatically learned contexts will be necessary, it is outside the main focus of this thesis and is rather an issue of HCI than of context computing. This evaluation aims to assess if context prediction is feasible in a completely unsupervised way. The predicted context classes can then be used by arbitrary applications for reconfiguration and alerting purposes on the notebook, offering direct benefits to the user. These applications are also out of the scope of this evaluation, we simply assume that predicted contexts can be used to provide new services.

6.2 Data Set

The real-world data set used in this evaluation has been gathered continuously over a period of about two months on a standard notebook computer which was used for daily work. No special considerations were taken during the use of the notebook regarding context recognition. Therefore, the data set should provide representative sensor data for the chosen scenario. A wide range of sensors was used, including:

- microphone
- active/foreground window
- plugged into charger
- WLAN
- GSM

Of these sensors, the respective features described in Section 5.4.1 are used, yielding 28 dimensions in the feature vector after the feature extraction step. One of these dimensions is the time stamp indicating when a sample was recorded. With a sampling rate of one sample in 30 seconds, about 90000 sample vectors were collected, containing different situations in which the notebook was used. It should be noted that the notebook was not running continuously, but was switched off regularly. Therefore, the data set is not contiguous, but contains 38 time frames during which no samples are available. These areas are easily recognizable because the sample vectors are time stamped and restarts of the application are also marked in the log file. The whole data set is publicly available and can be downloaded from the *Context Recognition Database* at `http://www.soft.uni-linz.ac.at/Research/Context_Database/`.

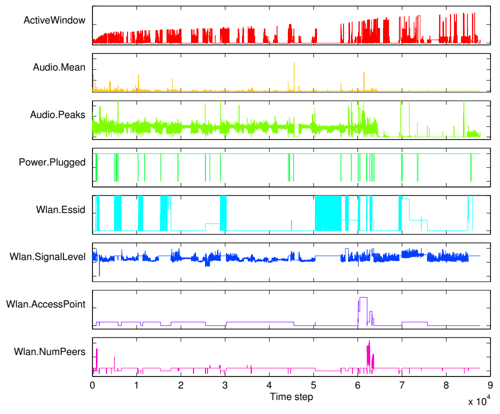


Figure 6.1: Selected features from the real-world data set

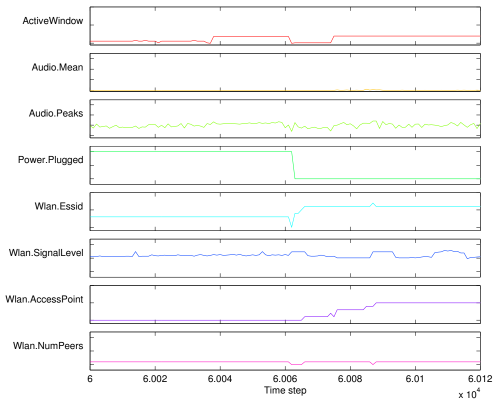


Figure 6.2: Detail view of the data set: 1 hour

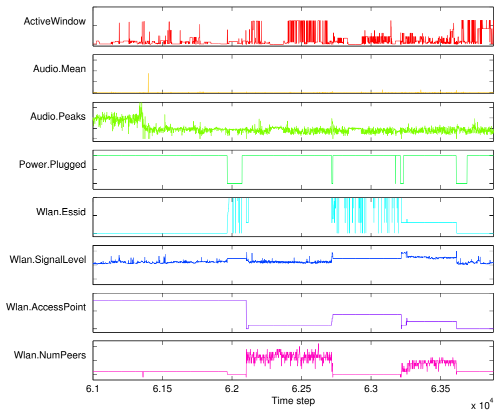


Figure 6.3: Detail view of the data set: 1 day

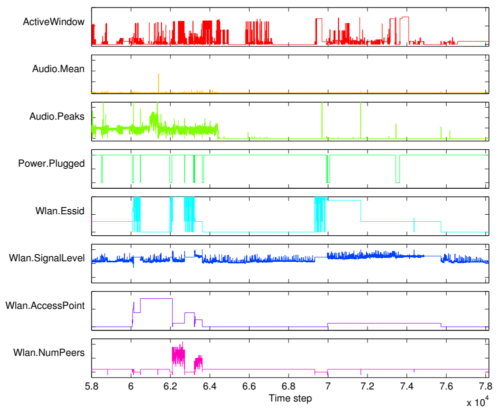


Figure 6.4: Detail view of the data set: 1 week

In Fig. 6.1, a subset of the features contained in the whole data set is shown. The 8 plots depict those feature values that are most likely to have an influence on context recognition:

- `ActiveWindow` represents the name of the currently active application window, abstracted to a numerical id.
- `Audio.Mean` describes the mean value of the audio stream within a time window, i.e. the average loudness.
- `Audio.Peaks` counts the number of peaks within the same time window.
- `Power.Plugged` is a binary feature and is 1 if the notebook was plugged into its charger.
- `Wlan.Essid` represents the name of the currently active ESSID, i.e. the WLAN network connected to.
- `Wlan.SignalLevel` corresponds to the signal strength of the WLAN connection.
- `Wlan.AccessPoint` represents the MAC address of the WLAN access point currently connected to.

- `Wlan.NumPeers` counts the number of WLAN access points or devices in ad-hoc mode in range.

The other features are not shown because they either yield non-atomic values like the list of WLAN devices in range or do not contribute significantly to context recognition like the 16 FFT coefficients extracted from the microphone audio time series.
Table 6.1 shows the mean cross-correlations between all 27 features (excluding the time stamp), interpreted as numerical features, from lags $0 \dots N$ where N is the number of samples. The diagonal elements are not equal to 1 because they describe the mean of the auto-correlation values for all lags, and only the auto-correlation coefficient for lag 0 is $\rho_{ii}(0) = 1$ for the general case. In [BD02, p. 237], a test for independence of two time series is described:

> *[...] the sample cross-correlations $\hat{\rho_{12}}(h)$, $\hat{\rho_{12}}(k)$, $h \neq k$, of $\{Z_{t1}\}$ and $\{Z_{t2}\}$ are for large n approximately independent and normally distributed with means* 0 *and variances n^{-1}. An approximate test for independence can therefore be obtained by comparing the values of $|\hat{\rho_{12}}(h)|$ with $1,96n^{-1/2}$ [...].*

This is valid when $\{Z_{t1}\}$ or $\{Z_{t2}\}$ are pre-whitened time series where at least one of the two series is white noise. However, since the heterogeneity of the feature space does not allow to easily fit a simple ARMA model to each series, we can only apply this test — in a not completely exact way — to the original series. The bounds for $N = 87673$ are in this case $\pm 0,00662$, and it can be seen from Table 6.1 that all features but number 25 are, according to this test, correlated with each other. In [BD02, Example 7.3.1], this test on the original, unfiltered time series also suggested correlations, while testing the pre-whitened series resulted in far less correlations. Another issue is that the time series are not numerical, but are only interpreted as numerical for this test. For non-numerical feature types, the cross-correlation function can not be computed in this standard way. Although we expect to find strong correlations between many of the features, as they have all been extracted from the same environment within the same contexts, the results of this test are arguable for heterogeneous feature vectors.

Although the data set itself does not contain initial transients because there was no set-up or training phase at the beginning of the sample period, online context recognition will need to consider this issue. For a closer inspection of the raw data, Figures 6.2, 6.3 and 6.4 show extracts of one hour, one day and one week respectively. To avoid initial transients in the following evaluation, those extracts are from the later parts of the data set. The considered hour lies is interval $[60000; 60120]$ of sample steps, the day in $[61000; 63880]$ and the week within the sample steps $[58000; 78160]$.
To evaluate the stability of the context recognition with regard to variability in the data set, the whole set has also been split into weeks according to the recorded time stamp. Table 6.2 lists the respective number of samples contained in each of the 12 weeks to give an overview of both the size and the consistency of the individual week data sets. As apparent, the sets are not uniform, and thus the analysis of the individual sets needs to consider their differences in the results. In the following evaluation on the context recognition part, the classification methods will be tested both on the whole data set and on the individual weeks.

As this data set can only represent a single user in a number of situations, it is most probably not representative for context recognition in general. When the public Context Recognition Database becomes more widely recognized and standard data sets become established for benchmarking purposes, a better evaluation will be possible. For the purpose of showing the feasibility of the approach presented in this

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0,10	0,02	0,19	-0,05	-0,05	-0,05	-0,05	-0,05	-0,05	-0,05	-0,05	-0,06	-0,06	-0,06	-0,06	-0,06	-0,06	-0,06	-0,06	0,22	0,07	0,22	0,22	0,11	0,00	0,18	-0,22
2	0,02	0,00	0,04	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	-0,01	0,05	0,01	0,05	0,05	0,02	0,00	0,04	-0,05
3	0,19	0,04	0,34	-0,10	-0,10	-0,10	-0,10	-0,10	-0,10	-0,10	-0,10	-0,11	-0,12	-0,12	-0,11	-0,11	-0,12	-0,12	-0,11	0,41	0,12	0,41	0,41	0,21	0,00	0,33	-0,41
4	-0,05	-0,01	-0,10	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	-0,12	-0,03	-0,12	-0,12	-0,06	-0,00	-0,09	0,12
5	-0,05	-0,01	-0,10	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,03	0,03	0,03	0,03	0,03	0,03	-0,12	-0,04	-0,12	-0,12	-0,06	-0,00	-0,10	0,12
6	-0,05	-0,01	-0,10	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	-0,12	-0,04	-0,12	-0,12	-0,06	-0,00	-0,10	0,12
7	-0,05	-0,01	-0,10	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	-0,12	-0,04	-0,12	-0,12	-0,06	-0,00	-0,09	0,12
8	-0,05	-0,01	-0,10	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	-0,12	-0,04	-0,12	-0,12	-0,06	-0,00	-0,09	0,12
9	-0,05	-0,01	-0,10	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	-0,12	-0,04	-0,12	-0,12	-0,06	-0,00	-0,10	0,12
10	-0,05	-0,01	-0,10	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	-0,12	-0,04	-0,12	-0,12	-0,06	-0,00	-0,10	0,12
11	-0,05	-0,01	-0,10	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	-0,12	-0,04	-0,12	-0,12	-0,06	-0,00	-0,09	0,12
12	-0,06	-0,01	-0,11	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,04	-0,13	-0,04	-0,14	-0,13	-0,07	-0,00	-0,11	0,14
13	-0,06	-0,01	-0,12	0,03	0,04	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,04	-0,14	-0,04	-0,14	-0,14	-0,07	-0,00	-0,11	0,14
14	-0,06	-0,01	-0,12	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,04	-0,14	-0,04	-0,14	-0,14	-0,07	-0,00	-0,11	0,14
15	-0,06	-0,01	-0,11	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,04	-0,14	-0,04	-0,14	-0,14	-0,07	-0,00	-0,11	0,14
16	-0,06	-0,01	-0,11	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,04	-0,14	-0,04	-0,14	-0,14	-0,07	-0,00	-0,11	0,14
17	-0,06	-0,01	-0,12	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,04	-0,14	-0,04	-0,14	-0,14	-0,07	-0,00	-0,11	0,14
18	-0,06	-0,01	-0,12	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,04	-0,14	-0,04	-0,14	-0,14	-0,07	-0,00	-0,11	0,14
19	-0,06	-0,01	-0,11	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,04	0,04	0,04	0,04	0,04	0,04	0,04	0,04	-0,14	-0,04	-0,14	-0,14	-0,07	-0,00	-0,11	0,14
20	0,22	0,05	0,41	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,13	-0,14	-0,14	-0,14	-0,14	-0,14	-0,14	-0,14	0,49	0,15	0,50	0,49	0,25	0,00	0,39	-0,50
21	0,07	0,01	0,12	-0,03	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	-0,04	0,15	0,04	0,15	0,15	0,07	0,00	0,12	-0,15
22	0,22	0,05	0,41	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,14	-0,14	-0,14	-0,14	-0,14	-0,14	-0,14	-0,14	0,50	0,15	0,50	0,50	0,25	0,00	0,40	-0,50
23	0,22	0,05	0,41	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,12	-0,13	-0,14	-0,14	-0,14	-0,14	-0,14	-0,14	-0,14	0,49	0,15	0,50	0,49	0,25	0,00	0,39	-0,50
24	0,11	0,02	0,21	-0,06	-0,06	-0,06	-0,06	-0,06	-0,06	-0,06	-0,06	-0,07	-0,07	-0,07	-0,07	-0,07	-0,07	-0,07	-0,07	0,25	0,07	0,25	0,25	0,13	0,00	0,20	-0,25
25	0,00	0,00	0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	-0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-0,00
26	0,18	0,04	0,33	-0,09	-0,10	-0,10	-0,09	-0,09	-0,10	-0,10	-0,09	-0,11	-0,11	-0,11	-0,11	-0,11	-0,11	-0,11	-0,11	0,39	0,12	0,40	0,39	0,20	0,00	0,32	-0,40
27	-0,22	-0,05	-0,41	0,12	0,12	0,12	0,12	0,12	0,12	0,12	0,12	0,14	0,14	0,14	0,14	0,14	0,14	0,14	0,14	-0,50	-0,15	-0,50	-0,50	-0,25	-0,00	-0,40	0,50

Table 6.1: Cross-correlations between all features

Week	# Samples
0	4958
1	6451
2	14139
3	11595
4	11311
5	7713
6	2428
7	3477
8	2851
9	10057
10	11570
11	1123

Table 6.2: Number of samples in each week of the data set

thesis, a single data set is sufficient; we do currently not strive to optimize towards highest recognition accuracy.
In the following, this data set will be used to evaluate some methods for context recognition and prediction, more specifically for the classification and prediction steps as described in Chapter 4. Some of these methods have been implemented as modules within the software framework described in Chapter 5, while for others various external tools are used for this evaluation. Section 6.3 compares classification methods for context recognition and Section 6.4 compares prediction methods for both continuous and categorical time series prediction.

6.3 Context Recognition

In this section, the context recognition part of the developed architecture is evaluated. This includes the feature extraction and classification steps and serves as the base for context prediction. For reasons described earlier in the qualitative comparison of classification algorithms in Section 4.6.3, the standard classification methods k-means and Kohonen SOM are compared with the enhanced LLGNG algorithm implemented within the framework. Both standard algorithms are unsupervised classification methods successfully used in a wide range of application areas — including context recognition [LL01] — and minimize the same cost function as LLGNG [DWM02]. For the comparison, the Euclidean distance metric has been used for all algorithms wherever applicable. All experiments have been performed on a P4 CPU with 2.4 GHz and 1 GB RIMM memory for comparability of run-time.
Both k-means and SOM are used in a batch training mode and are thus not susceptible to initial transients, while LLGNG suffers from such effects due to its online mode. In general, the results of context recognition in this evaluation could be improved. However, keeping the major aim of unobtrusiveness in mind, it is impossible to even define a good quality measure, because there is no labeled data for comparing the results of the classification methods with known values. For k-means and SOM, the data set has been pre-processed to transform all non-numerical into numerical input dimension by assigning one input dimension for each possible feature value, i.e. the one-of-C method. E.g. for the WLAN ESSID, the single, nominal dimension is split into four binary dimensions, where, at each time step, only one of those dimensions is set to 1 and all others must be set to 0. This transformation yields a 198

dimensional input space with a large number of binary inputs. All dimensions are further normalized to $[0;1]$, as recommended by standard literature on clustering like [Nel01, p. 143]. Since the implementations of the distance metrics specific to each feature (cf. Section 4.5) are also normalized to $[0;1]$, the overall classification error is assumed to be comparable even if the number of input dimensions is different. Although categorical data (e.g. the ESSID identifiers) are not mapped directly to numerical values, which would impose completely different — and most of the time erroneous — semantics on the data dimensions, some properties of the data are not taken into account and could therefore lead to a loss in classification accuracy. E.g. the binary input dimensions are not bound to assume only the values 0 and 1 in the internal structures of the respective clustering methods, but can assume values in between that have no meaning in the feature space. However, when a classification method can not deal directly with heterogeneous input data, the only option is to code it and deal with the possible loss in classification accuracy.

6.3.1 Quality Measure

As a quality measure for determining the classification quality, it is desirable to find one as simple as possible so as to be applicable to a wide range of different methods. Because the problem necessitates the use of unsupervised clustering as already mentioned in the requirements on the classification step in Section 4.6.2, there is no known solution to compare with. Therefore, it is not possible to find a measure for the “correctness” of a context classification for the general case, independent of the application area. Instead, the quality measure should capture how well the computed clusters represent the input probability distribution. Experimenting with two possible quality measures which were used in [DWM02] revealed that both are unable to select a plausible number of clusters for k-means clustering with this data set, and subsequently they are not used for these experiments.
Within the scope of this evaluation, we define the *final classification error* as the *average distance between each data point and its respective best matching cluster after training has been completed*, which is similar to the cost function minimized by k-means (cf. Section 4.6.3). This is an universal criterion suitable for evaluating the classification quality of arbitrary unsupervised clustering algorithms, and it is already well-defined and used in different implementations like the SOM Toolbox for Matlab; lower values for the classification error represent a better adaptation to the feature values and thus a more accurate classification.
In addition to this general quality measure with the mentioned shortcomings, the plausibility of the classification results will also be analyzed briefly. This allows a more qualitative assessment of the context classification results.

6.3.2 K-Means

K-means clustering divides a given data set into k clusters by minimizing the sum of distances between the data points and cluster centers, whereas k has to be pre-determined. Thus, k-means actually does not fulfill one of our requirements for classification methods listed in Section 4.6.2, namely the need for variable topology. It is nonetheless included in this comparison because it is widely used in many application areas. There exist some variants of k-means, e.g. Fuzzy C-Means (*FCM*) or online versions which would fulfill the other requirements but differ only slightly in the principal methodology. Since the number of clusters needs to be pre-determined, the k-means clustering implementation in the Statistics Toolbox for Matlab was used iteratively for $k = 2 \ldots 40$ to determine the optimal number of clusters, maximizing the following simple function:

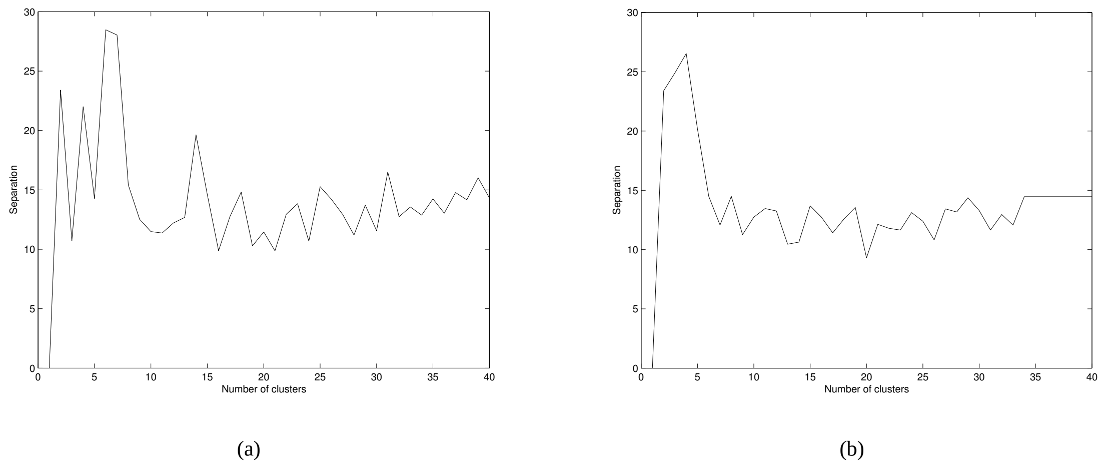


Figure 6.5: K-means classification quality depending on the number of clusters k (two different test runs)

$$q = \frac{1/k \sum_{i=1}^{k} \sum_{j=1}^{k} D_{ij}}{1/(n-1) \sum_{l=1}^{n} (B_l - E(B))^2}$$

where k is the number of clusters, D_{ij} denotes the distances between cluster prototypes i and j and B_l denotes the distances of the lth input vector to its respective best matching cluster. It computes the mean distance of clusters to each other, normalized by the variance of distances between input vectors and their assigned cluster, or in other words, the distance of clusters in relation to their compactness.

