ON THE IMPORTANCE of time synchronization for context aware applications

Bernd Niklas Klein

kassel university press

ON THE IMPORTANCE of time synchronization for context aware applications

Bernd Niklas Klein

kassel university press

Bernd Niklas Klein

On the importance of time synchronization for context aware applications

kassel
university
press

Die vorliegende Arbeit wurde vom Fachbereich Elektrotechnik / Informatik der Universität Kassel als Dissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) angenommen.

Erster Gutachter:	Prof. Dr.-Ing. Klaus David
Zweiter Gutachter:	Prof. Dr. rer. nat. Bernhard Sick

Tag der mündlichen Prüfung 22. Juli 2011

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar

Zugl.: Kassel, Univ., Diss. 2011
ISBN print: 978-3-86219-226-7
ISBN online: 978-3-86219-227-4
URN: http://nbn-resolving.de/urn:nbn:de:0002-32279

Acknowledgement

I am grateful for all the help and advice without which I would not have been able to finish this thesis.

My deepest gratitude is to Prof. Dr. Klaus David for his support and constant encouragement. I was very fortunate to have an adviser who always asks the right questions and forces me to give answers in a way that is more precise and at the same time more vivid. I am grateful to Prof. Dr. Bernhard Sick for agreeing to be my second adviser. I also thank Prof. Dr. Dirk Dahlhaus and the whole board of examiners for accepting to review this thesis.

All my colleagues at ComTec have been a great source of inspiration during the last four years. I am grateful to Sian Lun Lau for sharing some of the tools he developed to speed up the task of data collection and pre-processing for machine learning. He was always helpful and his dedication inspired me. With Andreas Pirali I had many great conversation that where tremendously helpful to endure the stress and recover from the crises that are inevitable when working on a task like that. Rico Kusber constantly answered my questions concerning the formal process of submitting the thesis. His experiences were very helpful to me.

Finally, I would like to express my heart-felt gratitude to my friends and family.

Abstract

Twenty years ago MarK Weiser formulated the vision of ubiquitous computing. A vision by definition is something unreal, so we should no longer talk of the vision of ubiquitous computing but rather of the reality ubiquitous computing. We are in the age of mobile computing, a key enabler of ubiquitous computing, home automation is spreading out more than ever, and evermore home electronics devices have built in network connectivity.

Central to Weiser's considerations was the threatening nature of the pervasive computing devices that constantly may demand our focus. That led Weiser to the belief that in ubiquitous computing the devices have to fade into the background. We share that belief, the technological advances we are experiencing have to be accompanied by software approaches that allow the devices to act autonomously to our benefit.

After more than fifteen years of research on the topic of context awareness we are on the verge of building consumer products that utilize the findings of this research field. However, we feel that one crucial aspect of context awareness is so far overlooked. In this thesis we plead for the importance of that aspect, time synchronization.

There is no single sensor that is able to correctly and completely access a user's situation, context awareness is always dependent on a multitude of sensor information to be aggregated to at least approximate a user's situation. It cannot be safely assumed that the clocks of all these sensor devices are accurately synchronized. The several reasons are detailed in this thesis. We also will show that not all, but a significant share of context awareness algorithms is dependent on proper time synchronization to achieve the desired accuracy.

Given this dependency, we propose two approaches to cope with the problems that arise when we deal with time synchronization in ubiquitous computing. The very nature of

ubiquitous computing hinders the simple reuse of known time synchronisation protocols.

These protocols are still valuable and may be useful in certain configurations of ubiquitous devices, but we will show why they are not always feasible. In the end, the known protocols together with the two proposed in this thesis will help to ensure better time synchronization in ubiquitous computing applications and thus better accuracy for context reasoning. Thus, the main concern of this thesis is to raise the awareness of the role of time synchronisation in context reasoning dependent applications.

Zusammenfassung

Vor zwanzig Jahren hat Mark Weiser seine Vision von 'ubiquitous computing' formuliert. Eine Vision ist per Definition nicht real, daher sollten wir nicht länger von einer Vision reden, wenn wir von 'ubiquitous computing' reden. 'Ubiquitous computing' ist vielmehr die Realität. Wir leben im Zeitalter des 'mobile computing', einem wichtigen Teil von 'ubiquitous computing'; es ist heute nichts Außergewöhnliches, Heizung, Beleuchtung, Belüftung und Ähnliches zuhause automatisch steuern zu lassen und immer mehr Haushaltsgeräte kommen mit Netzwerkfähigkeiten.

Zentraler Punkt in Weisers Überlegungen war die bedrohliche Natur der allgegenwärtigen Computer, die nach unserer Aufmerksamkeit verlangen. Dies führte Weiser zu der Überzeugung, dass beim 'ubiquitous computing' die Computer im Hintergrund verschwinden müssen. Wir teilen diese Überzeugung, dass die technologischen Fortschritte, die wir erfahren müssen, von Softwareansätzen begleitet werden, die es ermöglichen, dass die Technologie autonom zu unserem Vorteil handelt.

Nach mehr als fünfzehn Jahren der Forschung im Bereich der Kontextsensitivtät stehen wir kurz davor, dass Endverbraucherprodukte von den Erkenntnissen dieser Forschung Gebrauch machen. Allerdings denken wir, dass bisher ein wichtiger Aspekt der Kontextsensitivität übersehen wurde. In dieser Dissertation plädieren wir für die Beachtung dieses Aspekts, der Zeitsynchronisation.

Es gibt nicht den einen Sensor, der alleine die Situation eines Nutzers korrekt und vollständig erkennen könnte. Kontextsensitivität ist immer auf die Kombination von Informationen vieler verschiedener Sensoren angewiesen, um die Situation näherungsweise zu erkennen. Dabei kann man nicht einfach davon ausgehen, dass die Uhren aller Sensorgeräte korrekt synchronisiert sind. Die verschiedenen Gründe dafür sind in dieser Dissertation aufgeführt. Des weiteren wird gezeigt,

dass zwar nicht alle, aber doch sehr viele Algorithmen im Bereich der Kontextsensitivität davon abhängig sind, dass die Uhren richtig synchronisiert sind und nur so die gewünschte Genauigkeit erreicht werden kann.

Aufgrund dieser Abhängigkeit werden hier zwei Ansätze vorgestellt, wie die Probleme, die auftauchen, wenn es um Zeitsynchronisation in 'ubiquitous computing' geht, überwunden werden können. Es liegt in der Natur des 'ubiquitous computing', dass bekannte Protokolle zur Zeitsynchronisation nicht einfach übertragen werden können.

Sicher sind die bekannten Protokolle immer noch wertvoll und können in bestimmten Konstellationen im 'ubiquitous computing' eingesetzt werden. Trotzdem werden wir zeigen, warum die bekannten Protokolle nicht immer funktionieren. Insgesamt werden sich die bekannten Protokolle und die beiden hier vorgestellten Ansätze ergänzen und so eine bessere Zeitsynchronisation für Anwendungen des 'ubiquitous computing' sicherstellen und somit auch zu einer höheren Genauigkeit für kontextsensitive Ansätze führen. Daher ist es das Hauptanliegen dieser Dissertation, das Bewusstsein für die Rolle der Zeitsynchronisation im Umfeld der Kontextsensitivität zu verbessern.

Contents

1 Introduction **1**
1.1 Context Awareness and Ubiquitous Computing 3
1.1.1 Ubiquitous Computing 4
1.1.2 Context awareness 5
1.1.3 Low level and high level context 6
1.2 Problem statement 8
1.3 Challenges 10
1.4 Contribution 11
1.5 Outline 12
1.6 Publications 13

2 State of the art **15**
2.1 Computer clocks and synchronization protocols 15
2.1.1 Network Time Protocol (NTP) 17
2.1.2 Time synchronization in wireless sensor networks 18
2.2 Architectures for context awareness 25
2.2.1 The Context Toolkit 29
2.2.2 CARISMA 31
2.2.3 MiddleWhere 32
2.2.4 Solar 33
2.2.5 Gaia 34
2.2.6 Sentient Object Model 35
2.2.7 Hydrogen 36

2.2.8 RCSM 38
2.2.9 Software framework of Henricksen and Indulska 39
2.2.10 CASS 40
2.2.11 SOCAM 41
2.2.12 Software framework of Korpipää et al. . 43
2.2.13 CAPNET 44
2.3 Quality of context 45
2.4 Human activity recognition 49

3 Reasons for timing issues 53
3.1 Clock drift and possibility of clock adjustment . 54
3.1.1 Experimental setup 56
3.1.2 Results 57
3.1.3 Discussion of results 63
3.2 Construction related differences 63
3.3 Processing load of sensor devices and network sending time 64
3.4 Context processing delays 66

4 Analysis of the influence of time synchronization on reasoning accuracy 69
4.1 Influence on accuracy for user defined rules . . 70
4.2 Influence on accuracy for classification learning 79
4.2.1 Experimental setup 81
4.2.2 Results 83
4.3 Conclusion 101