The value of this criterion depending on the number of clusters k can be seen in Fig. 6.5 and shows quite clearly a peak for $k = 6$ in the first run and $k = 4$ for the second run. It is well known that k-means clustering is highly dependent on the (random) initialization of its cluster prototypes. For the detailed evaluation of k-means, 6 clusters are used to achieve a lower final classification error. Clustering the whole data set to 6 clusters took 84 seconds on our reference computer with a final classification error of 0,7451.

In Fig. 6.6, the assigned clusters are depicted for each time step in the initial feature data set. Figures 6.7, 6.8 and 6.9 present the assignment of clusters for the extracted time frames of one hour, one day and one week respectively. In the feature values depicted in Fig. 6.2, the active window and audio peaks features show changes shortly before time step 60040 and the active window, power and WLAN ESSID features show changes around time step 60060. Although the cluster trajectory for one hour shows plausible context class transitions at those times, these transitions are only temporary and the trajectory immediately returns to the previous context, which is not justified by the feature values. The trajectories for one day and one week show sensitivity to the power and WLAN ESSID features, but seem unstable and oscillate quickly between contexts. Without labeled data to compare with, a rigorous analysis of the consistence of context classes with real-world situations can not be performed. However, for a data set of 2 months resulting in 6 context classes, it is expected that the respective contexts are rather high-level, which is not supported by the oscillations in the trajectories computed by k-means. Examples for such oscillations can be seen in Fig. 6.8 between time steps 62000 and 62500.

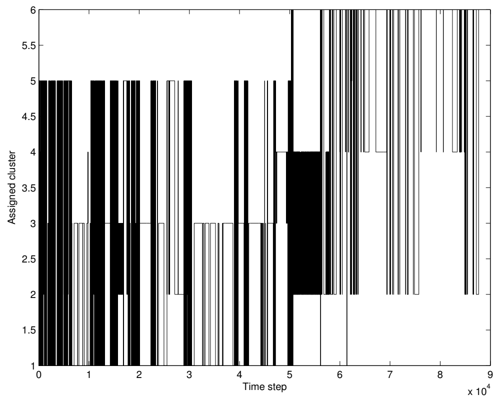


Figure 6.6: Cluster trajectory computed by k-means clustering

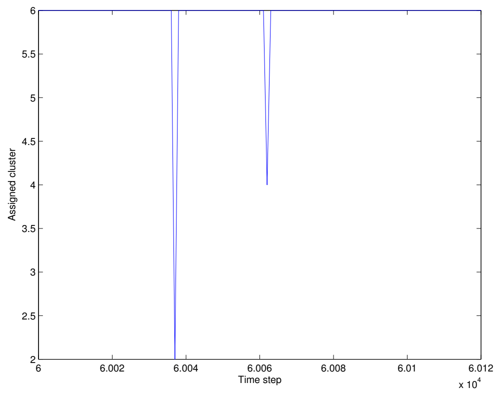


Figure 6.7: Detail view of the k-means computed trajectory: 1 hour

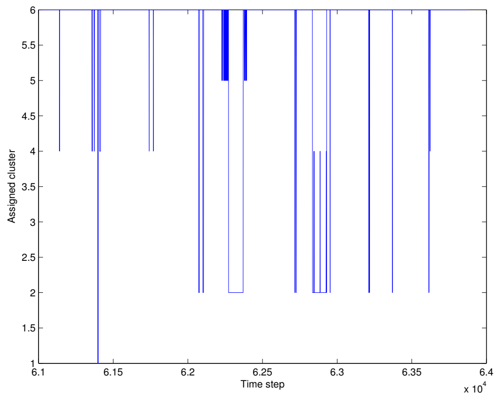


Figure 6.8: Detail view of the k-means computed trajectory: 1 day

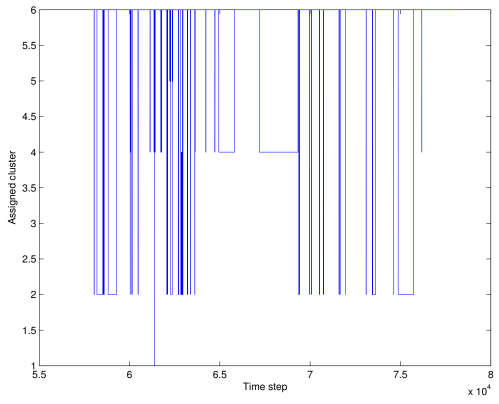


Figure 6.9: Detail view of the k-means computed trajectory: 1 week

In addition to the whole data set, k-means is also applied to the feature values of the 12 individual weeks. This allows to assess the stability of k-means with regard to different data sets. As k-means is sensitive to its random initialization, 5 test runs are performed and the mean classification error of these test runs is taken for each of the 12 weeks. Table 6.3 lists the classification errors for each trial run, the mean and variance for each week and the overall mean and variance of the mean classification errors of each week. The overall mean classification error of 0,6469 is slightly better than the error achieved for the whole data set at once, suggesting that k-means is better suited for smaller data sets; increasing the number of clusters for the larger data set does not improve the result quality as evident from Fig. 6.5.

K-means clustering in this form is infeasible for embedded systems due to the enormous computational power necessary to optimize the number of clusters; determining the number of clusters for this test case took over 2 hours with 5 computers similar to our reference machine being used in parallel.

Week	Classification error					Mean error	Variance of errors
	1	2	3	4	5		
0	0,9865	0,9865	1,0542	1,0209	1,0383	1,0173	9,29E-4
1	0,6869	0,6869	0,6869	0,6869	0,7361	0,6967	4,84E-4
2	0,5919	0,5905	0,5919	0,5919	0,5905	0,5913	5,58E-7
3	0,7907	0,5765	0,5770	0,7878	0,5770	0,6618	1,35E-2
4	0,3283	0,3364	0,3364	0,3364	0,3364	0,3348	1,28E-5
5	0,1784	0,1784	0,1784	0,1784	0,1784	0,1784	0
6	0,4906	0,4706	0,5167	0,4028	0,4745	0,4711	1,80E-3
7	0,8720	0,8095	0,8560	0,8092	0,8125	0,8318	8,96E-4
8	0,9404	1,0499	0,9404	1,0619	1,0619	1,0109	4,20E-3
9	0,7111	0,7055	0,7266	0,7055	0,7231	0,7144	9,83E-5
10	0,8209	0,8090	0,7619	0,8175	0,8417	0,8102	8,74E-4
11	0,5229	0,3639	0,3639	0,4452	0,5229	0,4438	6,30E-3
					Mean	**0,6469**	
					Variance	0,0664	

Table 6.3: K-means classification errors for each week of the data set

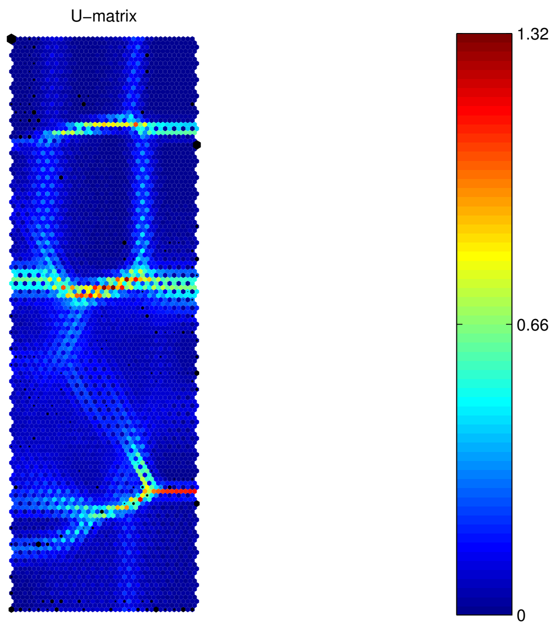


Figure 6.10: SOM u-matrix showing the distances between clusters and hits for the clusters

6.3.3 Kohonen SOM

The Kohonen SOM is, in contrast to k-means, a soft clustering approach: each input sample is assigned to each cluster with a certain degree of membership. For the purpose of this comparison, this can easily be reduced to hard clustering by searching for the so-called “best matching unit”, which is then assumed to be the context class for the respective time step. The following evaluations were performed with the specialized SOM Toolbox for Matlab developed by the Laboratory of Computer and Information Science at Helsinki University of Technology [VHAP99] because of its flexibility and simple means of visualization.
Training a new SOM with the whole data set took 690 seconds and results in a final classification error of 0,5659. The SOM grid is automatically set to 71x21 clusters with a heuristics embedded in the toolbox. There is no need to change the defaults, because the u-matrix, shown in Fig. 6.10, looks plausible. For an SOM, its u-matrix is defined as the unified distance matrix, where each element $u_{i,j}$ is computed as the distance between the map units i and j. Thus, it visualizes the distances between adjacent cluster prototypes and can be used to analyze if the input space is quantized evenly. It is probably the most popular method for displaying the structure of SOMs. The u-matrix for this SOM shows some clearly visible classification of the input space, as the clusters are not spread evenly. In Fig. 6.10, yellow to red colored elements represent areas with larger distances and the sizes of the black dots, which visualize the number of hits on the respective best matching units, are also not evenly distributed. In this case, the u-matrix indicates around 4 to 8 larger areas, which seems reasonable when compared to the 6 clusters found by the k-means method and which can be seen as some form of meta clusters. Although the final classification error is lower than for k-means clustering, this is not surprising because of the significantly larger number of cluster prototypes that are available for mapping the input space. It is currently unclear if meta clusters could be derived in an unsupervised way from a SOM structure; the u-matrix is typically used for visualizing a data set to allow human inspection. The large number of clusters formed by the SOM could not be used for directly representing high-level context, a second classification step would be necessary for the labeling, as in [Lae01].
The clusters assigned to the feature values for each time step are shown in Fig. 6.11 as the numbers of respective best matching units. For the one hour extract shown in Fig. 6.12, the trajectory is also unstable in the earlier part, but seems plausible starting from the transition shortly before time step 60060. In the period between 60040 and 60060, where the active window and audio peaks features in Fig. 6.2 exhibit a visible change, the trajectory is stable and a distinct cluster has been assigned. After 60060, where the power and WLAN ESSID features change, a different cluster is selected. On the other hand, the trajectories in the extracts of one day and one week, shown in Figures 6.13 and 6.14, show oscillating behaviour at this level of detail. Without a second step of clustering to exploit some form of meta clusters formed by the SOM, the resulting cluster trajectories seem unusable for context prediction on an abstract level. Additionally, the trajectory of the whole data set presented in Fig. 6.11 shows signs of unfavorable separation of clusters into areas in the input space around time step 50000: the apparent switch to a completely separate region of cluster prototypes is not supported by the visualization of the feature values. This effect should not be confused with catastrophic interference (cf. [Ham01]), where the network forgets older patterns and which can only occur in an online operation but not in the batch training approach used here. A change to different clusters around the time step 50000 is also visible in the k-means trajectory in Fig. 6.6, but it is not as drastic as for the SOM. This is especially bad for context prediction, where a model of the previous data will lead to erroneous forecasts when such sudden changes occur. A stable representation of context classes is of uttermost importance to achieve

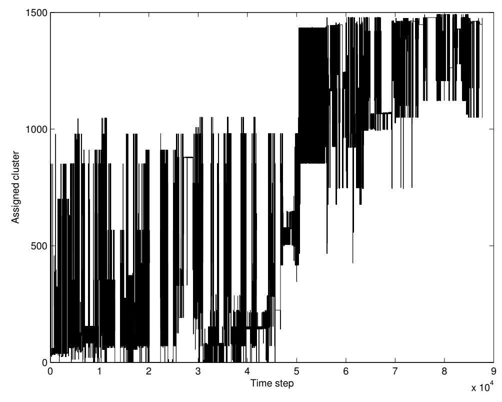


Figure 6.11: Cluster trajectory computed by the SOM

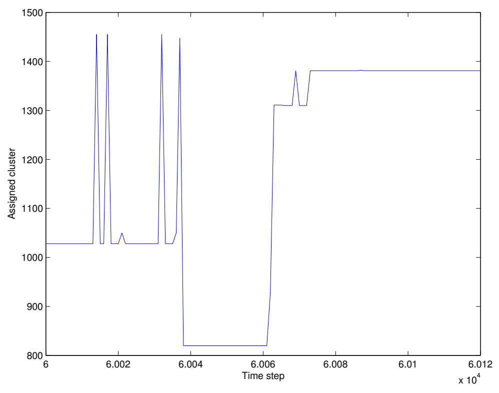


Figure 6.12: Detail view of the SOM computed trajectory: 1 hour

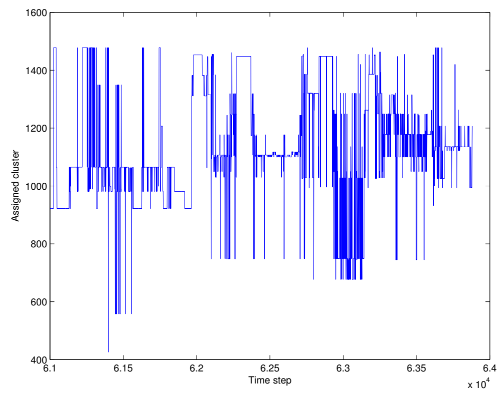


Figure 6.13: Detail view of the SOM computed trajectory: 1 day

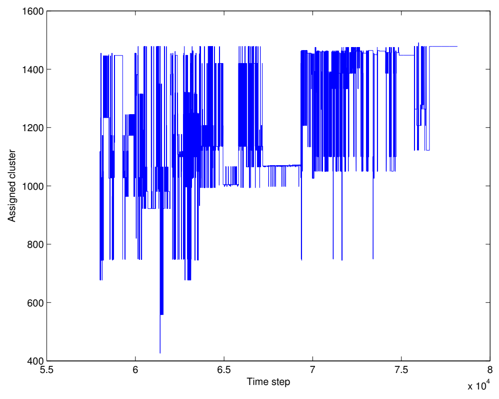


Figure 6.14: Detail view of the SOM computed trajectory: 1 week

usable results in prediction.

In the same way as done for k-means, Table 6.4 lists the SOM classification errors for the individual weeks. By also performing multiple test runs, it turned out that the SOM produces stable results for each of the individual weeks and that the classification errors are equal for consecutive test runs. Therefore, an averaging over 5 test runs is not necessary for the SOM. The mean classification error of 0,1342 is significantly lower than the error achieved for the whole data set, which is expected given the abrupt change in cluster regions visible in Fig. 6.11. Like k-means, the SOM also seems to cope better with smaller data sets for context recognition purposes and therefore seems unsuitable for continuous classification with consequently arbitrarily large data sets.

The heterogeneous variation of the SOM [NB01] might perform better on our data set, because the mapping of categorical data to higher-dimensional numerical data would not be necessary, reducing the number of input dimensions. Unfortunately, no implementation of this algorithm was available and it was infeasible to implement it in Matlab within the scope of this evaluation. It was also not possible to easily work with the meta clusters visible in the u-matrix, which might also lead to more stable results.

Week	Classification error
0	0,2443
1	0,2276
2	0,1838
3	0,1953
4	0,0715
5	0,0202
6	0,0942
7	0,0971
8	0,2708
9	0,1090
10	0,0671
11	0,0292
Mean	**0,1342**
Variance	0,0074

Table 6.4: SOM classification errors for each week of the data set

Meta cluster	Number of clusters	Number of hits	Average distance	Variance of distances
1	2	1489	0,003989	0,000166
2	1	34229	0,002864	0,000138
3	4	10422	0,016849	0,000285
7	3	735	0,083949	0,008128
8	7	9235	0,002995	0,000125
9	3	173	0,005163	0,000422
10	5	580	0,040609	0,008482
11	6	5289	0,015166	0,000255
12	72	25521	0,005366	0,000201

Table 6.5: Meta clusters constructed by the extended LLGNG

6.3.4 Extended LLGNG

Prior to training the extended LLGNG algorithm with this test data set, a simulated annealing procedure was used to optimize some of its static parameters that were explained in more detail in Section 4.6.4. However, the optimization does not significantly improve the classification error, suggesting stability against changes in the initial conditions or parameter sets and supporting the findings reported in [DWM02]. The final parameter setting was $\lambda = 10$ (a new unit is inserted after this number of input samples), $\vartheta_{delY} = 0,5$ (an unit is only deleted is it is older than this, i.e. their youth must be lower than this threshold), $\vartheta_{del} = 0,5$ (an unit is only deleted if its deletion criterion K_{del} is is lower than this threshold), $\vartheta_{delB^L} = 0,15$ (an unit is only deleted if its quality measure for leaning B^L is lower than this threshold), $\vartheta_{age} = 30$ (edges that are older than this threshold are removed), $T_S = 6$, $T_L = 170$, $T_Y = 50$, $T_\vartheta = 55$ (the time constants for decreasing the short-term and long-term error counters, the age of the best-matching unit and the insertion threshold, respectively), $\vartheta_{ins} = 0,02$ (the insertion tolerance used for determining the quality measure for insertion B^I), $\vartheta_L^i = 0,0005$, $\eta_b = 0,15$, $\eta_n = 0,01$ (the adaptation threshold, learning rate of the winner and learning rate of the neighbors, respectively, used

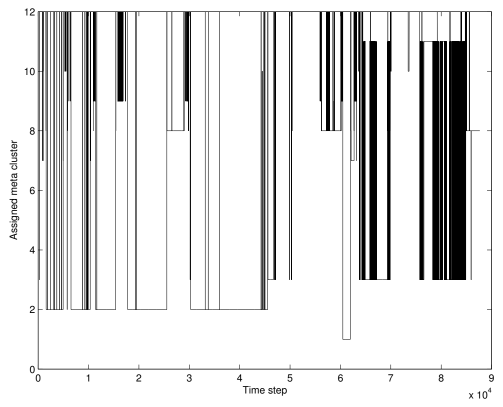


Figure 6.15: Cluster trajectory computed by the extended LLGNG

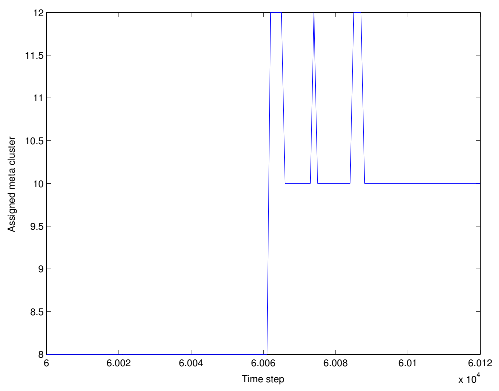


Figure 6.16: Detail view of the extended LL-GNG computed trajectory: 1 hour

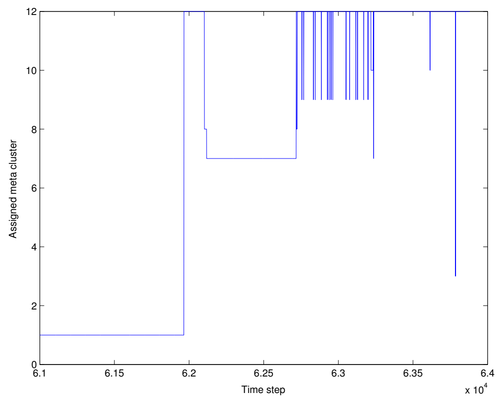


Figure 6.17: Detail view of the extended LL-GNG computed trajectory: 1 day

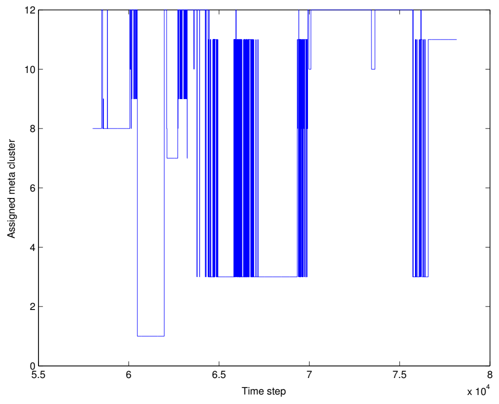


Figure 6.18: Detail view of the extended LL-GNG computed trajectory: 1 week

for determining the learning rate η^i), $\eta_\vartheta = 0,2$ (the learning rate of the adaptive insertion threshold) and a minimum learning rate of $\eta_{min} = 0,2$. For a detailed discussion of these parameters, we refer to [Ham01].