5 Requirements of an architecture for context awareness 103
5.1 Functional requirements 104
5.1.1 Encapsulation of sensor interfaces 106
5.1.2 Distributed processing and load distribution 107
5.1.3 Dynamic environment support 108

5.1.4 Time synchronization 109
5.2 Software qualities 110
5.2.1 Maintainability 111
5.2.2 Scalability 112
5.2.3 Reliability 114

6 Directed acyclic graph based reasoning 117
6.1 Stream barrier approach 118
6.2 DAG approach 119
6.3 Evaluation 123
6.4 Optimized DAG creation 125
6.4.1 Derive the directed graph 126
6.4.2 Derive the optimum reasoning sequence 127
6.5 Conclusion 129

7 Time locality as a new parameter for Quality of Context 131

8 Conclusion and Outlook 135
8.1 Conclusion 136
8.1.1 Applicability of known synchronization approaches 136
8.1.2 Reasons for timing issues 137
8.1.3 Impact of time synchronization on reasoning accuracy 138
8.1.4 Synchronization approach and new Quality of Context parameter 139
8.2 Outlook 140

Chapter 1

Introduction

When, nearly twenty years ago, Mark Weiser [1] formulated his vision of ubiquitous computing, while today in some parts it is still a vision, his prospects were surprisingly accurate. The only aspect which he was wrong about, is the time when all the technological advances are put together in order to provide a whole new experience of computing. While, today, we have achieved all technological advances Weiser has foreseen, we still need to go the last mile and put all this technology to work in a seamless way.

The advances Weiser has foreseen are the on-going decrease in size of computer devices along with increasing processing power and storage, the advances in display technologies and the increasing number of computing devices virtually everywhere in our lives. More precisely Weiser envisioned smart phones, tablet computers like the iPad, and smart boards. Also Weiser has given an accurate account of the challenges and developments in the software framework and operating system field that are imposed by the technological advances of hardware. If one carefully reads Weiser's visionary paper, one can even find that he hinted to the appearance of cloud computing.

Besides the technological foresight, even more important, Weiser formulated requirements how the technology has to be deployed in order to contribute to our everyday lives instead of interfering with it. Or as Weiser states it, 'the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.' [1]

To achieve this, users must be freed from the burden of the control loop, computer use has to be so simple, that we are no longer aware of it, 'only when things disappear in this way are we freed to use them without thinking and so to focus beyond them on new goals' [1]. The control loop, the constant need for explicit user input, prevents computers from fading into the background. Weiser, thus, demands that computers 'must know where they are' to adapt their behavior [1]. Today this concept has been extended by several authors and builds the foundation of an entire research area called context awareness [2]. The goal of context awareness is that a computing device not only knows where it is, but also knows all kinds of other facts about the environment. Basically, the ultimate goal is that the computer knows what the users wants and acts accordingly on the user's behalf, without the user's explicit input.

Smart phones are the first computing devices that are becoming truly ubiquitous and also are the largest deployed distributed computing platform. To make use of the arising opportunities in the sense of Mark Weiser, it is necessary to also address the software challenges he described. There need to be operating systems, or at least software frameworks on top of the existing operating systems, that support the dynamic, heterogeneous nature of ubiquitous computing. Software developers need to be supported by those frameworks to build applications that can adapt to the user's context. The main contribution of this thesis is a detailed analysis of the requirements that need to be fulfilled by software frameworks and architectures for ubiquitous computing, and in particular for context aware applications running on smart phones.

While there are already many different approaches towards software frameworks for ubiquitous computing and context aware applications published (see Chapter 2.2), there is at least one crucial requirement that is so far overlooked: time synchronization between the devices. We will show in detail where time synchronization is important in ubiquitous computing, why it is not yet solved, how it influences the computer's process of understanding the user's needs, and how software frameworks and architectures can deal with it. Since smart phones are the first truly ubiquitous devices we will especially focus on the use of smart phones in context aware applications.

In the following Chapter 1.1 the research areas important to this thesis and the basic definitions are discussed. Afterwards, In Chapter 1.2 the problem dealt with in this thesis is discussed and in Chapter 1.3 the challenges that arise from this problem are outlined. Chapter 1.4 summarizes the contributions of this thesis. Chapter 1.5 presents the outline of this thesis and in Chapter 1.6 all previous publications of results presented in this thesis are listed.

1.1 Context Awareness and Ubiquitous Computing

As already outlined, this thesis centers on time synchronization in applications that are context aware. Context aware applications are an important part of the vision of ubiquitous computing. Now we will give a more precise description of the central terms of this thesis.

1.1.1 Ubiquitous Computing

The term ubiquitous computing was first used by Mark Weiser to refer to a future era of computing where computers are ubiquitous and weaved into our lives [1]. Weiser has given a detailed description of what he meant by ubiquitous computing but no precise definition. That may be why his vision today is often misunderstood. The basic misunderstanding is what Weiser meant when he said computers will be invisible. Often this is taken as literally invisible. What Weiser really meant was that computer use will be no longer recognized but rather 'fade into the background' [1]. Even though the computing devices may become so small that they are not easily noticed and can be hidden.

What is not clear from Weiser description is, if he sees the evolution of hardware and new usage paradigms as one associated beneficial development or if the evolution of hardware poses a threat that has to be answered by the development of new usage paradigms. In every case, since Weiser formulated his vision, we have seen a fast evolution of hardware technology that is already threatening to the users and thus now we need to answer by bringing Weiser's envisioned usage paradigms to life.

Central to those paradigms is that computer use has to be so easy that it no longer requires thinking or the computer has to be intelligent enough that it no longer requires explicit user input. That way it can 'fade to the background'.

When a computer should fulfill a user's intention without a user's input, the computer has to a certain degree to understand the user. The process of understanding consists of two steps. First, the computer has to know the user's context and then the computer has to know the user's preferences for that context.

1.1.2 Context awareness

The research field dealt with in this thesis is context awareness, which is the first step of understanding a user's intention, and thus is an important part of the vision of ubiquitous computing.

There are several definitions of what context awareness is. The first definitions are very close to Weiser's idea that a computer should know where it is and so focus on the location of users and devices as only context [1, 3, 4]. Later Schilit et al extended this definition with the distinction of categories of context, namely user context, physical context and computing context [5].

A. Dey's definition of context awareness is the definition that is most commonly used [6]:

Definition 1. *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

While this definition is very broad, and thus often used, several authors criticize it as imprecise and for only shifting the abstraction to the word information, which is also very abstract [7, 8]. Liebermann et al thus define context as every input other than explicit input [9]. A detailed discussion of the definitions is given by Sigg [10]

As the many attempts to define context show, the word context is hard to grasp because it is not describing an actual object but rather can be anything. It is an abstract concept describing all that is known at an instance of time at which the process of understanding happens. Only small pieces of information out of all contexts is actually important for the understanding. So what is really meant by 'context' in the context of this thesis is all knowledge that is relevant to a decision a user may make.

Relevant knowledge can be anything a user has learned from past experiences and anything he experiences right at the time he understands a situation, which is what he senses from the environment. Unfortunately only the later can be, more or less easily, accessed be computers. That is done using sensors; environmental (hardware) sensors or software sensors.

Environmental sensors are sensors that sense physical facts from the environment, e.g. light sensors, temperature sensors, acceleration sensors or video cameras. Software sensors retrieve information from the user's software and are thus dependent on previous explicit user input, e.g. if a user is in a meeting can be sensed by a software sensor accessing the user's calendar on his smart phone.

The hope of context awareness is that it is possible to gather enough relevant information from available sensors that a user's intention can be guessed. How relevant information is mapped to a intention is not in the scope of this thesis, but a common approach is to rely on stored rules, that can either be user defined or learned from the history of previous decisions.

When, instead of static desktop machines, computers become tiny mobile devices that incorporate communication and sensing capabilities, due to this sensing capability, these devices provide a measure of context and can share it through wireless communication [11, 12]. In a world of ubiquitous devices, like Weiser envisioned, the control loop in which today's users and their desktop computers are stuck, will no longer be feasible due to the sheer number of devices.

1.1.3 Low level and high level context

The knowledge of context relies on the measurement of sensor data from various sensors, e.g. accelerometers or audio and video data as well as GPS data. This sensor data is aggregated into an abstract high level context, like "the user is standing"; this processing is commonly referred to as context reasoning.

Environmental sensors normally emit what is called raw data that is of no meaning to a human user without further processing. However, because physical quantities are measured, the quantities can be expressed as SI derived units.

The human mind has a different way of accessing the environment; it is not accessing every physical quantity on its own but rather aggregates tons of information and labels categories of information. For example, if a human is asked to describe the weather at a particular time, he will answer something like ”it is a pleasant summer day” instead of ”it is 32° C, with a light intensity of 100,000 lux and a relative humidity of 40 %”. Therefore most authors agree that context from sensors should be aggregated and processed in order to derive ’high level’ context [12, 13].