One-pass training with the whole data set took only 474 seconds, produces 103 clusters composing 9 meta clusters and achieves a final classification error of only 0,0069. This means that the input data distribution is well approximated by the automatically found clusters, even with the inherent noise in our real-world data set and significantly less clusters than used by the SOM ($71 \cdot 21 = 1491$). Because GNG/LLGNG are not restricted to a user-defined, fixed number of dimensions in the neighborhood topology like the SOM, their internal structure can not be easily visualized. Instead, Table 6.5 shows the meta clusters constructed during training, with the number of hits and the mean and variance of the distances of all assigned data samples to the respective meta cluster. With respect to the average distances, the respective variances indicate a small intra-cluster variance and thus good separation of meta clusters. For a quantitative assessment of this separation, the inter-cluster variances would need to be compared with the intra-cluster variances, but the former are currently not available as no sensible way of computing them has been found.

Fig. 6.15 shows the best matching meta clusters for each time step; the meta clusters are a higher-level concept than clusters and are thus suited better as abstract context identifiers. The hour extract of the trajectory shown in Fig. 6.16 shows that the change around time step 60060 is detected, while the change shortly before 60040 is ignored by LLGNG. This is supported by the fact that more features changed simultaneously at the later event. However, later changes in the trajectory before and after time step 60080 are not supported by significant changes of the features shown in Fig. 6.2, which suggests that the non-atomic features that are not shown exhibited changes. Fig. 6.17 shows that the cluster trajectory of the day extract looks plausible when compared with the respective feature values from Fig. 6.3, although it seems to be mostly sensitive to the WLAN ESSID in this time frame. The same behavior is visible in the week extract in Fig. 6.18, but within this time frame other features like the active window also seem to have an influence. When comparing the trajectories computed by k-means, SOM and the extended LLGNG variant, the latter ones are more stable, showing fewer oscillations, with the notable exception of the time frame between time steps 64000 and 70000 in Fig. 6.18, where the LLGNG trajectory also shows oscillating behavior.
It should be noted that, unlike SOM and k-means, the extended LLGNG is used in online mode, which is far more challenging. Without any further changes, the algorithm can immediately be used for continuous learning over arbitrary periods of time. K-means and SOM both had to be trained to produce the above results, with each sample being presented numerous times for training. The extended LLGNG had only a single pass over the whole data set and still achieves a significantly lower classification error and more stable results. One suspected reason for this success is that our extensions that enable LLGNG to deal directly with heterogeneous data indeed lead to higher classification quality due to the lower-dimensional input space and due to the preservation of the semantics of all feature dimensions.
The fact that LLGNG is used in online mode with a single pass over the respective data sets is also the suspected reason for its comparatively poor results on the individual data sets. Table 6.6 shows the respective final classification errors for each of the 12 weeks and the mean error of 0,1178, which is larger than the final classification error achieved on the whole data set by a factor of approximately 17. Without multiple epochs, LLGNG is presented with only a small number of training samples, e.g. 1123 for week 11, which seems to hinder its performance. However, the mean classification error and variance over all individual week results are still lower than those of k-means and SOM, even if the difference is not as pronounced as for the results on the whole data set.

In the following, the context class trajectory computed by LLGNG and shown above in Fig. 6.15 will be used for further investigations. There are two reasons for preferring it over the k-means and SOM results within the scope of this evaluation: Most importantly, the results were produced in a completely unsupervised and online way, which fulfills the core aims of this research. Additionally, the trajectory is more stable and thus promises slightly better results for context prediction.

6.4 Context Prediction

After showing the feasibility of online, unsupervised context recognition with a single classification step and limited resources in the last section, the resulting context class data can be used as the base for an evaluation of the context prediction step. To shortly recapitulate the interface between the classification and the prediction steps, the output of classification is a m-dimensional vector with degrees of membership for each context class. Therefore, the prediction step builds upon a m-dimensional trajectory of numerical values in the interval $[0;1]$. In this evaluation, there are 87673 class vectors with 9 dimensions

Week	Classification error
0	0,0249
1	0,1151
2	0,1559
3	0,0104
4	0,1860
5	0,0474
6	0,1376
7	0,1708
8	0,0203
9	0,1741
10	0,1853
11	0,1857
Mean	**0,1178**
Variance	0,0047

Table 6.6: LLGNG classification errors for each week of the data set

each, which correspond to the meta clusters formed by the extended LLGNG algorithm. The degree of membership of each of the meta clusters to a feature vector is defined as the distance of the nearest cluster within the respective meta cluster. As in the evaluation of classification algorithms, this section builds upon a single data set and can therefore not provide a general assessment of various methods for arbitrary application areas. In the following, a number of prediction methods will be used on the present data to predict future context classes and therefore evaluate the possibilities for context prediction in this scenario. Before the actual prediction is performed, a quality criterion for measuring the prediction accuracy in a quantitative way is defined and the time series is subjected to a brief analysis.

6.4.1 Quality Measure

For the classification step, the correct classification results were not even known in the offline, batch training mode that could be applied in the above section. In contrast, for the prediction step known results to compare with are available, at least for the batch mode used in this evaluation. The aim is to predict the future development of the context class trajectories; when using only a part of the known time series for training the prediction methods, which is usually called the *training set*, the remaining part, usually called *test set*, can be used for comparing the computed prediction with the real development of the trajectories. This way, prediction can actually become a supervised approach (but without human supervision).

For supervised approaches, it is possible to find better quality measures than for unsupervised ones. In the following, both options for time series prediction discussed in Section 4.8.3 will be evaluated, but the applied quality measures differ. For continuous time series prediction, the well-known Root Mean Squared Error (*RMSE*) is used for assessing the result quality by comparing the true time series from the test set with the series predicted by the specific methods. For categorical time series prediction, a loss function known as 0/1 loss is applied, which basically counts the number of erroneous predictions. These quality measures are explained in more detail in the respective sections.

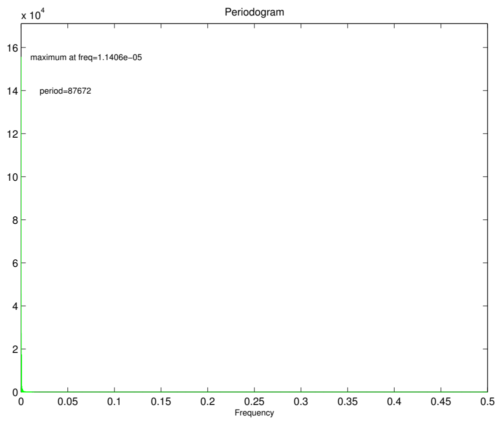


Figure 6.19: Periodogram of the aggregated, categorical time series of context classes

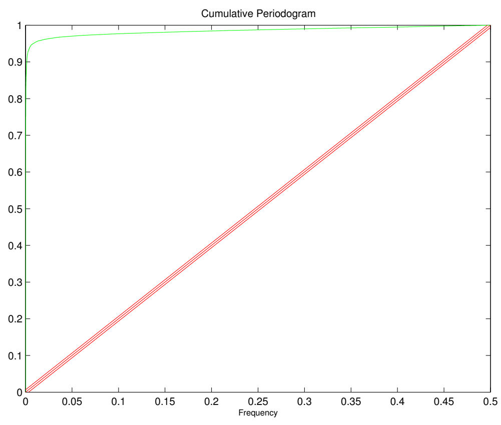


Figure 6.20: Cumulative periodogram of the aggregated, categorical time series of context classes

6.4.2 Analysis of the Data Set

For an initial assessment of the context class time series that should be predicted, a few statistical methods are appropriate. Continuous time series prediction methods produce richer data than categorical prediction methods, because with continuous methods, the detailed developments of all context class memberships can be predicted, while categorical methods predict the context class trajectory of best matching contexts. For comparability of continuous with categorical prediction methods, the categorical time series needs to be used and is subsequently the subject of this brief initial analysis of the time series data.

A direct visualization of the categorical time series used for the following considerations is given in Fig. 6.15. Visual inspection suggests that the series contains periodic patterns which could be exploited for prediction purposes. The periodogram and cumulative periodogram are established visualization techniques that show the period of cycles in a time series. Figures 6.19 and 6.20 show the periodogram and cumulative periodogram, respectively, and it can be observed that, contrary to the expectation, no significant periods are evident. Another common approach to time series inspection are the (sample) autocorrelation function (*ACF*) and (sample) partial autocorrelation function (*PACF*), shown for this time series in Figures 6.21 and 6.22, each for a maximum lag of up to 50. The slowly decaying positive ACF suggests the application of a differencing operator before fitting a simple ARMA model (cf. [BD02, p. 181f]), which will be evaluated below in Section 6.4.3. More generally, this characteristic shape suggests a long-term dependency within the time series. The shape of the PACF suggests a first- or second-order auto-regressive component, often denoted as AR(1) or AR(2), respectively, for the original time series.

On a more qualitative level, a few possible issues for this time series can be identified from knowledge of the problem domain and the experimental setup. As the notebook recorded sensor data in different contexts depending partially on the time of the day, we would expect a few periodic patterns with different period lengths to be contained in the data. On the other hand, linear or higher order trends are not expected to be contained in the data. So-called *aging* effects could, in this scenario with the chosen time frame for the whole experiment, not be caused by deterioration. It is also unlikely that learning on

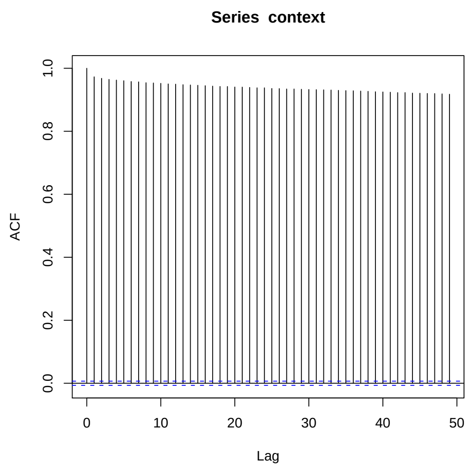


Figure 6.21: Autocorrelation function (ACF) for the categorical time series of context classes

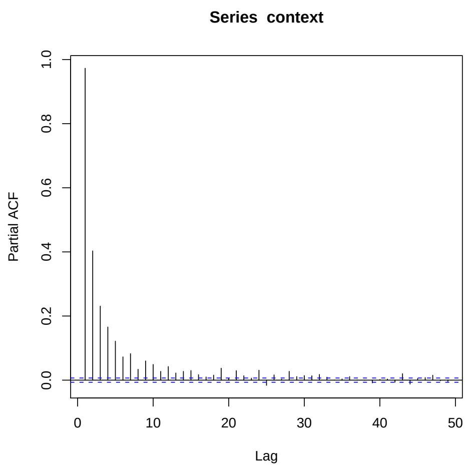
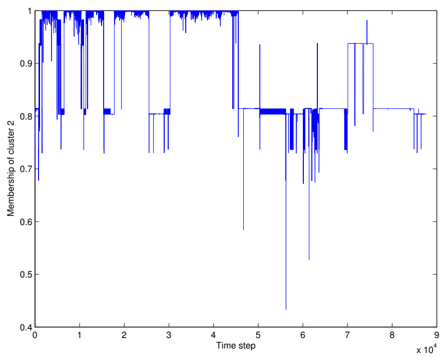


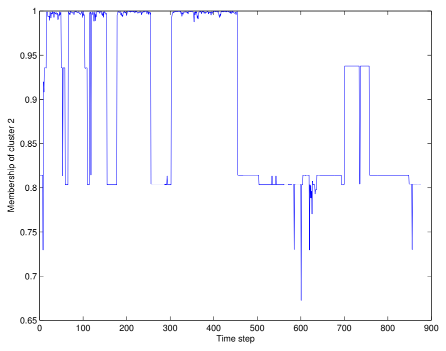
Figure 6.22: Partial autocorrelation function (PACF) for the categorical time series of context classes

the part of the user, another possible cause for aging effects, will have noticeable impact within the rather short period of two months, in which the user has attended no special training. However, it is nonetheless expected to be a difficult problem for time series prediction methods, because the behavior of a single user is often unpredictable without detailed insight into his intents or emotional states. In contrast to predicting the behavior of a large number of people, which is e.g. frequently performed by insurance companies, the law of large numbers does not apply to this case. When observing a sufficiently-sized population, random effects tend to compensate each other. For context prediction, where a single user is in the focus of the time series data, random changes in behavioral patterns are rather the rule than the exception.

For continuous time series prediction, a detailed inspection of one specific context class trajectory can give more insight into the suitability of the compared methods. The predicted categorical time series, which is necessary for comparison with categorical prediction methods, will be constructed by aggregating all individually predicted context class trajectories, taking the respective best matching, i.e. highest ranked, context for each time step. From the nine possible time series that are available, i.e. the nine distinct context class trajectories, time series number two is selected for a more detailed analysis. The reason for this selection, which is based on a visual inspection of all nine trajectories, is that the second trajectory features the largest variability and shows periodic patterns. This makes it challenging and thus interesting for continuous time series prediction. As apparent from Fig. 6.23, it includes periodic patterns in the beginning and a pattern change in the end and the degree of membership of the associated second context class varies from lower than $0,5$ to 1. Other class trajectories do not show such clear patterns or are noticeably correlated. To cope with the size of the data set and speed up training time, only every hundredth value was included in the time series, effectively reducing the temporal resolution by a factor of 100. This should actually improve long-term prediction accuracy, as a single prediction step in this lower-resolution time series accounts to hundred steps in the original one.



(a) full resolution



(b) reduced resolution, by factor 100

Figure 6.23: Trajectory of the second context class

6.4.3 Continuous Time Series

Continuous time series prediction can be seen as a mapping of multiple input values to one or multiple output values and thus the aim is to find a model for this mapping. Three methods seem particularly well suited for this purpose: ARMA, as a specialized time series prediction method, and the general backpropagation Multi-Layer Perceptron (MLP) and Support Vector Machine (SVM) as universal function approximators. While SVM is only suited for single-step prediction, the ARMA and MLP methods can deal directly with multi-step prediction, though with different approaches. Independently of single- or multi-step prediction, the same quality measure can be applied, as the aim is for both to predict the trajectory as closely as possible. The RMSE has been widely established as a standard error measure not only for time series prediction and is defined as:

$$e = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

where y_i and $\bar{y}_i$ are the true and predicted values of the time series at time step i.

It should be noted that all three methods for continuous time series prediction evaluated in this chapter are actually batch training approaches and can not be used easily in an online mode. Thus, they violate the requirement of online learning as explained in Section 4.8.1.

6.4.3.1 ARMA

For evaluating ARMA models, the specialized ITSM2000 software package, which is bundled with [BD02], has been used. Contrary to the expectation based on the shape of the sample ACF as shown in Fig. 6.21, only differencing the time series does not yield any usable results, as evident in Fig. 6.24. The maximum likelihood evaluation based on the AICC criterion (cf. [BD02]) found, after differencing the time series once, an ARMA-2-2 model as best fit, but the prediction of 50 steps starting from time step 200 suggests that the model does not fit the present time series. In this section, an ARMA-p-q model denotes a model with AR(p) and MA(q) components, i.e. with auto-regressive terms

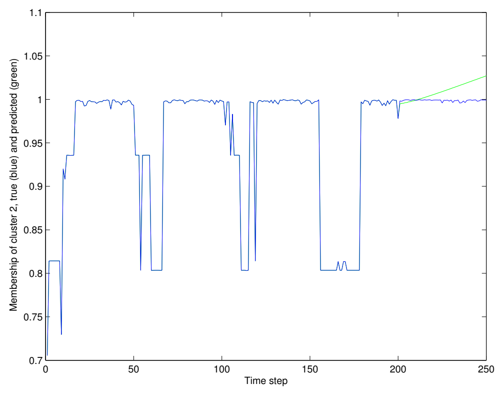


Figure 6.24: ARMA forecast of the second context class trajectory with differencing

Prediction number	Start of forecast (time step)	RMSE
1	50	0,0347
2	100	0,0479
3	150	0,1032
4	200	0,0608
Mean		**0,0617**

Table 6.7: RMSE values of ARMA forecasts

of order p and moving average terms of order q. Applying a seasonal fit with a lag of 55 before fitting an ARMA model yields significantly better results. All forecasts shown in Fig. 6.25 were created with a seasonal correction with that lag and the ARMA-1-1 model found by autofitting, i.e. doing an exhaustive search for models between 0-0 and 5-5 for the best performing one, rated again by the AICC criterion. As the data series with reduced resolution is still manageable, the grid search approach is chosen over heuristics to achieve the best possible model and eliminate any possibility for local minima in optimization procedures. The first three predictions of 50 steps each, starting at time steps 50, 100 and 150 respectively, show a reasonable match between the true and the predicted trajectories. Around time step 200, a change in the periodicity of the trajectory occurred, which could naturally not be captured by the simple ARMA model as shown in the fourth prediction in Fig. 6.25. The respective RMSE values for each of the four predictions are given in Table 6.7.
The remaining issue is to find the lag for the seasonal fit. During this evaluation, the lag of 55 was found by visual inspection and trial-and-error, which is the typical [BD02, p. 188]:

> *Trend and seasonality are usually detected by inspecting the graph of the (possibly transformed) series.*

However, this approach is not possible for estimating the model automatically in an unsupervised way. Exhaustively trying all possible lags with all possible ARMA models might be an option for batch training and in environments where computational resources or computing time are not an issue, but it is inadequate for embedded systems. An estimation if there are seasonal components in the time series can be extracted from the ACF [BD02, p. 188]:

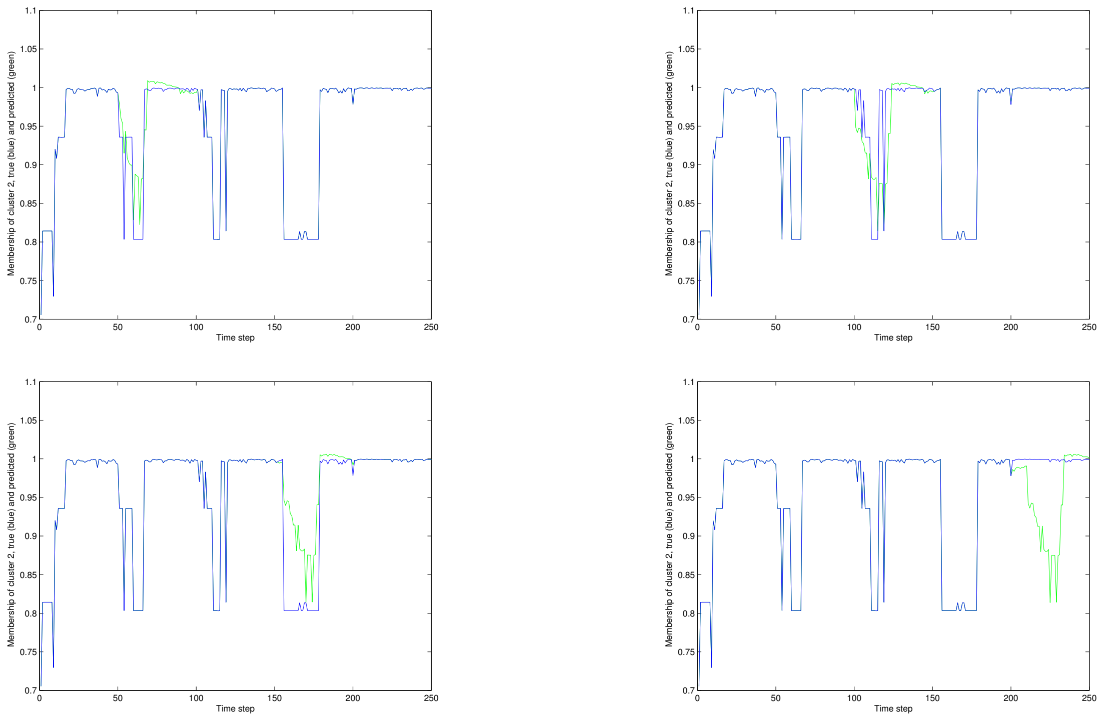


Figure 6.25: ARMA forecast of the second context class trajectory with seasonal correction

However, they [= trend and seasonality] are also characterized by autocorrelation functions that are slowly decaying and nearly periodic, respectively.