As of now, there is no clear definition how high level context is distinguished from low level context. We define it as follows:

Definition 2. *Low level context is context that can be expressed as physical quantity with unit or as Boolean value.*

Definition 3. *High level context is context that is described by symbolic labels where the mapping from the set of low level contexts to a label depends on user preferences.*

For example, the low level context set (32° C, 100,000 lux, relative humidity 40 %) may be labelled as ’hot summer day’ for one user but as ’pleasant summer day’ for another user.

The distinction between high level and low level contexts, along with the necessity to process raw context in several steps, in order to derive high level context, will play an important part in the analysis of the causes for time synchronization issues.

1.2 Problem statement

The research conducted for and described in this thesis centers on the necessity for and the current lack of time synchronization support in context aware applications. As of now, context aware applications, which aggregate context data from several sources, either neglect the need for time synchronization or implicitly assume that time on the involved devices is synchronized. However, while the neglect of the need for time synchronization may lead to decreased reasoning accuracy, the assumption that device clocks are in sufficient sync is disproved in this thesis.

The aim of this thesis is to analyze what causes time differences and to evaluate the influence of these differences on the reasoning accuracy for different applications of human movement detection with accelerometers. With directed acyclic graph based reasoning an architectural approach to cope with identified timing issues will be proposed and evaluated in simulation studies as well as practical experiments.

Even if the time on involved devices is synchronized, there are still time related issues. Reasoning accuracy often relies on the aggregation of context data that is gathered at the same time frame. Therefore, not only the clocks need to be synchronized, but also the context gathering needs to be synchronized. Current approaches favor a publish/subscribe model of communication, which is problematic when contexts need to be gathered in a synchronized way.

In common architectural solutions for communication in context aware applications publish/subscribe communication (e.g. [14] and [2]) is used to inform interested parts of a context aware application whenever a context value changes. These values then have time stamps attached, which are taken from the sensor device's clock. However there are four different reasons that cause time differences when retrieving measurements from sensor devices [15]:

- The clocks are not synchronized with the desired accuracy. Wireless Sensor Networks (WSNs) are often associated with low cost sensors that may cause device quality related and other problems, e.g. unstable oscillators, limited energy and limited communication bandwidth [16]. While there are several clock synchronization approaches proposed that can archive sufficient accuracy [16, 17, 18, 19, 20, 21], ubiquitous applications often combine data from several sensors that are not part of a WSN but are rather discovered dynamically, and thus are likely to be not synchronized at all. Additionally, some devices may not offer any means of time adjustment.

- Sensors have different construction-related physical capabilities. In dynamic environments where several sources of sensor data may be dynamically discovered and aggregated, it is not safe to assume that all sensors have the same physical capabilities even if they are of the same general type. Furthermore, sensors of different types clearly have different physical capabilities and different measurement characteristics like sampling rate. Therefore, some sensors may take more time than others to sense a change in the physical environment.

- The sensor nodes have additional tasks to fulfill. For example, smart phones are often equipped with accelerometer sensors and light sensors, however they clearly have additional processor load. Normally, there is no way to influence the scheduling algorithms on the device to ensure measurements at a fixed frequency.

- Common understanding of context reasoning foresees the pre-processing of sensor data in several steps and the inference of high level contexts out of the raw data [15, 22, 23, 24, 25]. These processing steps all consume processor capacity and in distributed scenarios are foreseen to be carried out on different devices with respect to the device capabilities, which consumes time while sending data over the network. Therefore, the time it takes before sensor data reaches the application is not only determined by the sensor itself, but also by the time it takes to process the data. This time is the sum of the time consumed by the processing steps and the network delays and can vary from value to value and sensor to sensor.

In none of the popular architectures for context aware applications the timing issues are considered [22, 23, 24, 25]. As computations on context are of distinct time complexity and context information is frequently updated, a time synchronization mechanism is necessary. Otherwise, a computed context value, which is associated with a single time interval might result from input data of distinct time intervals. Ensuring a synchronized processing might increase the accuracy of reasoning and thus the quality of context [26]. We will show that, for an example of movement recognition from acceleration data, the accuracy decreases in the order of 5 % for commonly used classification learning algorithms and a typical clock drift we observed on smart phones.

1.3 Challenges

Due to the fact that smart phones are the first real world deployment of ubiquitous devices, real world occurrences of context aware applications still are limited to location based services. Therefore, as of now, there is no data available about large scale context aware applications in ubiquitous environments that really aggregate multiple sensors and how time synchronization influences the behavior of context aware applications.

The very nature of ubiquitous computing makes the time synchronization in such environments challenging. Most devices will be small and mobile; thus ubiquitous environments are highly dynamic. Devices will appear and vanish all the time. Also the devices will be as heterogeneous as their owners and their preferences.