Not only the fact that seasonal components are present, but also suitable lag values for a seasonal correction can be indicated by the ACF, which should show local peaks for those lags that describe seasonal aspects of the respective time series. Fig. 6.26 shows the autocovariance function (ACVF) for the considered trajectory of the second context class with a peak near lag 5000, although it is not as pronounced as in the autocovariances of the categorical context class trajectory of the respective best matching classes shown in Fig. 6.27 (which includes larger lags than computed in the initial analysis in Section 6.4.2). This peak seems plausible when compared with the lag of 55 — at a resolution reduced by factor 100 and thus equivalent to a lag of 5500 in the original time series — and could be used for an automatic fitting of seasonal correction. However, a specific method to derive seasonal lags from a possibly incrementally computed ACF is not further investigated within the scope of this evaluation.

For a better comparability with approaches based on categorical time series forecast, ARMA models of all 9 context class time series are created. They are also constructed with a seasonal fit with lag 55, but with different orders for the AR and MA terms, which were found by autofitting. On the 9 distinct models, 50-step forecasts are computed starting from time steps 50 and 150, because those were respectively the most successful and least successful for the second class trajectory with regard to the RMSE, as apparent from Table 6.7. The resulting forecasts are then aggregated by searching for the respective maximum degree of membership for each time step and taking the context class that is attributed with this maximum activation as (categorical) value for the time step. Fig. 6.28 shows that the true (blue) and predicted (green) time series differ only slightly. The respective 0/1 loss values for the

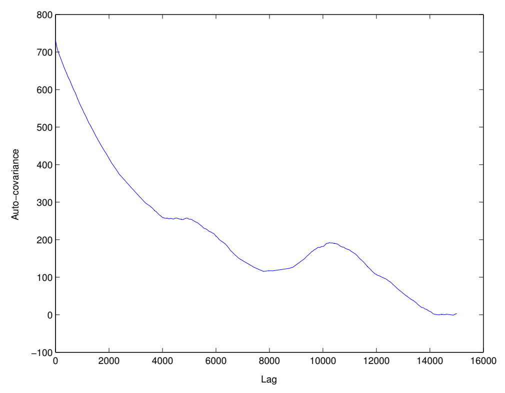


Figure 6.26: ACVF of the second context class trajectory

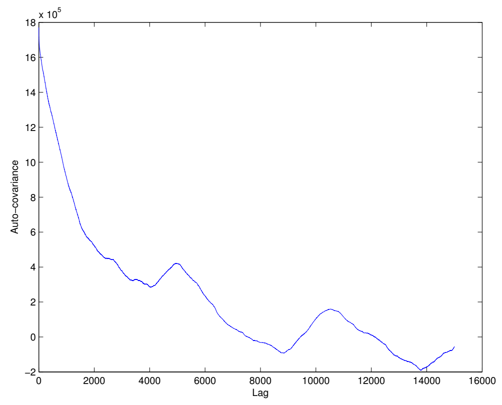
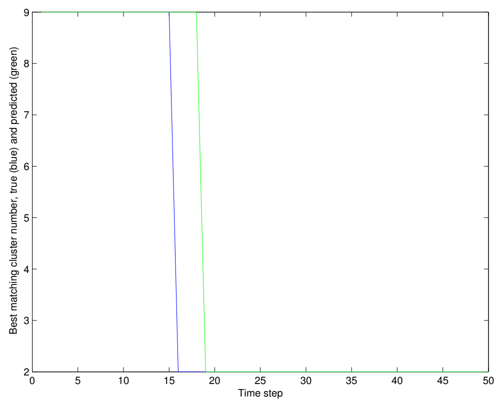
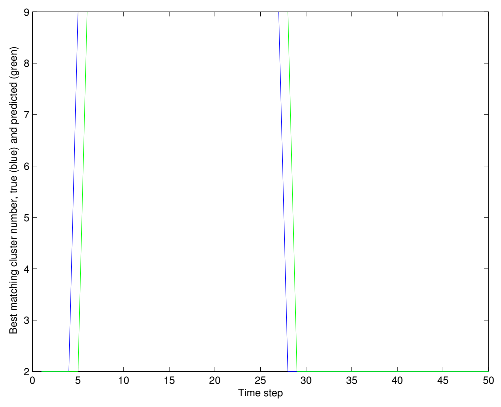


Figure 6.27: ACVF of the aggregated, categorical context class trajectory



(a)



(b)

Figure 6.28: Aggregated ARMA forecast showing the categorical cluster trajectory for 50 time steps, starting from step 50 (a) and 150 (b)

50-step predictions, explained in more detail in Section 6.4.4, are 0,06 starting from time step 50 and 0,04 starting from time step 150.

6.4.3.2 Backpropagation MLP

For evaluating the Multi-Layer Perceptron trained by backpropagation, the same time series is used as with the ARMA model for better comparability. Due to the resolution reduction by a factor of 100, roughly 900 data points of this time series were available. Out of these, the first 600 samples are used as the training set and the samples in the interval $[601;800]$ are used as the test set, With this partitioning, $2/3$ of the whole data set should provide a sufficient number of samples for training the network, while the remaining test set is still large enough to detect possible effects of overfitting. In practice, training set sizes between $2/3$ and $4/5$ of the whole set are used in different application areas and haven proven

Hidden units	MSE $[10^{-3}]$					Mean
2	50,4	14,7	16,1	12,5	23,8	23,5
3	9,6	11,0	12,5	25,1	10,6	**13,8**
4	10,6	8,7	21,2	22,7	19,6	16,6
5	9,3	10,5	30,6	24,7	25,2	20,1
10	18,6	41,7	46,5	25,3	37,9	34,0
15	52,2	51,5	43,0	24,5	42,0	42,6
20	43,8	37,4	44,5	36,3	21,5	36,7
25	43,8	50,2	36,4	53,6	60,0	48,8
30	36,8	54,9	62,9	47,1	61,9	52,7
40	64,0	74,8	52,4	58,7	68,4	63,7
50	47,6	53,5	63,6	37,4	69,6	54,3
70	88,4	72,2	70,9	77,8	62,9	74,4
100	87,3	97,7	89,0	89,4	101,3	92,9

Table 6.8: MLP training error depending on the number of hidden units

to be appropriate. The following evaluation was performed with the Neural Network Toolbox in Matlab by interactive construction of the network models and batch training.

The MLP is evaluated under the same conditions as ARMA to achieve comparability of the approaches, but, as an MLP is a general model for mappings from input to output vectors and not a specialized time series prediction technique, its applications is different. By applying a function of sigmoid type to the hidden units (the “tansig” function in the Neural Network Toolbox) and a linear function to the output units (“purelin”), an MLP becomes a general function approximator [HSW89]. However, such an universal function approximator only maps a multi-dimensional, numerical input vector to a multi-dimensional, numerical output vector. To apply it to time series forecast, the time series values must be transformed from time to state space via a sliding window approach, sometimes also called *sliding window method* [Die02]. When starting from a sample at time step t in the time series, samples in the past of t, i.e. lower indices, can be taken as the input vector and samples in the future of t, i.e. higher indices, can be taken as the corresponding output vector that is necessary for training of supervised methods. As apparent in Fig. 6.23 and already discovered with the seasonal fit of the ARMA model, the periodicity of the time series has a period length of more than 50 time steps with reduced resolution. If the MLP is to discover this periodicity, it is thus necessary to use an input vector that covers a larger time window to include at least one period in each input vector. Additionally, a MLP can produce multi-dimensional output, which allows to implement multi-step prediction in a direct, non-iterative manner. As a consequence, the sets of indices for input and output vector have been chosen as $\{-70,-60,-50,-40,-30,-20,-15,-10,-7,-5,-4,-3,-2,-1,0\}$ and $\{1,5,10,15,20,25,30,35,40,45,50\}$ respectively. For the input vector, the more recent history is included in more detail, while the indices for the output vector are equally spaced. All index values are relative to the current sample, which leads to the actual time window sliding over the samples $[71;600]$ for the training and $[671;750]$ for the test sets. In this setting, there are 530 combinations of input and corresponding output vectors in the training set and 80 in the test set.

MLP type networks require their structure to be defined a priori as only the parameters but not the structure can be determined by a training procedure. A standard MLP has an input layer, a hidden layer and an output layer. For the input and output layers, the number of neurons is fixed by the respective number of dimensions in the input and output vectors: in this setting 15 input and 11 output neurons. In [TF93], it has been suggested to use the same number of input units as there are AR terms in the corresponding ARMA model. However, since this would not consider the seasonal fit that is required for this data set, a larger number of inputs is necessary. An appropriate number of neurons in the hidden layer is often found by trial-and-error, e.g. also in [TF93]. Within the scope of this evaluation, an appropriate range of networks with different configurations are constructed and compared. The actual training is performed over 500 epochs each, where one epoch is defined as the presentation of all training samples to the network, with five test runs for each network topology. For assessing the suitability of a specific network configuration, the mean squared error (MSE) over the whole training set is used as a quality measure. As can be seen from Fig. 6.29, 500 epochs is — for this learning problem — an appropriate number, as the MSE does not further decrease after that. Table 6.8 shows the results of this network size selection. Because different training runs produce noticeably different results, the mean error over 5 test runs is minimized for selecting the best network size, resulting in a hidden layer size of 3 neurons. One specific training instance of this network structure with a MSE of 0,0106 was then used for cross-validation with the test set and yields a MSE of 0,0055 or 0,074 RMSE. When considering only the first output dimension, i.e. a prediction of the sample at one time step ahead, the MSE declines to 0,0028 or 0,053 RMSE. As indicated by the RMSE in comparison to the results of ARMA prediction, the MLP is in this form incapable of estimating the future context class trajectory in the test set. Fig. 6.30 shows the real trajectory in blue and the trajectory predicted by the MLP instance in single-step mode in green. At least for this scenario, the MLP approach does not seem suitable for context prediction due to its prediction accuracy, aside from the fact that it is also a batch training approach that is not easily applicable in an online way. This assessment is supported by the aggregation of the MLP predictions of all 9 context trajectories, each of which is generated by a distinct MLP with 3 hidden units and trained the same way as described above. For the MLP, 50 single-step predictions starting from time step 150 are performed due to the definition of the input vectors, which require time series values at 60 and 70 steps before the current one. Thus, starting at time step 50 would not allow to extract complete input vectors for training the network. Fig. 6.31a shows clearly that the predicted categorical time series differs completely from the real time series and the 0/1 loss is 1. It should be noted that the interval of time steps $[151;200]$, for which the 9 context class trajectories have been predicted, lies completely within the interval $[71;600]$ of the training set and the network has thus actually been trained on it.

So far, only the first network output has been used and the network has thus been restricted to its single-step forecast. When exploiting the multiple-output capabilities of the MLP and using not only the first output but all 11 for deriving the categorical time series, results are expected to be more favorite. Fig. 6.31b shows the concatenated predicted time series from only 5 prediction steps (instead of 50 single-step predictions) but using the multi-step predictions as defined in the output vector. The associated loss in this case is 0,62. It should be noted that the MLP predictions are generated with the inputs from the test set, which are extracted from the real time series. An iterative approach, where the values predicted by previous steps are used as the input, would match the procedure used by ARMA, but is expected to perform even worse. This means that for the MLP 50-step prediction, only every fifth value would be available after a single prediction step.

6.4.3.3 SVR

Support Vector Regression has proven to be highly successful for chaotic time series prediction. In this evaluation, libSVM [CL01] is applied with the approach described in [HCL03], i.e. the sliding

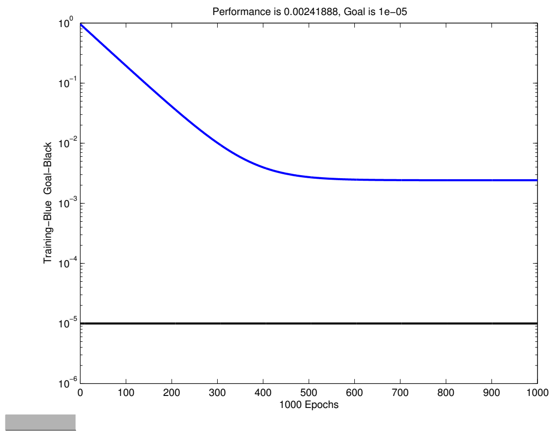


Figure 6.29: Effect of the number of training epochs on the MLP training error

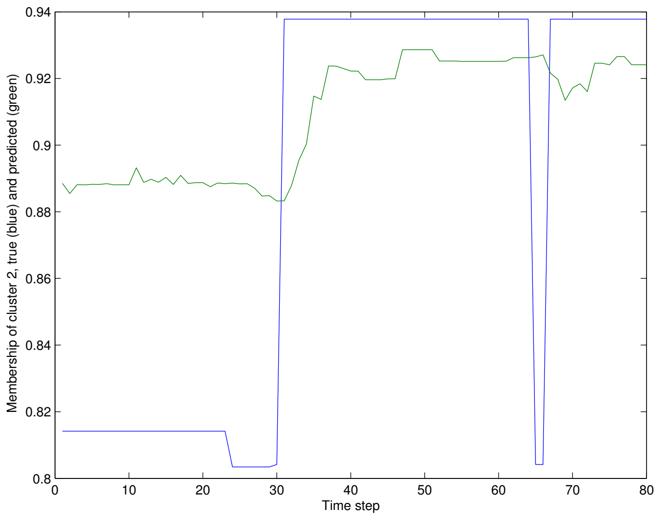
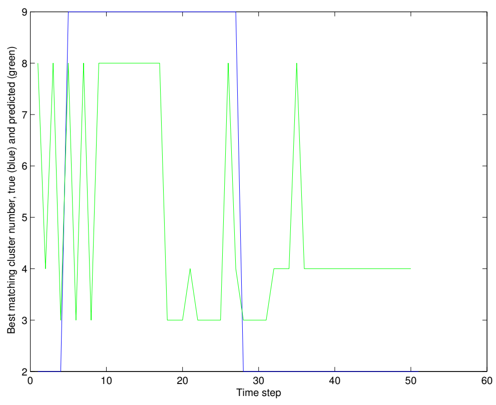
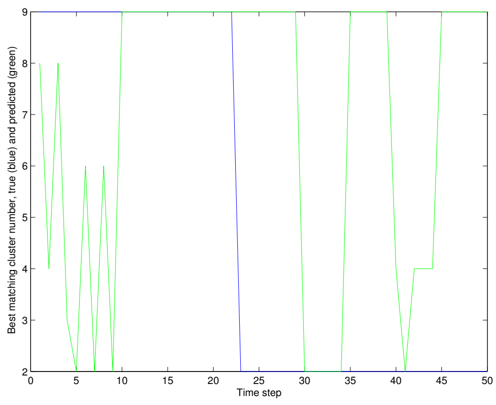


Figure 6.30: MLP forecast of the second context class trajectory

window method already described above for MLP prediction. As the minimal embedding dimension (cf. [MOG97]) for our problem is not known, it needs to be estimated based on a human inspection of the data set. Without additional knowledge and to achieve direct comparability, the same training and test data sets that were used for the MLP are also used for SVR prediction. Therefore, an embedding of 15 non-consecutive points is used for this data set, which lies between the embedding of 6 and 20 used in [MSR$^+$97]. With a resolution reduced by a factor of 100, 50 steps are predicted, starting from step 150. In contrast to the MLP, the SVR approach has only two free parameters with reasonable default values already set in libSVM. In [HCL03], it is suggested to perform an exhaustive grid search to find optimal values for these two parameters in the case that the regression accuracy is not sufficient. As shown in Table 6.9, SVR performs reasonably well in its default configuration: for the 50 single-step predictions on each context trajectory, a mean RMSE of 0,067 is achieved. Although this does not



(a) using only the first network output



(b) using all network outputs

Figure 6.31: Aggregated MLP forecast showing the categorical cluster trajectory for 50 time steps, starting from step 150

Context class trajectory	RMSE
1	0,0690
2	0,0949
3	0,0392
4	0,0240
5	0,0977
6	0,0892
7	0,0089
8	0,0894
9	0,0909
Mean	**0,0670**

Table 6.9: RMSE values of continuous SVM forecasts for 50 single-step predictions starting at step 150

compare favorably to the MLP approach, Figures 6.32 and 6.33 show that, at least for the 50 single-step predictions, SVR produces a usable result. The loss for the aggregated categorical context trajectory prediction is 0,24.

6.4.4 Categorical Time Series

For evaluating categorical time series prediction methods, the respective best matching context class at each time step is used as the time series value. Fig. 6.15 shows the class trajectory computed by LLGNG, which will be used as the basis for the following evaluations. As a quality measure for categorical time series prediction, a common loss function known as the *0/1 loss* is both simple and appropriate and has already been used for evaluating time series prediction methods (cf. [Die02]). For each incorrectly predicted context class, a loss of 1 is received, while a loss of 0 is assigned for every correctly predicted value. The overall loss is obtained by simply averaging over all predicted values:

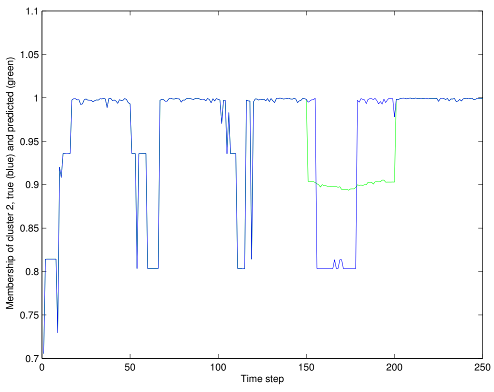


Figure 6.32: SVR forecast of the second context class trajectory

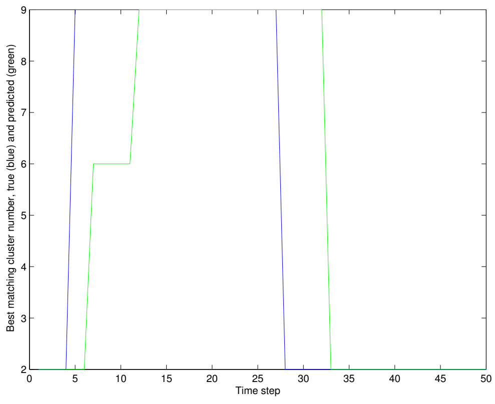


Figure 6.33: Aggregated SVR forecast showing the categorical cluster trajectory for 50 time steps, starting from step 150

$$l = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} 1 & if\, y_i \neq \bar{y}_i \\ 0 & otherwise \end{cases}$$

where y_i and $\bar{y}_i$ are the true and predicted values of the time series at time step i. As for the RMSE e, smaller values of l describe better prediction results.