In contrast to the well studied wireless sensor networks, which consist of homogeneous, special purpose devices, in a controlled environment, in ubiquitous computing it cannot be assumed that all devices provide unified APIs to access and adjust device time. Additionally, no assumptions should be made about the accuracy and quality of the device's clocks.

At last, the fact that devices will belong to different users may be the biggest obstacle for time synchronization. It must be assumed that every user has different preferences how or if at all the clocks on his personal devices can be adjusted by external entities, which are likely unknown to the user and out of his control. It cannot be excluded that a user intentionally sets his devices' clocks to a wrong time.

1.4 Contribution

This thesis contributes in several ways to the understanding and the solution of the time synchronization issues in context aware applications.

- First of all, known time synchronization approaches are discussed and analyzed for their applicability in context aware applications.

- To grasp the dimension of the issues, the reasons that cause time differences between devices are analyzed and experimentally verified.

- Based on findings about typical time differences from the analysis of the reasons, common reasoning algorithms are tested for their susceptibility to time synchronization related decreases of reasoning accuracy.

- To cope with the time synchronization issues a conceptual approach along with a new architecture for context awareness is proposed and evaluated.

- At last, the incorporation of time synchronization information into Quality of Context information is proposed. Quality of Context is the attempt to rate context data quality, e.g. how likely a context date is accurate. Given the dependency of reasoning accuracy on time synchronization shown in this thesis, information on time synchronization seems to be a natural component of every attempt to predict the accuracy of context reasoning.

Generally, because the important influence of time synchronization on accuracy of algorithms for context recognition is not regarded so far, the main contribution of this thesis is to raise the awareness of the time synchronization issues in the context awareness research community.

1.5 Outline

This thesis is organized in 8 Chapters. Following the first, introductory chapter, the state of the art is summarized. In Chapter 2, the state of the art for four different research areas is regarded, because these four areas relate to the topic of this thesis: computer clocks and synchronization protocols, architectures for context awareness, Quality of Context, and human activity recognition.

Following the state of the art, in Chapter 3, the four major reasons for timing issues are analyzed: clock drift, construction related differences, processing load and network load, and context processing delays.

Based on the analysis of the major reasons for timing issues and the observed typical clock offset and clock drift, in Chapter 4 the dependency of context reasoning accuracy on timing is evaluated.

From the analysis of timing issues and their influence on context reasoning accuracy requirements of an architecture for context awareness are derived. Chapter 5 summarizes these requirements and state of the art requirements as described in Chapter 2.

In Chapter 6 a conceptual approach toward time synchronized context processing is proposed and evaluated. Also, an alternative approach is discussed and compared to the first approach.

Finally, in Chapter 7 a new parameter for Quality of Context is proposed before Chapter 8 concludes the findings of this thesis.

1.6 Publications

Parts of the work conducted for this thesis have been previously published:

- B. N. Klein, S. L. Lau, A Pirali, T. Löffler and K. David, "DAGR - DAG Based Context Reasoning: An Architecture for Context Aware Applications," in *Proceedings of the 2008 Eighth International Workshop on Applications and Services in Wireless Networks (ASWN '08)*, Kassel, Germany, 2008.

- B. N. Klein, S. Sigg, K. David and M. Beigl, "DAG Based Context Reasoning: Optimised DAG Creation," in *Proceedings of Workshop on Context-Systems Design, Evaluation and Optimisation. ARCS 2010 - Architecture of Computing Systems,* Hannover, Germany, 2010.

- B. N. Klein and K. David, ”Time Locality: A Novel Parameter for Quality of Context,” in *Proceedings of Seventh International Conference on Networked Sensing Systems (INSS '10),* Kassel, Germany, 2010.
- B. N. Klein and K. David, Basic Approach of Timing in Context Aware Architectures verified by concrete Advantages,” in *Proceedings of The 10th Annual International Symposium on Applications and the Internet (Saint '10),* Seoul, South Korea, 2010.
- B. N. Klein, S. L. Lau and K. David, ”Evaluation of the Influence of Time Synchronization on Classification Learning Based Movement Detection with Accelerometers,” in *Proceedings of The 11th IEEE/IPSJ International Symposium on Applications and the Internet (Saint '11),* Munich, Germany, 2011.

Chapter 2

State of the art

This thesis contributes to and builds on the research of four different areas of research in the broader field of context awareness and mobile computing. In the following the major results of these areas and how they relate to this thesis is summarized.