This quality criterion is applicable to arbitrary categorical time series prediction methods. More complex loss functions have been developed (see [Die02]), but for time series prediction, every incorrect prediction should be considered equally. It might be possible to weight farther predictions lower than nearer ones to emphasize correct short-term prediction over longer-term prediction, but then the (exponential) decay parameter becomes of importance and determines the final prediction quality. For simplicity, the simple 0/1 loss is used in the following.

In contrast to the continuous time series prediction methods described in the last section, the full resolution of the (categorical) time series can be used, as the methods are able to cope with a data set of this size with reasonable computing time. As most of the following methods have already been implemented within the framework described in Chapter 5, they can be used directly in embedded systems.

The first three methods can be and consequently are used in online mode, which means that only a single, sequential pass is used over the whole data set; training and evaluation are both done in this single pass. At each time step, a prediction of the current value is computed and compared with the real value from the time series, enabling the assessment of the loss function. Only then, the real current value is fed into the prediction method for incrementally updating the internal model. This procedure implicates that at the beginning of the time series, the predictive model is just being constructed and the predictions are expected to be erroneous. By already including these initial transients in the overall loss function, those methods are obviously handicapped when compared to batch training methods, where the evaluation is performed after training has been completed and thus after the model has been constructed. On the other hand, it offers a more realistic assessment of the prediction accuracy when used in online settings. While not directly comparable to the loss values achieved by the other, batch-trained methods which are evaluated in the following, they give an upper bound on the prediction error. When using the online methods with batch training, which is always possible, the respective loss will typically be lower.

Window size	Loss (complete, single step)	Loss (discarding duration)	Loss (50 steps)
3	0,0203	0,9870	0,4600
5	0,0207	0,9712	0,4600
7	0,0224	0,9495	0,4600

Table 6.10: Influence of the sliding window size on the predictive accuracy of a categorical central tendency predictor

For the remaining two methods, the loss is computed over a test set — after batch-training the predictor with a training set. To allow the prediction method access to long-term trends and periodical patterns in the data set, $4/5$ of the data are used as a training set and the remaining fifth is used as a test set, corresponding to the time step intervals $[1;70138]$ and $[70139;87673]$ respectively.

6.4.4.1 Central Tendency Predictor

The central tendency predictor is used with a variable sliding window size and should be considered as the baseline in prediction accuracy. As it is used for categorical time series prediction, the arithmetical mean or median can not be computed. Instead, the context class with the maximum number of occurrences within the time window is used as the prediction for the next time step. Table 6.10 shows the loss values for the whole data set processed in single-pass, online mode, when discarding duration information, and for a 50-step prediction for different sizes of the sliding window. The values in the first column are obtained by counting erroneous predictions during a single run over the whole data set, while the values in the second column are obtained the same way, but discarding duration information. More precisely, consecutive sequences of equal classes are shortened to only a single time step. Therefore, by discarding duration information, no direct repetitions will occur and central tendency predictors are consequently not suited well. The loss values in the second column are shown for comparability with the following prediction method. Finally, the loss values given in the third column are obtained by “training” the predictor with only the first 150 time steps and then iteratively predicting the next 50 steps. For comparability with the continuous prediction methods, the same time frame is used and, only for this column, the resolution of the data set has also been reduced by a factor of 100. It is important to note that columns are not comparable with each other, but only with respective loss values from other methods.

The good results for online, single step prediction owe to the fact that the data set shows slow changes when processed in its full resolution, and a good assumption is therefore to continue with the last context class for the next time step. As apparent from the loss, this simple assumption is true for over 97% of the cases. For the 50-step iterative prediction at reduced resolution, the central tendency method performs far worse, because it is limited to predict a single context class, specifically the one that was the most frequent in the time window when the multi-step prediction started, indefinitely. An interesting detail is that the loss values are independent of the sliding window size, which suggests that the time series was stable at time step 150 (the starting point of the 50-step prediction) even at reduced resolution. As expected, the loss values for the time series with discarded duration information indicate complete unsuitability in this case, where a prediction of a single step corresponds to multiple steps in the uncompressed time series.

6.4.4.2 Active Lempel-Ziv

ALZ has no parameters that need to be set in advance but constructs its internal model, composed of Markov chains with varying order, in a completely unsupervised way. Moreover, it is strictly deterministic like the simple central tendency predictors and thus each (online) pass over the data set will produce the same result. A single, online pass over the whole data set leads to an overall loss of 0,0245, which is higher than the maximum loss caused by central tendency predictors and thus not acceptable. The loss for 50-step prediction at reduced resolution is 0,46, because ALZ incidentally predicts the same time series as the central tendency predictors (a single context class repeated for 50 steps). It is also unacceptable, given the added complexity of ALZ with no gain in predictive accuracy over the simpler central tendency predictors.
However, by discarding duration information as described for the central tendency predictor, the resulting loss of 0,3055 is noticeably higher than for the unprocessed time series, but compares very favorably to the results of the central tendency predictor. These results are to be expected: discarding the duration information results in a time series with the same resolution — in terms of the frequency of changes — but in a highly compressed form, containing only 1843 instead of 87673 samples. This implicates that a single predicted step in this compressed form corresponds to multiple steps in the uncompressed form (on average $87673/1843 \approx 48$ consecutive steps). Obviously, for comparing the predicted compressed trajectory with an uncompressed one, the duration that will be spent in predicted context classes also needs to be predicted. This can be performed with the duration predictor, which is discussed next.

6.4.4.3 Duration Predictor

The duration predictor currently complements ALZ prediction where duration information is explicitly discarded in the described setting. This separation of concern also enables both methods to be exchanged independently; one predicts the next context, the other one the duration until this predicted context is entered, i.e. the number of time steps until the next transition. While ALZ is currently used for the first task in this combination, a duration predictor for the second method that applies the principles described in Section 4.8.5 has been developed. For the present evaluation, it is trained in parallel to ALZ on the same time series, i.e. the first 150 steps with reduced resolution in online mode. The resulting loss is 0,44, improving the ALZ-only result only slightly and thus suggesting further research on the duration predictor.

6.4.4.4 HMM

Although currently incapable of online prediction with incremental model update, a HMM predictor has also been implemented within the framework and is used for this evaluation. As usual with HMMs, batch training is applied to learn the model parameters, in this case 4/5 of the categorical time series, and the remaining time series is used as a test set for cross validation. After parameter estimation with the standard Baum-Welch algorithm, single-step context prediction is performed in a straight-forward way, also used e.g. in [RC03]: for each of the possible next contexts classes, a sequence is created by appending the possible next class to a the sequence of the last n context classes. This new sequence is then tested against the HMM with the standard Viterbi algorithm to estimate the probability with which the HMM would create this sequence. After all possible next contexts have been iteratively rated, the context that produced the highest probability is taken as the prediction for the next time step. For multi-step prediction, this procedure is performed iteratively, appending the predicted context to the sequence while sliding the window forwards at each time step. The prediction accuracy will be higher when performing a direct multi-step prediction by evaluating multiple prediction steps at once, but the generation of appropriate test sequences for the Viterbi algorithm shows exponential behavior with regard to the number of time steps. Therefore, direct multi-step prediction with HMMs would — in contrast to ALZ — be possible, but is NP-hard and thus intractable for practical applications. For this evaluation, a sequence with a fixed length of 30 has been used.

One of the issues with HMMs is that the model structure, which includes the number of hidden states and connections between the states, has to be selected a priori and can not be determined by the standard learning method. For some problem domains, knowledge about the properties of the underlying process that generated the respective time series can be used by experts to explicitly create the model structure. However, for many practical problems, including the problem of context prediction, the number of hidden variables in the creating process is unknown and needs to be determined systematically. Various approaches have been proposed to select an appropriate number, e.g. [RC03] suggests that the number of hidden states has been chosen to be equal to the size of the output alphabet. For predictors within the architecture used in this thesis, this would mean to set the number of hidden states to the number of automatically found context classes. Another approach is to use model selection criteria like the Bayesian Information Criterion (BIC), which considers both the training error and the size of the network [LB00]. Given the lack of knowledge about the context trajectory for the general case, the number of hidden states k is selected by evaluating fully connected models with $k = 2 \dots 30$. The number of steps necessary for a convergence of the Baum-Welch training and the loss value obtained by single-step cross-validation with the whole test set is presented in Table 6.11. As can be seen, the HMMs with 16, 17, 18, 19, 24 and 25 hidden states perform similarly well for the single-step prediction. The second group with 24 and 25 hidden states does not perform better than the first group with 16 to 19 states, but already consumes noticeably more processing power and has therefore not been used. In the first group, the differences in the loss values are marginal, so any number of hidden states from 16 to 19 is a reasonable choice. Because there are nine different symbols in the time series, 18 hidden states seem plausible (when the underlying model has pairs of states with similar observation symbol probability distributions) and consequently this number of hidden states is selected for the computationally more intensive multi-step prediction. However, it should be noted that the best HMM with its loss of 0,0534 still performs worse than the central tendency and ALZ predictors for the single-step prediction case.

For the 50-step prediction starting at time step 150 with reduced resolution, the HMM predictor also achieves a loss of 0,46 by predicting again the same time series as the central tendency and ALZ predictors. Considering the successes of HMMs in other application areas, this result seems unreasonable. Therefore, further experiments are performed to improve the accuracy of the HMM predictor by increasing the size of the training set, whose initial 150 samples might be too small for the HMM approach. Interestingly, the HMM with 18 hidden states can not improve its performance even when the training set includes the complete prediction period, but produces a 50-step prediction loss of 1,0 when trained with over 600 samples of the time series with reduced resolution, i.e. with roughly 2/3 of the whole data set. Neither changing the number of hidden states nor the size of the sliding window, which was performed in a full grid search from $k = 2 \dots 40$ hidden states and $l = 1 \dots 150$ time steps in the sliding window, can improve the loss any further.

Hidden states	Steps to convergence	Loss
2	7	1,0000
3	12	0,8485
4	10	0,5332
5	17	0,3987
6	9	0,5317
7	29	0,2371
8	20	0,2480
9	17	0,2474
10	14	0,0662
11	22	0,0652
12	24	0,2367
13	93	0,2781
14	19	0,2371
15	23	0,2371
16	57	0,0552
17	54	0,0535
18	64	0,0550
19	119	0,0546
20	64	0,2378
21	107	0,3056
22	16	0,5227
23	92	0,1782
24	57	0,0534
25	27	0,0538
26	27	0,2368
27	54	0,0655
28	93	0,3052
29	42	0,2639
30	17	0,2366

Table 6.11: Influence of the number of hidden states on the predictive accuracy of a HMM

6.4.4.5 SVM

Support Vector Machines solve, in contrast to SVR, a classification problem. Although initially developed for two-class classification, there exist methods for achieving a multi-class classification by combining multiple two-class SVM classifiers. Such a method is implemented in libSVM and can be used for categorical time series prediction by mapping the time to a state space in a similar way as done for continuous MLP and SVR predictions. For comparability, the same sliding window method with identical embedding is used, but on the aggregated categorical time series instead of the individual continuous time series. This accounts for a training set of 530 samples with an embedding of 15 and a test set of 50 samples with the same embedding. In this case, the loss for 50 single-step predictions is 0,04, i.e. only two samples from the test set were erroneously predicted. However, when using an iterative procedure to compute a single 50-step prediction by using the predicted values as input for the next prediction steps, the loss is increased to 0,46, as the same (constant) trajectory is predicted that is also generated by the central tendency predictors and the unmodified ALZ.

6.5 Summary and Discussion

The evaluation presented in this chapter is a proof-of-concept that context recognition and prediction are possible with the developed architecture. For the intended scenario described in Section 6.1, context prediction with a forecasting horizon of a few hours allows to provide additional benefit to the user. In this evaluation, the methods were compared for a 50-step prediction at reduced resolution with a sample width of $30 \cdot 100 = 3000$ seconds, yielding a prediction horizon of 150000 seconds or $41,\overline{6}$ hours. This exceeds the time frame necessary for achieving a first user-visible benefit of context prediction and allows applications to exploit predictions for horizons of over a day. Although this does not tackle long-term prediction in the area of weeks or months, it is already challenging and most evaluated time series prediction methods perform poorly in this setting. Another complication in the considered scenario is that the data set features missing values where no samples are available. For short-term prediction of horizons under a day, this is not an issue because these non-consecutive areas are mostly consistent with the sensing device being switched off during the night. However, mid-term and long-term predictions are seriously handicapped by missing values, as apparent in this evaluation. Any improvement in prediction accuracy most probably requires to explicitly consider the missing values in the predictor. Initial transients are also an issue for prediction methods. Although there are no initial transients in the data set itself, because no special considerations were taken during the experiment, online prediction methods will suffer from the fact that their internal models are still being constructed while predictions already need to be performed. In many real-life situations, such a “training” phase will be acceptable to users. Therefore, initial transients have not been considered in this evaluation.

To allow the two main steps in the architecture, namely classification and prediction, to be assessed independently, the quantitative evaluation and comparison of different methods has been split into two parts. For context recognition, the final classification error has been defined in Section 6.3.1 and is used for comparing three different unsupervised classification methods. Table 6.12 shows that the extended LLGNG classification algorithm described in more detail in Section 4.6.4 compares favorably to the k-means and SOM methods on this data set. This supports the results of the qualitative comparison presented in Section 4.6.3. The k-means algorithm showed the highest classification error and visual inspection of the computed context class trajectories was also not satisfying. For the SOM, a visualization of the one-hour extract of the data set looks plausible, but the one-day and one-week extracts show strong oscillations, which are unsuitable for context classes. Moreover, the final classification error on the whole data set is only slightly lower than the error produced by k-means. LLGNG achieves a significantly lower classification error on the whole set, even though it is used in online mode which is more challenging. It also produces the lowest mean classification error for the individual weeks, although the difference to the SOM is not as pronounced as for the whole data set. A visual inspection of the generated context trajectories shows more stability with fewer oscillations. Summarizing, LLGNG is clearly the most favorite of the evaluated classification algorithms for this setting. Its properties, analyzed in Section 4.6.3, are also promising for other settings in the area of context recognition.

For context prediction, the results of this evaluation are not as clear. The prediction evaluation is based on the context trajectory computed by LLGNG in the previous step and uses the 0/1 loss for quantitatively comparing different methods. This criterion has the advantage that it can be used for both continuous and categorical prediction. Although the online single-step prediction results can give some hint on how well a specific approach adapted to the time series, for practical applications the multi-step prediction will be more important because of the involved time frame. In this experiment, the sample rate was chosen to be 30 seconds; a single-step prediction would therefore only cover this period. Table 6.13

Algorithm	Number of clusters	Final classification error (whole set)	Final classification error (weeks mean)
K-means	6	0.7451	0,6469
SOM	1491	0.5659	0,1342
Extended LLGNG	103	**0.0069**	**0,1178**

Table 6.12: Summary of the classification algorithm evaluation

Algorithm	0/1 loss
ARMA	**0,04**
MLP	0,62 1)
SVR	0,24 1)
Central tendency	0,46
ALZ	0,46
ALZ + duration	0,44
HMM	0,46
SVM	0,46

1) for 50 single-step predictions instead of a single 50-step prediction

Table 6.13: Summary of the prediction algorithm evaluation

shows the result values of a 50-step prediction starting from time step 150. It should be noted that those steps refer to a context trajectory with a resolution reduced by a factor of 100 to make it manageable for all prediction methods. Therefore, a single prediction step represents almost an hour. Surprisingly, ARMA performed very well for this data set and outperformed all other methods in terms of the 0/1 loss. The expected reason for this success is the seasonal correction used in the pre-processing step of the ARMA approach. It is though difficult to perform this correction automatically in an online way, no solution is yet implemented for estimating the lag from an incrementally computed ACF as it is available for context prediction data sets. For categorical time series prediction, ALZ in combination with the duration predictor performed best, but nonetheless only slightly better than the other methods and still unacceptable in terms of accuracy. Generally, the loss values of the continuous and categorical methods are not completely comparable because for the continuous prediction methods and SVM, a larger part of the data set was used for training, which actually contained the 50 steps that were subsequently predicted. For the other methods, only the first 150 steps could be used for training. The reason is that the online methods (as implemented in the form of modules within the framework) keep an internal state that would need to be reset if they were first trained with the full data set but then applied to a sequence that was already presented to the algorithm. This reset can not be done in a way independent of the algorithm internals and would not represent the normal operation mode of those predictors. Thus, batch training algorithms were given a better chance on predicting this specific sequence. This should be considered when interpreting the results summarized in Table 6.13 as a preparation for other applications or data sets.

In this evaluation, the results of the MLP are most probably worse than they could be. Under certain conditions, MLP networks have been shown to outperform ARMA methods, e.g. in [TF93], but in all

of those cases, careful pre-processing and selection of network structure was the result of a supervised, expert-driven process. The purpose of this evaluation is to show the applicability of different prediction methods without significant human intervention, mostly in an unsupervised manner. MLPs can perform very well, but they need to be carefully applied to do so. ARMA, on the other hand, performed surprisingly well without especial intervention. Although it could not be applied directly to context prediction in information appliances in the same way as described above, mainly due to the seasonal correction, it seems more reasonable to derive automatic model selection procedures for ARMA than for MLPs.
The HMM predictor also performed worse than it would have been expected and the two most likely reasons for this poor performance are that the number of training samples was too low for this type of statistical model and that the intrinsic exponential distribution of state durations in the standard formulation of HMMs was not compatible with the longer sub-sequences of similar states where no state changes occur. VDHMMs or other models address the latter issue and are subject to future research. Another probable explanation is that a first-order Markov model is simply not appropriate for modelling this time series, as suggested in Section 4.8.5. The fact that the ARMA model performed far better than all other models suggests that the explicit seasonal correction, which was only used by the ARMA approach, was the key to its success. It is still an open issue how such a seasonal correction could be applied to categorical time series prediction or how the lag can be found automatically for ARMA models in an online, unsupervised way. The Episode Discovery algorithm might be a promising candidate, but not enough details have been published to reproduce the results reported in [HC03]. Another open problem area is the topic of missing values: It is currently only dealt with in the classification step, but explicitly taking missing values in the data set into account in the prediction step is also an issue for future research on context prediction.
The majority of the categorical prediction methods have the significant advantage that they are online methods and can cope with large data sets. Some of the methods, more specifically ARMA, the central tendency predictors, ALZ, the duration predictor, SVR and SVM are strictly deterministic and therefore are not dependent on some random initialization. The advantage that they do not converge against local maxima should not be underestimated, as in an online setting, multiple model instances can not be tried to find the best one.

It is important to note that the results presented in this chapter are *not* universally valid. A rigorous analysis of multiple different data sets and a quantitative comparison of the presented classification and prediction methods based on these are outside the scope of this thesis, whose main aim is to develop an architecture for context prediction independently of specific application areas and scenarios. It is subject to future, more specific studies to set up and perform multiple experiments for collecting sensor data with a controlled level of variability in the settings. Nonetheless, the presented initial evaluation based on a single but extensive data set shows the general applicability of the developed architecture by successfully predicting future contexts in the time frame of multiple hours, even if the prediction accuracy leaves room for improvement. The quantitative comparison of different methods also shows how they can be used for context prediction and therefore eases their application to other data sets.

Chapter 7

Conclusion

Pervasive computing strives towards a future world of interconnected, embedded and smart devices dispersed over and embedded into the environment. In this vision, artifacts of the physical world act as an implicit interface to the virtual one, blurring the borders between them. The possibilities that emerge from computer systems being available everywhere, everytime and to everybody can currently hardly be estimated; neither can the risks. A key issue in a world of pervasive computing, where devices communicate and interact seamlessly and autonomously with each other is to simplify user interaction. Technological advances always have an impact on the daily lives of people who are not involved with the scientific or technical field that enabled these advances. A logical consequence is that interfaces to new technologies with possibly large impact must be designed in a way so as to be suitable for the majority of their intended user base.

For the vision of pervasive computing with accustomed objects of the physical world as its proclaimed interface to new, hidden functionality, this demands user interfaces that are intuitive and efficient. One prerequisite for such interfaces is that the devices must be aware of their environment, and that they must adapt to the context in which they are used. The term *context awareness* has been established as the property of computer systems to be aware of and adapt to context, and the associated research field of *context computing* is currently a highly active one. The main focus of the present thesis is on a branch thereof, namely *context prediction*. Context prediction aims at inferring future contexts from past (observed) contexts. A device that is able to predict future contexts in addition to capturing current and remembering past contexts enables the development of applications which offer additional functionality by proactively assisting the user.

In this thesis, the possibilities for predicting context on an abstract level have been examined, an architecture for context prediction has been developed and different methods for context prediction within this architecture have been evaluated qualitatively as well quantitatively. This last chapter shall add some concluding remarks to the present thesis by summarizing it, giving a critical evaluation and providing pointers for future research.

7.1 Summary

The problem statement of this thesis, and subsequently the main focus, was to find concepts, architectures and methods for context prediction. Along with specific challenges and a definition of the scope of the present work, this has been more closely described in Chapter 1.

Context prediction generally builds upon two concepts introduced in earlier work: context computing — more specifically awareness of past and current contexts — and proactivity, known from other research fields like software agents. In Chapter 2, these foundations have been introduced and the notion of context has been examined, including a general motivation for the use of context in computer systems and a discussion of various definitions and different aspects of context. Because the use of proactivity in combination with abstract context descriptions is a new issue in context computing research, a first taxonomy of four application areas that can benefit from context prediction is given: *reconfiguration*, *accident prevention*, *alerting* and *planning aid*.

Since context computing is currently an active research field, many publications deal with topics around the general area of context. In Chapter 3, the most closely related projects are listed and the present thesis is positioned among and against other publications. The summarizing table of characteristics of the specific projects shows that the concept of predicting context on the level of abstract identifiers has, to the best of our knowledge, not been covered before by earlier published work.

The developed architecture for context prediction is presented in the main part in Chapter 4 and allows online, unsupervised context recognition and prediction from a multitude of different sensors. Designed as a multi-layer architecture, it has five steps which are coupled via simple interfaces: *sensor data acquisition* (to gather raw data from sensing devices), *feature extraction* (to generate a more relevant representation of sensor data, exploiting domain-specific knowledge), *classification* (to find similarities and common patterns in the input data), *labeling* (to assign simple context labels to recognized classes), and *prediction* (to forecast future contexts based on past behaviors). With the exception of the feature extraction step, no knowledge about the application domain is incorporated into the internal models; the developed architecture intentionally operates on a syntactic instead of a semantic level. This has the advantage that the system is independent of the application area, but the disadvantage that semantic interrelationships between contexts are disregarded.
Due to the consistent definition of simple interfaces between adjacent steps, the architecture achieves a clear separation of concerns, supporting the development and evaluation of methods for context recognition and prediction independently of each other. Because the architecture is based on a filtering approach where data flows only in one direction, this additionally facilitates its application in embedded systems with limited resources and addresses privacy issues. One distinctive feature of this architecture is that heterogeneous features extracted from various sensors can be used directly and without a specific coding in a common classification step for context recognition; by directly incorporating non-numerical features, all information gathered from the available sensors is utilized without a typically deteriorative mapping from non-numerical values to numerical ones. Abstracting arbitrary features to a set of operations that need to be available for each one enables the classification step to be performed directly on a heterogeneous input space. For the majority of classification algorithms, only two operations are necessary on each dimension of the input space, i.e. on each element of the feature vector that spans the input space: a similarity measure, i.e. a distance metric, and an adaptation operator. Both operations have been defined formally and in terms of their characteristics, and examples for various types of features have been presented. By implementing these operations for all used features, a majority of the well-known

classification methods can be used directly within the architecture. To ease the selection of appropriate candidates, a list of requirements has been compiled and a selection of algorithms has been evaluated qualitatively based on an extensive literature survey. One of the considered algorithms, namely the Life-long Growing Neural Gas (LLGNG) classifier has been designed explicitly for online classification over arbitrary long periods and is thus well suited for the purpose of context recognition. Within the scope of the evaluation of classification algorithms, its basic methodology has been examined more closely. As an adaptation towards context recognition, properties of the internal model are used to derive high-level context identifiers, which have been termed *meta clusters* in this thesis. Similarly to the classification step, a literature survey on time series prediction methods also results in the compilation of a list of requirements and a subsequent evaluation of a set of algorithms based on these requirements.

To assess the applicability of the chosen approach and to allow for the development of applications that utilize context prediction, the concepts developed in terms of the architecture have been implemented as a flexible, cross-platform software framework. The flexibility of the multi-layer architecture coupled by simple interfaces is realized by a modularized implementation with loadable modules for the different steps. A selection of feature extractor, classifier and predictor modules has already been implemented, as presented briefly in Chapter 5 and in more detail in an accompanying diploma thesis [Rad04].

As an initial proof-of-concept of the presented architecture, a quantitative evaluation based on an extensive real-world data set has been shown in Chapter 6. The used data set was collected over a period of about two months, includes highly heterogeneous feature samples for different situations, and is thus challenging for context prediction. Following the general aim of unobtrusiveness, user behavior was merely monitored but the user was not actively participating in the experiment, e.g. by periodically entering the current context. Consequently, the data has been collected in a completely unsupervised manner, reflecting the intended way of integrating context recognition and prediction into embedded systems. A comparison of the extended LLGNG algorithm against the standard clustering methods k-means and SOM shows that it outperforms the other methods with regard to classification error after training. The quantitative evaluation of different prediction methods shows that context prediction for prediction horizons of a few tens of time steps is possible, but the currently achieved prediction accuracy suggests further research.

Assumptions

The developed architecture and its implementation are based on a few general assumptions about the problem domain:

- Memory and computational resources are sparse, but not limited to some specific value. More specifically, the architecture has been designed to use resources economically, but no eviction policies for coping with a fixed amount of memory or prioritization methods for managing concurrent threads have been considered.
- There are no defined constraints concerning real-time operation.
- Other temporal constraints, e.g. when specific sensors can only be queried within defined intervals, are not considered.
- When sensors are not directly connected but queried over some communication link like Bluetooth or WLAN, the traffic volume is not limited.

- Missing values in sensor data acquisition are explicitly handled, while faulty values are not. The difference is that missing values can be detected by the sensing process, e.g. when a sensor does not respond, but detection of faulty values necessitates an explicit error model specific to each sensor.

Most of the presented considerations are valid only within these assumptions.

7.2 Contribution

The main contribution of this work can be seen in three interrelated areas:

- The development of a general architecture for context prediction which features:
  - Flexibility due to simple interfaces between multiple well-defined steps: This makes the architecture suitable for arbitrary kinds of sensors and a wide range of appliances.
  - Clear separation of concerns: Implementations of feature extractors, classifiers, predictors and user interaction are completely independent of each other and thus easily exchangeable.
  - Coping with heterogeneity: The architecture has been designed to cope with heterogeneity in sensor and feature data and apply a common classification step without necessitating a mapping to numerical values.
- A systematic survey of classification and prediction algorithms for finding suitable candidates. This includes the compilation of requirements on classification and prediction methods to support a methodical, qualitative evaluation.
- The implementation of this architecture and the selected classification and prediction methods with the following properties:
  - Use of heterogeneous feature vectors for context recognition: Within the framework, the concept to cope with heterogeneous sensor and feature vectors has been implemented; feature vectors do not need to be mapped to numerical values at any stage. The classification step deals directly with arbitrary types of features and thus works on an input space composed of heterogeneous dimensions.
  - The implemented LLGNG classification module has been extended by the concept of meta clusters to support the direct generation of high-level context identifiers.
  - Development of a cross-platform framework suitable for devices with limited resources: No framework for context recognition *and* prediction is currently available that is aimed explicitly at local operation on such devices.

This thesis is neither focused on hardware and sensing technology, nor is it about user interface design and applications, nor does it develop new concepts for classification or prediction of data sets. It is an integrative work with the aim of combining context awareness with prediction techniques, and therefore describes a general architecture.

7.3 Critical Evaluation

An important topic of pervasive computing in general and especially for context computing is privacy. As briefly discussed in Section 4.2, user acceptance is and will remain an issue for current and near-term applications within pervasive computing, mainly due to privacy concerns. This thesis does not explicitly address these concerns either, which is a possible point for critique. Although the architecture has been designed in a way that allows context prediction without storing a complete log of sensor data or the past context trajectory and thus allows to design applications that adhere to high privacy standards, it does not provide any means for enforcing a privacy policy upon every applied method and algorithm over all steps.

The chosen approach to context recognition is a simple one, which tries to extract as much high-level context information from low-level sensor data as possible with a severely limited knowledge about the application area. By using additional constraints, it will most probably always be possible to find a better and more accurate way for detecting the current context, given an in-detail knowledge about the specific application area. Owing to the general aim of unobtrusiveness in all phases of device usage, context recognition as presented in this thesis is limited to the syntactic level. By intentionally disregarding domain-specific knowledge, semantic relationships between contexts can not be exploited. It could be argued that context awareness is necessarily dependent on such knowledge, but this thesis explores the possibilities of a completely syntactic approach. This confinement allowed the development of an unobtrusive, unsupervised and online approach, which has the substantial advantages that it enforces independence of the application area and is thus applicable to arbitrary ones and that it does not necessitate a distinction between training and operational phases of context-aware systems. Nonetheless, for many applications an approach on a semantic level might be more appropriate.

However, the general approach and subsequently derived methods for context prediction that have been presented in this thesis are applicable to arbitrary application areas and can thus be applied on top of other means of inferring the current context. As long as high-level context is identified by some concept that can be regarded as that state of a user, the methodology is applicable.

There are a few problem areas associated with context recognition on mobile devices. Besides the limitations defined by the general assumptions mentioned earlier, the problem area of context changes has not yet been explored explicitly. Slow changes in user behavior, usually termed trends, are partially solved by the classification step with its requirement of adaptivity. Sudden changes, which can be further distinguished as permanent and temporary, are also partially addressed by the required variable topology in the classification step: when a sudden change in sensor readings occurs, either a previously learned context class is reactivated or a new one is created. For the prediction step, slow or sudden changes in the user behavior have currently not been considered, neither for temporal nor permanent changes. The architecture for context prediction is not influenced by these issues, but methods used in the prediction step should consider them.

In the opinion of the author, the weakest point in this thesis is the data set used as the basis for the evaluation in Chapter 6. Conclusions drawn from this evaluation suggest a more complex experimental design: by considering a set of different applications, multiple different test subjects acting as users of the applications, different sets of factors like time and place, and most importantly, recording multiple instances of each combination of these variables, more generally valid conclusions could be drawn. For assessing the context classification and prediction accuracies of different methods and algorithms, application-independent experiments will be necessary to infer recommendations of specific ones. The results of the evaluation presented in this thesis give indications on the suitability of various methods, but for a general benchmark, a controlled level of variability in the data sets used for an evaluation is necessary. Efforts are ongoing to develop and publish a publicly available, extensive data set explicitly designed for benchmarking purposes, and the repository for context data sets mentioned in Section 5.5.2 is also expected to strongly promote this aim.

7.4 Outlook

Today, there are no "killer applications" for context computing, and it is debatable if a quest for such applications is indeed a worthwhile one. Context computing is one of the building blocks of pervasive computing and can be seen as a tool for application designers to assist in the development of more user-friendly computer systems, allowing a broader public to use computing services as part of their everyday life. Context prediction is a more special and very powerful tool, but it is complex to implement. Only when it is available in ready-to-use software libraries and frameworks, application designers can effectively start to use it as a tool for solving their problems. The architecture developed in this thesis has already been implemented as a framework to allow the easy development of applications that use context prediction, but a few issues remain open for future work:

- Only a first set of prediction modules has been implemented and the initial evaluation showed that a wider range of methods should be available to application designers. Additional methods need to be implemented as modules for the framework.
- Logging of feature streams and recognized contexts and complete replaying of feature streams is currently supported locally at each device, but accessing a central context data repository has not yet been considered.
- The framework has been designed to enable an autonomous operation on resource limited devices, and distributed execution on multiple networked devices is currently not supported. An integration with the Context Toolkit, which is more flexible but not as light-weight, could offer the advantages of both approaches.
- Event features as explained in Section 4.1.1 are currently not supported explicitly but need to be converted to data streams by the respective feature extractors. Nonetheless, a direct support could improve results for certain types of sensing technology.

Outside the scope of the specific implementation in terms of the presented software framework, some ideas for future research on the architecture or the general approach to context prediction arise:

- An inclusion of domain-specific knowledge into the context model could improve the results in a few areas: it can allow more accurate recognition of current context due to reduced ambiguities, it can provide a more powerful query interface for applications, and it could assist in the context prediction step by providing hints for future occurrences of contexts. All of these advantages emerge from the additional information that is embodied in interrelationships between contexts. An approach to combine the benefits of online, unsupervised context recognition as considered in this thesis with a semantic context model that inherently depends on domain-specific knowledge still needs to be devised.
- To improve the accuracy of prediction results, an automatic, online assessment of periodic patterns within the context trajectories is necessary. It is also important to deal with multiple periodicities

of different period length and phase. A new algorithm for detecting periodic patterns with unknown period with a single pass over the time series has recently been presented [EAE04] and its applicability to context prediction should be evaluated in detail.

- To enable the development and implementation of different user interface options for the labeling step, rigorous research on HCI issues will be necessary, including studies on efficiency, distraction and user acceptance.
- An integration of off-the-shelf sensors with off-the-shelf devices can support broader empirical evaluations with a large number of test subjects. When context recognition and prediction is incorporated into consumer goods, no special considerations will need to be taken for their use. An example could be the use of Bluetooth-enabled bio sensors — which might be embedded into wrist watches or body-worn jewelry — in conjunction with Symbian OS based smart phones.
- Finally, an evaluation with a controlled level of variability in the data sets and multiple test subjects will be necessary to derive more general statements about the context recognition and prediction accuracies achieved with different methods. For recording long-term data sets, the generation of appropriate data sets is difficult due to the human resources necessary for such a study.

Context prediction is a young topic, still at the outset of methodical research. When applied in a way that still leaves users in the loop of control (cf. [BD03]), it can be a powerful tool to support users in their daily lives and to foster a broad availability of computing services to a larger public. With intuitive and context-aware user interfaces and continuous prediction of future, expected contexts, computer systems can virtually disappear and let users again concentrate on their main tasks instead of becoming an end in itself. The present thesis adds a further step towards this ambition, but many more still need to be done. However, the social implications of pervasive computing must not be neglected; not only technological, but more importantly non-technological issues like a felt loss of control will rather sooner than later become urgent concerns. As it is neither possible nor desirable to slow down or prevent technological advances, possible future issues of developments with far-reaching impact must necessarily be tackled now, technologically, socially and legally.

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ART	Adaptive Resonance Theory
ASCII	American Standard Code for Information Interchange
CMU	Carneggie Mellon University
CSCW	Computer Supported Cooperative Work
DBN	Dynamic Bayes Network
DSP	Digital Signal Processor
DTD	Document Type Definition
FCM	Fuzzy C-Means
FOLDOC	Free On-line Dictionary of Computing
GNG	Growing Neural Gas
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Groupe Speciale Mobile (Global System for Mobile Communications)
HCI	Human/Computer Interaction
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JNI	Java Native Interface
JVM	Java Virtual Machine
NG	Neural Gas
OOP	Object Oriented Programming

ORL	Olivetti Research Ltd.
OWL	Web Ontology Language
PAN	Personal Area Network
PARC	Palo Alto Research Center
PDA	Personal Digital Assistant
PIM	Personal Information Manager
RBF	Radial Basis Function
RFID	Radio Frequency Identification
RPC	Remote Procedure Call
SGML	Standard Generalized Markup Language
SMS	Short Message Service
SNN	Spiking Neural Network
SOM	Self Organizing Map
SVM	Support Vector Machine
SVR	Support Vector Regression
VQ	Vector Quantization
WLAN	Wireless Local Area Network
WSDL	Web Service Definition Language
XML	Extensible Markup Language

Bibliography

[ABKS99] M. Ankerst, M. M. Beunig, H.-P. Kriegel, and J. Sander, *OPTICS: Ordering points to identify the clustering structure*, Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, pp. 49–60.

[ABM$^+$03] B. Apolloni, A. Brega, D. Malchiodi, G. Palmas, and A. M. Zanaboni, *Learning rule representations from boolean data*, Proceedings of the International Conference on Artificial Neural Networks (ICANN'03), 2003.

[AC03] C. Angulo and A. Català, *Online learning with kernels for smart adaptive systems: A review*, Proceedings of the European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (EUNITE'03), July 2003.

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, *Automatic subspace clustering of high dimensional data for data mining applications*, Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, 1998, pp. 94–105.

[ARS98] K. Alsabti, S. Ranka, and V. Singh, *An efficient k-means clustering algorithm*, Proceedings of IPPS: 11th International Parallel Processing Symposium, IEEE Computer Society Press, 1998.

[BBC97] P. J. Brown, J. D. Bovey, and X. Chen, *Context-aware applications: from the laboratory to the marketplace*, IEEE Personal Communications **4** (1997), no. 5, 58–64.

[BBHS03] M. Bauer, C. Becker, J. Hähner, and G. Schiele, *Contextcube - providing context information ubiquitously*, Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCS 2003) (Los Alamitos, California, USA), Universität Stuttgart, IEEE Computer Society Press, May 2003, pp. 308–313 (english).

[BBL$^+$00] P. Brown, W. Burleson, M. Lamming, O.-W. Rahlff, G. Romano, J. Scholtz, and D. Snowdon, *Context-awareness: Some compelling applications*, Proceedings of the CH12000 Workshop on The What, Who, Where, When, Why and How of Context-Awareness, April 2000, available at `http://www.dcs.ex.ac.uk/~pjbrown/papers/acm.html`.

[BD02] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*, second ed., Springer Texts in Statistics, Springer, 2002.

[BD03] L. Barkhuus and A. Dey, *Is context-aware computing taking control away from the user? Three levels of interactivity examined*, Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp'03) (A. Dey, A. Schmidt, and J. McCarthy, eds.), Lecture Notes in Computer Science, vol. 2864, Springer, October 2003, pp. 149–156.

[BFB94] S. B. H. Bruder, M. Farooq, and M. M. Bayoumi, *A feature-level approach to multisensor integration; emphasizing robotic applications*, Proceedings of the 1994 IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems, October 1994, pp. 477–484.

[BGS01] M. Beigl, H.-W. Gellersen, and A. Schmidt, *Mediacups: experience with design and use of computer-augmented everyday artifacts*, Computer Networks (Amsterdam, Netherlands) **35** (2001), no. 4, 401–409.

[BHMJ97] H. W. P. Beadle, B. Harper, G. Q. Maguire, and J. Judge, *Location aware mobile computing*, Proceedings of the IEEE/IEE International Conference on Telecommunications (ICT'97), April 1997, pp. 1319–1324.

[BJ02] P. J. Brown and G. J. F. Jones, *Exploiting contextual change in context-aware retrieval*, Proceedings of the 2002 ACM Symposium on Applied Computing, ACM Press, 2002, pp. 650–656.