2.1 Computer clocks and synchronization protocols

According to [27] *time* is an abstraction to determine the ordering of events in a given *timescale*. The International Standard (SI) definition of the *time interval* second is "the duration of 9.192.631.770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium^{-133} atom."

An *oscillator* is a generator with precise frequency within a specified tolerance. This tolerance is expressed in parts-per-million (ppm). A *clock* is an oscillator together with a counter that records the number of electrical oscillations. The value of this counter at any given time is called *epoch*, which is recorded as the *time stamp* of that epoch.

Today's electronic clocks are based on the convenient properties of quartz (silicon dioxide). Quartz, when properly shaped and mounted, will oscillate if it is triggered by an alternating current. Formed into a small tuning fork quartz oscillates at 32.768 Hz or 2^{15} cycles per second. Counting these cycles allows to generate a time of day.

Most hardware clocks are based on this principle, whether it may be a wrist watch, clock in a desktop computer or a clock in a smart phone. Quartz clocks are sensitive to temperature, pressure and other factors. For example the clock on Intel 'ICH' chip based motherboards can have a loss of up to 3 seconds per day at a temperature of 60° C [28].

In desktop computers the hardware clock is backed by a battery and thus runs whether the computer is on or off. Modern operating systems almost never use the hardware clock, instead, at boot time they initialize a software clock or system clock with the hardware clock's time. This system clock is implemented as a counter that is increased with every timer interrupt. On linux systems, the kernel measures the time in jiffies; the size of a jiffy is determined by a kernel constant. On i386 machines, since kernel 2.6.20, a jiffy is 4 ms [29].

Clocks have a certain *stability*, which is 'how well it can maintain a constant frequency', and clocks have a certain accuracy, which is 'how well its time compares with national standards. ... The time offset of clock i relative to clock j is the time difference between them $T_{ij}(t) \equiv T_i(t) - T_j(t)$, while the frequency offset of clock i relative to clock j is the frequency difference between them $R_{ij}(t) \equiv R_i(t) - R_j(t)$.' [27]

Since oscillators cannot maintain a precise frequency with infinite accuracy, the offset between two clocks will always increase over time, which is called *clock drift*. In order that multiple computing devices can work together, the synchronization of their clocks is necessary for many applications, such as context aware applications (which will be shown in Chapter 4).

To synchronize clocks they have to be compared in time and frequency to some reference clock and then time and frequency have to be adjusted accordingly. There are several approaches to synchronize clocks in computer networks to some reference time (e.g. Coordinated Universal Time, UTC). Which approach to use depends on network characteristics. In the following, the most important approaches will be discussed.

2.1.1 Network Time Protocol (NTP)

The most commonly used synchronization protocol in today's Internet is the Network Time Protocol (NTP). NTP really is a set of protocols used to synchronize clocks between peers as well as to organize and maintain a network of time servers. NTP is built on IP and UDP.

Time servers are organized in primary and secondary servers. Primary servers synchronize directly to external time sources, e.g. radio clocks. Secondary time servers synchronize with the primary server or other secondary servers. How many hops from a primary server a secondary server is, is marked by a stratum number. A primary server has stratum one, a secondary server that synchronizes with a stratum one server has stratum two.

Basically, to synchronize two peers P_1 and P_2 with NTP, they exchange NTP messages containing the latest three time stamps T_1, T_2 and T_3. When a message arrives at P_2, $T1$ is the time stamp P_2 has taken before sending the message to P_1, T_2 and T_3 are time stamps from P_1. T_2 is taken when the message from P_2 arrives and T_3 before the message back to P_2 is send. P_2 takes a fourth time stamp T_4 and thus can calculate the offset and delay as follows (see fig. 2.1).

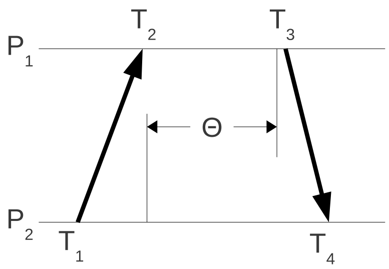


Figure 2.1: NTP basic delay and offset calculation.

Assume that delays from P_1 to P_2 and from P_2 to P_1 are similar, than the delay δ and offset θ of P_2 relative to P_1 are close to $\delta = (T_2 - T_1) - (T_3 - T_4)$ and $\theta = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$ [27].

NTP uses several additional algorithms to secure an accurate synchronization, e.g. clock-selection algorithms to select peers that are likely to provide most accurate time. According to [30], at least 30 % of NTP time servers maintain a time accurate to about 30 ms, while some few servers are off up to 10 seconds.