[BKP02] S. M. Bohte, J. N. Kok, and H. La Poutré, *Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks*, IEEE Transactions on Neural Networks **13** (2002), no. 2, 1–10.

[BOP96] M. Brand, N. Oliver, and A. Pentland, *Coupled hidden markov models for complex action recognition*, Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), June 1996, pp. 994–999.

[BOST04] N. B. Bharatula, S. Ossevoort, M. Stäger, and G. Tröster, *Towards wearable autonomous microsystems*, Proceedings of the 2nd International Conference on Pervasive Computing (PERVASIVE 2004) (A. Ferscha and F. Mattern, eds.), Lecture Notes in Computer Science, vol. 3001, Springer, April 2004, pp. 225–237.

[Bov00] A. Bovik, *Handbook of image and video processing*, Academic Press, 2000.

[BR02] B. Brown and R. Randell, *Building a context sensitive telephone: Some hopes and pitfalls for context sensitive computing*, Tech. Report Equator-02-004, Equator, 2002.

[Bru99] S. B. H. Bruder, *An information centric approach to heterogeneous multi-sensor integration for robotic applications*, Robotics and Autonomous Systems **26** (1999), no. 4, 255–280.

[BS95] J. Bruske and G. Sommer, *Dynamic cell structure learns perfectly topology preserving map*, Neural Computation **7** (1995), 845–865.

[BTK$^+$03] T. Balomenos, N. Tsapatsoulis, S. Kollias, S. Kasderidis, and J. G. Taylor, *Attention-driven artificial agents*, Proceedings of the European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (EUNITE'03), July 2003.

[Bur98] C. J. C. Burges, *A tutorial on support vector machines for pattern recognition*, Data Mining and Knowledge Discovery **2** (1998), 121–167.

[CCLG02] O. Cakmakci, J. Coutaz, K. Van Laerhoven, and H.-W. Gellersen, *Context awareness in systems with limited resources*, Proceedings of the 3rd Workshop on Artificial Intelligence in Mobile Systems (AIMS), ECAI 2002, 2002, pp. 21–29.

[CCRR02] J. L. Crowley, J. Coutaz, G. Rey, and P. Reignier, *Perceptual components for context aware computing*, Proceedings of the 4th International Conference on Ubiquitous Computing (UbiComp'02) (G. Borriello and L. E. Holmquist, eds.), Lecture Notes in Computer Science, vol. 2498, September 2002, pp. 117–134.

[CDM$^+$00] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou, *Developing a context-aware electronic tourist guide: Some issues and experiences*, Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press, 2000, pp. 17–24.

[CGY96] I. J. Cox, J. Ghosn, and P. N. Yianilos, *Feature-based face recognition using mixture-distance*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1996, pp. 209–216.

[Cho59] N. Chomsky, *On certain formal properties of grammars*, Information and Control **2** (1959), 137–167.

[CK00] G. Chen and D. Kotz, *A survey of context-aware mobile computing research*, Tech. Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[CK02] ______, *Solar: A pervasive-computing infrastructure for context-aware mobile applications*, Tech. Report TR2002-421, Dept. of Computer Science, Dartmouth College, February 2002.

[CL01] C.-C. Chang and C.-J. Lin, *libSVM: a library for support vector machines*, 2001, Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CMP00] B. Clarkson, K. Mase, and A. Pentland, *Recognizing user context via wearable sensors*, Proceedings of the 4th International Symposium on Wearable Computers (ISWC 2000), 2000, pp. 69–76.

[CP98] B. Clarkson and A. Pentland, *Unsupervised clustering of ambulatory audio and video*, Tech. Report 471, MIT Media Lab, Perceptual Computing Group, 1998.

[CSP98] B. Clarkson, N. Sawhney, and A. Pentland, *Auditory context awareness via wearable computing*, Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI'98) (San Francisco, CA, USA), November 1998.

[CYEOH$^+$03] D. J. Cook, M. Youngblood, III E. O. Heierman, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja, *MavHome: An agent-based smart home*, Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03), IEEE Computer Society Press, March 2003, pp. 521–524.

[DA00] A. K. Dey and G. D. Abowd, *Towards a better understanding of context and context-awareness*, Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness, April 2000.

[DAS99] A. K. Dey, G. D. Abowd, and D. Salber, *A context-based infrastructure for smart environments*, Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99), 1999, pp. 114–128.

[DB92] P. Dourish and V. Bellotti, *Awareness and coordination in shared work spaces*, Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'92), ACM Press, 1992, pp. 107–114.

[DCB$^+$02] S. K. Das, D. J. Cook, A. Bhattacharya, III E. O. Heierman, and T.-Y. Lin, *The role of prediction algorithms in the MavHome smart home architecture*, IEEE Wireless Communications, special issue on Smart Homes **9** (2002), no. 6, 77–84.

[Die02] T. G. Dietterich, *Machine learning for sequential data: A review*, Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition, Springer, 2002, pp. 15–30.

[DLX01] M. Dash, H. Liu, and X. Xu, *'1 + 1 > 2': Merging distance and density based clustering*, Proceedings of the 7th International Conference on Database Systems for Advanced Applications (DASFAA), 2001.

[Dou04] P. Dourish, *What we talk about when we talk about context*, Personal Ubiquitous Computing **8** (2004), 19–30.

[DSAF99] A. K. Dey, D. Salber, G. D. Abowd, and M. Futakawa, *The conference assistant: Combining context-awareness with wearable computing*, Proceedings of the 3rd International Symposium on Wearable Computers (ISWC 1999), 1999, pp. 21–28.

[DWM02] M. Daszykowski, B. Walczak, and D. L. Massart, *On the optimal partitioning of data with k-means, growing k-means, neural gas, and growing neural gas*, Journal of Chemical Information and Computer Sciences **42** (2002), no. 1, 1378–1389.

[EAE04] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, *Using convolution to mine obscure periodic patterns in one pass*, Proceedings of the 9th International Conference on Extending Database Technology (EDBT), March 2004, pp. 605–620.

[EHAB99] M. Esler, J. Hightower, T. Anderson, and G. Borriello, *Next century challenges: Data-centric networking for invisible computing: The Portolano project at the University of Washington*, Proceedings of the 5th annual ACM/IEEE International Conference on Mobile Computing and Networking, August 1999, pp. 256–262.

[EHL01] M. R. Ebling, G. D. H. Hunt, and H. Lei, *Issues for context services for pervasive computing*, Proceedings of the Middleware 2001 Workshop on Middleware for Mobile Computing, 2001.

[EKS$^+$98] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu, *Incremental clustering for mining in a data warehousing environment*, Proceedings of the 24th International Conferance on Very Large Data Bases (VLDB), 24–27 1998, pp. 323–333.

[Eri02] T. Erickson, *Ask not for whom the cell phone tolls: Some problems with the notion of context-aware computing*, Communications of the ACM **45** (2002), 102–104.

[FBN01] A. Ferscha, W. Beer, and W. Narzt, *Location awareness in community wireless LANs*, Informatik 2001: Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit, Tagungsband der GI/OCG-Jahrestagung (K. Bauknecht, W. Brauer, and T. A. Mück, eds.), September 2001, pp. 190–195.

[Fer99] A. Ferscha, *Adaptive time warp simulation of timed petri nets*, IEEE Transactions on Software Engineering **25(2)** (1999), 237–257.

[Fer03a] ______, *Coordination in pervasive computing environments*, Proceedings of the 12th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2003), IEEE Computer Society Press, June 2003, (*invited paper*), pp. 3–9.

[Fer03b] ______, *Pervasive computing*, Datenbank-Spektrum (2003), 48–51.

[Fer03c] ______, *What is pervasive computing?*, Peter Rechenberg: Festschrift zum 70. Geburtstag (G. Blaschek, A. Ferscha, H. Mössenböck, and G. Pomberger, eds.), Universitätsverlag Trauner, June 2003, pp. 93–108.

[FG97] N. Friedman and M. Goldszmidt, *Sequential update of bayesian network structure*, Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI 97), 1997.

[FHMO04a] A. Ferscha, M. Hechinger, R. Mayrhofer, and R. Oberhauser, *A light-weight component model for peer-to-peer applications*, Proceedings of the 2nd International Workshop on Mobile Distributed Computing (MDC04), IEEE Computer Society Press, March 2004, pp. 520–527.

[FHMO04b] ______, *A peer-to-peer light-weight component model for context-aware smart space applications*, International Journal of Wireless and Mobile Computing (IJWMC), special issue on Mobile Distributed Computing (2004), *(invited article)*, extended version of [FHMO04a], *to appear*.

[FK96] K. Fokianos and B. Keden, *Recursive estimation for time series following generalized linear models*, Tech. Report TR 1996-48, Institute for Systems Research, University of Maryland, 1996.

[Fol93] *The free on-line dictionary of computing*, `http://www.foldoc.org/`, Editor Denis Howe, 1993.

[Fre] The Abbington Group, *The Wi-Fi-FreeSpot Directory*, `http://www.wififreespot.com/`.

[Fri95a] B. Fritzke, *Growing grid - a self-organizing network with constant neighborhood range and adaptation strength*, Neural Processing Letters (1995), 9–13.

[Fri95b] ______, *A growing neural gas network learns topologies*, Advances in Neural Information Processing Systems 7 (G. Tesauro, D. S. Touretzky, and T. K. Leen, eds.), MIT Press, Cambridge MA, 1995, pp. 625–632.

[Fri96] ———, *Growing self-organizing networks – Why?*, Proceedings of ESANN'96: European Symposium on Artificial Neural Networks (M. Verleysen, ed.), D-Facto Publishers, 1996, *(invited paper)*, pp. 61–72.

[Fri97a] J. H. Friedman, *On bias, variance, 0/1—loss, and the curse-of-dimensionality*, Data Mining and Knowledge Discovery **1** (1997), 55–77.

[Fri97b] B. Fritzke, *Some competitive learning methods*, Tech. report, Systems Biophysics, Inst. for Neural Comp., Ruhr-Universität Bochum, April 1997.

[Fri98] N. Friedman, *The Bayesian structural EM algorithm*, Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI), 1998, pp. 129–138.

[GC03] K. Gopalratnam and D. J. Cook, *Active Lezi: An incremental parsing algorithm for sequential prediction*, Proceedings of the Florida Artificial Intelligence Research Symposium, 2003.

[GJ97] Z. Ghahramani and M. I. Jordan, *Factorial hidden markov models*, Machine Learning (1997), no. 29, 245–275.

[Gol84] A. Goldberg, *Smalltalk-80: The interactive programming environment*, Addison-Wesley, 1984.

[GPZ04a] T. Gu, H. K. Pung, and D. Q. Zhang, *A bayesian approach for dealing with uncertain contexts*, Advances in Pervasive Computing (A. Ferscha, H. Hörtner, and G. Kotsis, eds.), vol. 176, Austrian Computer Society (OCG), April 2004, part of the Second International Conference on Pervasive Computing (PERVASIVE 2004), pp. 205–210.

[GPZ04b] ———, *A middleware for building context-aware mobile*, Proceedings of the IEEE Vehicular Technology Conference (VTC-Spring 2004), May 2004.

[Gro76] S. Grossberg, *Adaptive pattern classification and universal recoding: Parallel development and coding of neural feature detectors*, Biological Cybernetics **23** (1976), 121–134.

[GSB02] H.W. Gellersen, A. Schmidt, and M. Beigl, *Multi-sensor context-awareness in mobile devices and smart artefacts*, Mobile Networks and Applications **7** (2002), 341–351.

[GSSS02] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, *Project Aura: Towards distraction-free pervasive computing*, IEEE Pervasive Computing **21** (2002), no. 2, 22–31.

[GWPZ04] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang, *An ontology-based context model in intelligent environments*, Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS'04), January 2004.

[Ham01] F. H. Hamker, *Life-long learning cell structures—continuously learning without catastrophic interference*, Neural Networks **14** (2001), no. 4–5, 551–573.

[HBB02] J. Hightower, B. Brumitt, and G. Borriello, *The Location Stack: A layered model for location in ubiquitous computing*, Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002) (Callicoon, NY, US), IEEE Computer Society Press, June 2002, pp. 22–28.

[HC03] E. Heierman and D. J. Cook, *Improving home automation by discovering regularly occurring device usage patterns*, Proceedings of the International Conference on Data Mining, IEEE Computer Society Press, November 2003, pp. 537–540.

[HCL03] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, *A practical guide to support vector classification*, July 2003, available at `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

[Hea03] R. Headon, *Movement awareness for a sentient environment*, Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03), IEEE Computer Society Press, March 2003, pp. 99–106.

[Hec95] D. Heckerman, *A tutorial on learning with bayesian networks*, Tech. Report MSR-TR-95-06, Microsoft Research, 1995, Revised June 1996.

[HG98] F. H. Hamker and H.-M. Gross, *A lifelong learning approach for incremental neural networks*, Proceedings of the 14th European Meeting on Cybernetics and Systems Research (EMCSR'98), 1998, pp. 599–604.

[HGZP04] X. H.Wang, T. Gu, D. Q. Zhang, and H. K. Pung, *Ontology based context modeling and reasoning using OWL*, Proceedings of the Workshop on Context Modeling and Reasoning (CoMoRea'04), March 2004.

[HH97] F. Hamker and D. Heinke, *Implementation and comparison of growing neural gas, growing cell structures and fuzzy artmap*, Tech. Report 1/97, Technical University of Ilmenau, April 1997.

[HHS92] T. K. Ho, J. J. Hull, and S. N. Srihari, *On multiple classifier systems for pattern recognition*, Proceedings of the 11th International Conference on Pattern Recognition, August 1992, pp. 84–87.

[HI04] K. Henricksen and J. Indulska, *A software engineering framework for context-aware pervasive computing*, Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom'04), IEEE Computer Society Press, March 2004, pp. 77–86.

[Hig03] J. Hightower, *From position to place*, Proceedings of the 2003 Workshop on Location-Aware Computing, October 2003, part of the 2003 Ubiquitous Computing Conference, pp. 10–12.

[HIR01] K. Henricksen, J. Indulska, and A. Rakotonirainy, *Infrastructure for pervasive computing: Challenges*, GI Jahrestagung (1), 2001, pp. 214–222.

[HKKJ02] E. Horvitz, P. Koch, C. M. Kadie, and A. Jacobs, *Coordinate: Probabilistic forecasting of presence and availability*, Proceedings of the 18th Conference on Uncertainty and Artificial Intelligence, Morgan Kaufmann Publishers, July 2002, pp. 224–233.

[HKM$^+$01] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmäki, and H. Toivonen, *Time series segmentation for context recognition in mobile devices*, Proceedings of the 2001 IEEE International Conference on Data Mining (Washington, DC, USA), IEEE Computer Society Press, 2001, pp. 203–210.

[HLA+04] L. Harvel, L. Liu, G. D. Abowd, Yu-Xi Lim, C. Scheibe, and C. Chatham, *Context Cube: Flexible and effective manipulation of sensed context data*, Proceedings of the 2nd International Conference on Pervasive Computing (PERVASIVE 2004) (A. Ferscha and F. Mattern, eds.), Lecture Notes in Computer Science, vol. 3001, Springer, April 2004, pp. 51–68.

[HNKF03] K. Hisazumi, T. Nakanishi, T. Kitasuka, and A. Fukuda, *Campus: A context-aware middleware*, Proceedings of the 2nd CREST Workshop on Advanced Computing and Communicating Techniques for Wearable Information Playing, May 2003, pp. 42–49.

[HP98] J. Healey and R. W. Picard, *Digital processing of affective signals*, Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, May 1998.

[HSW89] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural Networks **2** (1989), no. 5, 359–366.

[IEE99] *802-11: IEEE standards for information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*, 1999.

[IEE02] *802-15.1: IEEE standard for information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)*, 2002.

[JS03] G. Judd and P. Steenkiste, *Providing contextual information to pervasive computing applications*, Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03), IEEE Computer Society Press, March 2003, pp. 133–142.

[Kea96] R. B. Kearfott, *Interval computations: Introduction, uses and resources*, Euromath Bulletin **2(1)** (1996), 95–112.

[KKP+03] P. Korpipää, M. Koskinen, J. Peltola, S.-M. Mäkelä, and T. Seppänen, *Bayesian approach to sensor-based context awareness*, Personal and Ubiquitous Computing **7** (2003), no. 2, 113–124.

[KMS+00] T. Kindberg, H. Morris, J. Schettino, B. Serra, M. Spasojevic, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, and V. Krishnan, *People, places, things: Web presence for the real world*, Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'00), IEEE Computer Society Press, December 2000, pp. 19–30.

[KOA+99] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. D. Mynatt, T. Starner, and W. Newstetter, *The Aware Home: A living laboratory for ubiquitous computing research*, Proceedings of the Cooperative Buildings, Integrating Information, Organization, and Architecture, Second International Workshop, CoBuild'99, Lecture Notes in Computer Science, vol. 1670, Springer, 1999, pp. 191–198.

[Koh95] Tuevo Kohonen, *Self-organizing maps*, Series in Information Sciences, vol. 30, Springer, Berlin, Heidelberg, New York, 1995.

[Kol57] A.N. Kolmogorov, *On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition*, Doklady Akademii **114** (1957), 953–956.

[Lae99] K. Van Laerhoven, *On-line adaptive context awareness starting from low-level sensors*, Master's thesis, Free University of Brussels, 1999.

[Lae01] ______, *Combining the Self-Organizing Map and K-Means clustering for on-line classification of sensor data*, Proceedings of the International Conference on Artificial Neural Networks (ICANN), Springer, 2001, pp. 464–469.

[LB00] C. Li and G. Biswas, *A bayesian approach to temporal data clustering using hidden markov models*, Proceedings of the 17th International Conference on Machine Learning (ICML 2000), Morgan Kaufmann Publishers, 2000, pp. 543–550.

[LBVW03] S. Lühr, H. H. Bui, S. Venkatesh, and G. A.W. West, *Recognition of human activity through hierarchical stochastic learning*, Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03), IEEE Computer Society Press, March 2003, pp. 416–422.

[LC00] K. Van Laerhoven and O. Cakmakci, *What shall we teach our pants?*, Proceedings of the 4th International Symposium on Wearable Computers (ISWC 2000), IEEE Computer Society Press, 2000, pp. 77–83.

[LC01] B. Leuf and W. Cunningham, *The Wiki way*, Addison-Wesley, April 2001.

[LF94] M. Lamming and M. Flynn, *Forget-me-not: Intimate computing in support of human memory*, Proceedings of FRIEND21'94, the International Symposium on Next Generation Human Interface, February 1994.

[LFPR04] T. Lindner, L. Fritsch, K. Plank, and K. Rannenberg, *Exploitation of public and private WiFi coverage for new business models*, Proceedings of the IFIP, IFIP World Congress 2004, 2004, *to appear*.

[LJS$^+$02] P. Lukowicz, H. Junker, M. Stäger, T. von Büren, and G. Tröster, *WearNET: A distributed multi-sensor system for context aware wearables*, Proceedings of the 4th International Conference on Ubiquitous Computing (UbiComp'02) (G. Borriello and L. E. Holmquist, eds.), Lecture Notes in Computer Science, vol. 2498, September 2002, pp. 361–370.

[LL01] K. Van Laerhoven and S. Lowette, *Real-time analysis of data from many sensors with neural networks*, Proceedings of the 5th International Symposium on Wearable Computers (ISWC 2001), IEEE Computer Society Press, October 2001.

[Lou01] S. Loughran, *Towards an adaptive and context aware laptop*, Tech. Report HPL-2001-158, Hewlett-Packard Labs, June 2001.

[LRT04] K. Laasonen, M. Raento, and H. Toivonen, *Adaptive on-device location recognition*, Proceedings of the 2nd International Conference on Pervasive Computing (PERVASIVE 2004) (A. Ferscha and F. Mattern, eds.), Lecture Notes in Computer Science, vol. 3001, Springer, April 2004, pp. 287–304.

[LS00] H. Lieberman and T. Selker, *Out of context: Computer systems that adapt to, and learn from, context*, IBM Systems Journal **39** (2000), no. 3&4, 617–632.

[LSG02] K. Van Laerhoven, A. Schmidt, and H.-W. Gellersen, *Multi-sensor context aware clothing*, 2002, pp. 49–57.

[LT99] R. Low and R. Togneri, *Evolution of markovian speech models*, Tech. report, Centre for Intelligent Information Processing Systems, University of Western Australia, December 1999.

[LWJ$^+$04] P. Lukowicz, J. A. Ward, H. Junker, M. Stäger, G. Tröester, A. Atrash, and T. Starner, *Recognizing workshop activity using body worn microphones and accelerometers*, Proceedings of the 2nd International Conference on Pervasive Computing (PERVASIVE 2004) (A. Ferscha and F. Mattern, eds.), Lecture Notes in Computer Science, vol. 3001, Springer, April 2004, pp. 18–32.

[May02] R. Mayrhofer, *A new approach to a fast simulation of spiking neural networks*, Master's thesis, Johannes Kepler University of Linz, Austria, July 2002.

[May04] ______, *An architecture for context prediction*, Advances in Pervasive Computing (A. Ferscha, H. Hörtner, and G. Kotsis, eds.), vol. 176, Austrian Computer Society (OCG), April 2004, part of the Second International Conference on Pervasive Computing (PERVASIVE 2004), pp. 65–72.

[MBS93] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, *"Neural-Gas" network for vector quantization and its application to time-series prediction*, IEEE Transactions on Neural Networks **4** (1993), no. 4, 558–569.

[MHH03] J. Mäntyjärvi, J. Himberg, and P. Huuskonen, *Collaborative context recognition for handheld devices*, Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03), IEEE Computer Society Press, March 2003, pp. 161–168.

[MM99] K. Murphy and S. Mian, *Modelling gene expression data using dynamic bayesian networks*, Tech. report, Computer Science Division, University of California, Berkeley, CA, 1999.

[MOG97] S. Mukherjee, E. Osuna, and F. Girosi, *Nonlinear prediction of chaotic time series using a support vector machine*, IEEE Neural Network for Signal Processing (NNSP'97), September 1997.

[Moz98] M. C. Mozer, *The neural network house: An environment that adapts to its inhabitants*, Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments, AAAI Press, 1998, pp. 110–114.

[MR03] S. Meyer and A. Rakotonirainy, *A survey of research on context-aware homes*, Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003 (Darlinghurst, Australia), vol. 21, Australian Computer Society, Inc, 2003, pp. 159–168.

[MRF03a] R. Mayrhofer, H. Radi, and A. Ferscha, *Feature extraction in wireless personal and local area networks*, Proceedings of the 5th IFIP-TC6 International Conference on Mobile and Wireless Communications Networks (MWCN 2003) (Khaldoun Al Agha and Cambyse Guy Omidyar, eds.), World Scientific, October 2003, pp. 195–198.

[MRF03b] ______, *Recognizing and predicting context by learning from user behavior*, Proceedings of the International Conference On Advances in Mobile Multimedia (MoMM2003) (W. Schreiner G. Kotsis, A. Ferscha and K. Ibrahim, eds.), vol. 171, Austrian Computer Society (OCG), September 2003, pp. 25–35.

[MRF04a] ______, *A context prediction code and data base*, Proceedings of the Benchmarks and a Database for Context Recognition Workshop (H. Junker, P. Lukowicz, and J. Mäntyjarvi, eds.), ETH Zurich, April 2004, part of the Second International Conference on Pervasive Computing (PERVASIVE 2004), pp. 20–26.

[MRF04b] ______, *Recognizing and predicting context by learning from user behavior*, Radiomatics: Journal of Communication Engineering, special issue on Advances in Mobile Multimedia **1** (2004), no. 1, *(invited article)*, extended version of [MRF03b], *to appear*.

[MSR$^+$97] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, *Predicting time series with support vector machines*, Proceedings of the 7th International Conference on Artificial Neural Networks, Lecture Notes in Computer Science, Springer, 1997, pp. 999–1004.

[MTP03] J. Ma, J. Theiler, and S. Perkins, *Accurate on-line support vector regression*, Neural Computation **15** (2003), 2683–2704.

[Mur02] K. Murphy, *Dynamic bayesian networks: Representation, inference and learning*, Ph.D. thesis, UC Berkeley, Computer Science Division, July 2002.

[NB01] S. Negri and L. Belanche, *Heterogeneous Kohonen networks*, Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001 (José Mira and Alberto Prieto, eds.), Lecture Notes in Computer Science, Springer, 2001, pp. 243–252.

[Nel01] O. Nelles, *Nonlinear system identification*, Springer, Germany, 2001.

[Nor98] D. A. Norman, *The invisible computer*, MIT Press, Cambridge, 1998.

[Ore02] *Deliverable D05: 1st year progress report of the Oresteia project*, January 2002, available at `http://manolito.image.ece.ntua.gr/oresteia/htmldocs/deliverables.htm`.

[OSG03] The OSGI Alliance, *OSGi service platform, release 3*, IOS Press, Amsterdam, The Netherlands, 2003.

[PBTU03a] J. Petzold, F. Bagci, W. Trumler, and T. Ungerer, *Context prediction based on branch prediction methods*, Tech. Report 2003-14, Institut für Informatik, Universität Augsburg, July 2003.

[PBTU03b] ______, *Global and local state context prediction*, Proceedings of the Artificial Intelligence in Mobile Systems 2003 (AIMS 2003), October 2003.

[PBTU03c] ______, *The state predictor method for context prediction*, Adjunct Proceedings of the 5th International Conference on Ubiquitous Computing 2003, October 2003.

[PBTU04] ______, *Global state context prediction techniques applied to a smart office building*, Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference, January 2004.

[Pic75] F. Pichler, *Mathematische Systemtheorie: Dynamische Konstruktionen*, Walter de Gruyter, Berlin, New York, 1975.

[PLFK03] D. J. Patterson, L. Liao, D. Fox, and H. Kautz, *Inferring high-level behavior from low-level sensors*, Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp'03) (A. Dey, A. Schmidt, and J. McCarthy, eds.), Lecture Notes in Computer Science, vol. 2864, Springer, October 2003, pp. 73–89.

[PPR] *Portland pattern repository*, `http://c2.com/ppr/`.

[Rab89] L. R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Proceedings of the IEEE **77** (1989), 257–286.

[Rad04] H. Radi, *Adding smartness to mobile devices - recognizing context by learning from user habits*, Master's thesis, Johannes Kepler University Linz, Austria, 2004, *to appear*.

[RC98] M. T. Rosenstein and P. R. Cohen, *Concepts from time series*, Proceedings of the 15th National/10th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI/IAAI), 1998, pp. 739–745.

[RC99] ______, *Continuous categories for a mobile robot*, Proceedings of the 16th National/11th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI/IAAI), 1999, pp. 634–640.

[RC03] S. P. Rao and D. J. Cook, *Improving the performance of action prediction through identification of abstract tasks*, Proceedings of the Florida Artificial Intelligence Research Symposium, May 2003, pp. 43–47.

[RM02] C. Randell and H. Muller, *The well mannered wearable computer*, Personal and Ubiquitous Computing (2002), 31–36.

[RMF04] H. Radi, R. Mayrhofer, and A. Ferscha, *A notebook sensory data set for context recognition*, Proceedings of the Benchmarks and a Database for Context Recognition Workshop (H. Junker, P. Lukowicz, and J. Mäntyjarvi, eds.), ETH Zurich, April 2004, part of the Second International Conference on Pervasive Computing (PERVASIVE 2004), pp. 17–19.

[RN95] S. Russell and P. Norvig, *Artificial intelligence: A modern approach*, Prentice Hall Series in Artificial Intelligence, Prentice Hall, Upper Saddle River, New Jersey 07458, 1995.

[Ruf98] Berthold Ruf, *Computing and learning with spiking neurons - theory and simulations*, Ph.D. thesis, Graz University of Technology, 1998.

[SA96] R. Srikant and R. Agrawal, *Mining sequential patterns: Generalizations and performance improvements*, Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology, Springer, March 1996, pp. 3–17.

[Sat90] M. Satyanarayanan, *Scalable, secure, and highly available distributed file access*, Computer **23** (1990), no. 5, 9–18, 20–21.

[SAW94] B. N. Schilit, N. Adams, and R. Want, *Context-aware computing applications*, IEEE Workshop on Mobile Computing Systems and Applications, IEEE Computer Society Press, 1994.

[SBG99] A. Schmidt, M. Beigl, and H.-W. Gellersen, *There is more to context than location*, Computers and Graphics **23** (1999), no. 6, 893–901.

[Sch95] B. N. Schilit, *A system architecture for context-aware mobile computing*, Ph.D. thesis, Columbia University, May 1995.

[Sch02a] A. Schmidt, *Ubiquitous computing – computing in context*, Ph.D. thesis, Lancaster University, November 2002.

[Sch02b] K. Schmidt, *The problem with 'awareness': Introductory remarks on 'awareness in CSCW'*, Computer Supported Cooperative Work **11** (2002), 285–298.

[SDA99] D. Salber, A. K. Dey, and G. D. Abowd, *The context toolkit: Aiding the development of context-enabled applications*, Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), May 1999, pp. 434–441.

[SG97] A. Silberschatz and P. B. Galvin, *Operating system concepts*, Addison-Wesley, 1997.

[SLS$^+$02] A. Schmidt, K. Van Laerhoven, M. Strohbach, A. Friday, and H. W. Gellersen, *Context acquisition based on load sensing*, Proceedings of the 4th International Conference on Ubiquitous Computing (UbiComp'02) (G. Borriello and L. E. Holmquist, eds.), Lecture Notes in Computer Science, vol. 2498, September 2002, pp. 333–351.

[SS97] Y. Sazeides and J. E. Smith, *The predictability of data values*, Proceedings of the International Symposium on Microarchitecture, 1997, pp. 248–258.

[TF93] Z. Tang and P. A. Fishwick, *Feed-forward neural nets as models for time series forecasting*, Tech. Report 91-008, University of Florida, 1993, also published in ORSA Journal of Computing 5(4), 374–386.

[TIL04] E. M. Tapia, S. S. Intille, and K. Larson, *Activity recognition in the home using simple and ubiquitous sensors*, Proceedings of the 2nd International Conference on Pervasive Computing (PERVASIVE 2004) (A. Ferscha and F. Mattern, eds.), Lecture Notes in Computer Science, vol. 3001, Springer, April 2004, pp. 158–175.

[Toi00] S. Toivonen, *Definition and usage of a software agent*, Arpakannus (2000), 9–13.

[UPRS03] A. Upegui, C. A. Peña-Reyes, and E. Sánchez, *A methodology for evolving spiking neural-network topologies on line using partial dynamic reconfiguration*, Proceedings of the International Congress on Computational Intelligence, CIIC-2003, November 2003.

[VHAP99] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas, *Self-organizing map in Matlab: the SOM toolbox*, Proceedings of the Matlab DSP Conference (Espoo, Finland), November 1999, pp. 35–40.

[Web] *Merriam-webster online dictionary*, `http://www.m-w.com/`.

[Wei91] M. Weiser, *The computer of the twenty-first century*, Scientific American **1496** (1991), 94–100.

[WHFG92] R. Want, A. Hopper, V. Falcão, and J. Gibbons, *The active badge location system*, ACM Transactions on Information Systems (TOIS) **10** (1992), 91–102.

[Wik] *Wikipedia*, `http://wikipedia.org/`.

[WJ94] M. Wooldridge and N. R. Jennings, *Intelligent agents: Theory and practice*, `http://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h` (Hypertext version of Knowledge Engineering Review paper), 1994.

[WLW01] M. A. Orgun W. Lin and G. J. Williams, *Multilevels hidden markov models for temporal data mining*, Proceedings of the KDD 2001 Workshop on Temporal Data Mining (held in conjunction with the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)) (San Francisco, CA, USA), August 2001.

[WN94] A. Weigend and D. Nix, *Predictions with confidence intervals (local error bars)*, Proceedings of the International Conference on Neural Information Processing (ICONIP'94), 1994, pp. 847–852.

[XCS$^+$04] S. A. Xynogalas, M. K. Chantzara, I. C. Sygkouna, S. P. Vrontis, I. G. Roussaki, and M. E. Anagnostou, *Context management for the provision of adaptive services to roaming users*, IEEE Wireless Communications (2004), 40–47.

[XEKS98] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander, *A distribution-based clustering algorithm for mining in large spatial databases*, Proceedings of the 14th International Conference on Data Engineering (ICDE'98), IEEE Computer Society Press, 1998, pp. 324–331.

[XRCA04] S. Xynogalas, I. Roussaki, M. Chantzara, and M. Anagnostou, *Context management in virtual home environment systems*, Journal of Circuits, Systems and Computers (JCSC) **13** (2004), no. 2, 293–311.

[YGT03] K. Yang, A. Galis, and C. Todd, *Policy-driven mobile agents for context-aware service in next generation networks*, Proceedings of the MATA 2003 - IFIP 5th International Conference on Mobile Agents for Telecommunications, October 2003.

[Zel94] Andreas Zell, *Simulation neuronaler Netze*, R. Oldenbourg Verlag München Wien, 1994.

[Zim96] T. Zimmerman, *Personal area networks: Near-field intrabody communication*, IBM Systems Journal **35** (1996), no. 3&4.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny, *BIRCH: an efficient data clustering method for very large databases*, Proceedings of the 1996 ACM SIGMOD international conference on Management of data, ACM Press, 1996, pp. 103–114.

CURRICULUM VITAE

PERSONAL DATA

Name	Rene Michael Mayrhofer
Date of Birth	April 30th 1979
Place of Birth	Graz, Austria
Nationality	Austrian
Marital status	Single

EDUCATION

1985–1989	Elementary School: Volksschule Behamberg, Austria
1989–1993	Secondary School: Bundesrealgymnasium Steyr, Austria
1993–1998	Technical High School: HTBLA for Electrical Engineering and Computer Science, Steyr, Austria
1998–2002	Studies in Computer Science at the Johannes Kepler University of Linz, Austria
2002–2004	Doctoral Studies in Technical Sciences at the Johannes Kepler University of Linz, Austria

POSITIONS

July 1995	HTML development at Profactor GmbH Steyr
July 1996	System administration and installation of Oracle databases at RiS GmbH Steyr
August 1996	Migration of network infrastructure (Novell based to Microsoft Windows NT) at Profactor GmbH Steyr
July 1997	System and network administration at RiS GmbH Steyr
August–September 1997	OLE programming at Profactor GmbH Steyr
August 1998	Foundation of ViaNova DI Johannes Guger KEG
July–September 1998	Development of parts of CAPP Knowledge (SAP R/3 extension), customization of CAPP Knowledge for BMW GmbH Steyr
since March 1999	Development and project management of Gibraltar (Linux based firewall product), see `http://www.gibraltar.at/`
July–September 1999	Development of parts of CAPP Knowledge
February 2000	Development of parts of CAPP Knowledge
March–June 2001	Tutor at the University of Linz
June–September 2001	Student trainee at BMW, Munich (research and development center)
March 2002	Retirement from ViaNova DI Johannes Guger KEG

since November 2002	University Assistant at the Institut für Pervasive Computing at the Johannes Kepler University of Linz, Austria
February 2003	Approval of trade license for information technology
since June 2003	Cooperation with eSys GmbH for development, marketing and distribution of Gibraltar firewall

PUBLICATIONS

1. R. Mayrhofer, M. Affenzeller, H. Prähofer, G. Höfer, A. Fried
"DEVS Simulation of Spiking Neural Networks", *Proceedings of the 16th European Meeting on Cybernetics and Systems Research 2002,* vol. 2, pp. 573–578, April 2002.

2. R. Mayrhofer
"A New Approach to a Fast Simulation of Spiking Neural Networks", Diploma thesis, Johannes Kepler University of Linz, Austria, July 2002.

3. M. Affenzeller, R. Mayrhofer
"Generic Heuristics for Combinatorial Optimization Problems", *Proceedings of the 9th International Conference on Operational Research (KOI),* pp. 83–92, 2002.

4. R. Mayrhofer, F. Ortner, A. Ferscha, M. Hechinger
"Securing Passive Objects in Mobile Ad-Hoc Peer-to-Peer Networks", *Electronic Notes in Theoretical Computer Science: SECCO 2003*, Elsevier Science, issue 85.3, ISSN 1571-0661, June 2003.

5. R. Mayrhofer, H. Radi, A. Ferscha
"Recognizing and Predicting Context by Learning from User Behavior", *Proceedings of the International Conference On Advances in Mobile Multimedia (MoMM2003),* Austrian Computer Society (OCG), ISBN 3-85403-171-8, pp. 25–35, September 2003.

6. R. Mayrhofer, H. Radi, A. Ferscha
"Feature Extraction in Wireless Personal and Local Area Networks", *Proceedings of the 5th IFIP TC6 International Conference on Mobile and Wireless Communications Networks (MWCN 2003),* World Scientific, ISBN 981-238-686-6, pp. 195–198, October 2003.

7. A. Ferscha, M. Hechinger, R. Mayrhofer, R. Oberhauser
"A Light-Weight Component Model for Peer-to-Peer Applications", *Proceedings of the 2nd International Workshop on Mobile Distributed Computing (MDC04),* IEEE Computer Society Press, pp. 520–527, March 2004.

8. R. Mayrhofer
"An Architecture for Context Prediction", *Advances in Pervasive Computing,* Austrian Computer Society (OCG), volume 176, ISBN 3-85403-176-9, pp. 65–72, April 2004.

9. R. Mayrhofer, H. Radi and A. Ferscha
"A Context Prediction Code and Data Base", *Proceedings of the Benchmarks and a Database for Context Recognition Workshop*, part of the 2nd International Conference on Pervasive Computing (PERVASIVE 2004), ETH Zurich, ISBN 3-9522686-2-3, pp. 20–26, April 2004.

10. H. Radi, R. Mayrhofer and A. Ferscha
"A Notebook Sensory Data Set for Context Recognition", *Proceedings of the Benchmarks and a Database for Context Recognition Workshop*, part of the 2nd International Conference on Pervasive Computing (PERVASIVE 2004), ETH Zurich, ISBN 3-9522686-2-3, pp. 17–19, April 2004.

11. A. Ferscha, M. Hechinger, R. Mayrhofer, M. dos Santos Rocha, M. Franz, and R. Oberhauser
"Digital Aura", *Advances in Pervasive Computing*, volume 176, pages 405-410. Austrian Computer Society (OCG), April 2004.

12. R. Mayrhofer, H. Radi, A. Ferscha
"Recognizing and Predicting Context by Learning from User Behavior", *Radiomatics: Journal of Communication Engineering, special issue on Advances in Mobile Multimedia, (invited article, extended version of 5.),* ITB Press, ISSN 1693-5152, vol. 1, no. 1, May 2004, *(to appear)*.

13. A. Ferscha, M. Hechinger, R. Mayrhofer, R. Oberhauser
"A Peer-to-Peer Light-Weight Component Model for Context-Aware Smart Space Applications", *International Journal of Wireless and Mobile Computing (IJWMC), special issue on Mobile Distributed Computing, (invited article, extended version of 7.),* issue 4, 2004, *(to appear)*.

Eidesstattliche Erklärung

Ich erkläre an Eides Statt, dass ich die vorliegende Dissertation mit dem Titel "An Architecture for Context Prediction" selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, 23. Oktober 2004 Dipl.-Ing. Rene Michael Mayrhofer