2.1.2 Time synchronization in wireless sensor networks

In wireless sensor networks (WSNs) time synchronization is one of the basic middleware services. Sensor fusion as well as other coordinated actions requires proper time synchronization. While applications in WSNs often require higher clock accuracy than possible with NTP (often in the order of microseconds), time synchronization in WSNs has to cope with several additional challenges. First of all, in WSNs there commonly is a limitation of energy and other resources and thus time synchronization protocols have to be energy efficient. Second, in WSNs there are several nondeterministic delays in message delivery, which affect the achievable accuracy of protocols like NTP. In the following chapter these nondeterministic delays are decomposed.

Uncertainties in WSN message delivery

According to [31] there are the following ten sources of message delivery delays:

1. *Send time* – time is takes on the sender's side to assemble a message and issue a send request to the MAC layer. This time is nondeterministic because it depends on the current processor load and the implementation of the system call.
2. *Access time* – time is takes on the sender's side to access a transmission channel. This time is the most nondeterministic part, depending on the technology and current network traffic it can take up to seconds.
3. *Transmission time* – time it takes to actually transmit the message. This time depends on the radio speed.
4. *Propagation time* – time it takes before the message reaches the receiver when it has left the sender. This time depends on the node distance and is in the order of microseconds and therefore can be disregarded.
5. *Reception time* – time it takes for the receiver to receive the message. It is the receiver's correspondent of the transmission time.
6. *Receive time* – time to process the message on the receiver's side and notify the applications. It is the receiver's correspondent of the send time.
7. *Interrupt handling time* – time between the raising of an interrupt by the radio chip and the controller responding to it. This time contributes to the reception time.

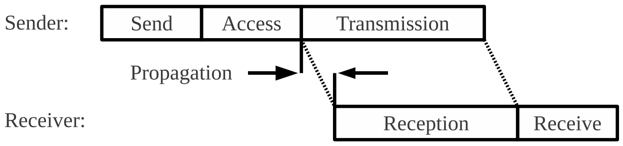


Figure 2.2: Decomposition of the message delivery delay over a wireless link [31].

8. *Encoding time* – time it takes the radio chip on the sender's side to encode and transform a message to radio waves. This time contributes to the transmission time.
9. *Decoding time* – time on the receiver's radio chip to decode a radio wave to a binary representation of the message. This time contributes to the reception time.
10. *Byte alignment time* – time it takes to shift the message bytes on the receivers side in case the radio chip cannot capture the byte alignment. This time contributes to the reception time.

Figure 2.2 shows how the message delivery delay over a wireless link is composed of the first six sources of delay. Sources 7 - 10 are part of the transmission and reception. Table 2.1 shows the typical magnitude and the distribution of this delays measured on the Mica2[1] platform by Maróti et al. [31].

In the following, three common approaches to time synchronization in WSNs that strive to minimize the errors incurred by the ten uncertainties are discussed.

Reference Broadcast Synchronization

The Reference Broadcast Synchronization (RBS) [18] is a protocol that is not intended to do sender-receiver time

[1] `http://www.xbow.com/Products/Wireless_Sensor_Networks.htm`

Time	Magnitude	Distribution	Depending on
Send and Receive	0–100 ms	non-deterministic	processor load
Access	10–500 ms	non-deterministic	channel contention
Transmission / Reception	10–20 ms	deterministic	message length
Propagation	$<$ 1 µs for distances up to 300 meters	deterministic	distance
Interrupt Handling	$<$ 5 µs in most cases, but can be as high as 30 µs	non-deterministic	interrupts being disables
Encoding and Decoding	100–200 µs, $<$ 2 µs variance	deterministic	radio chipset and settings
Byte Alignment	0–400 µs	deterministic	can be calculated

Table 2.1: Sources of message delivery delay on the Mica2 platform, according to [31].

synchronization to synchronize clocks to an external reference time source (but can be extended to synchronize clocks to an external reference time source). Instead RBS enables receiver-receiver synchronization. That is RBS synchronizes a set of receivers with one another to achieve a relative synchronization of the receivers' clocks.

In RBS nodes synchronize with their neighbors by periodically sending synchronization messages. These messages are send utilizing the physical layer broadcast and do not contain explicit time stamps. Instead receivers use the arrival time of the periodical messages as reference points.

By using the physical layer's broadcast capability, RBS eliminates two of the error sources described in Chapter 2.1.2, the access time and the send time. Thus, the main influencing factors on synchronization accuracy that can be achieved using RBS are propagation time and receive time. The basic scheme RBS uses to estimate clock offset of two receivers i and j is [18]:

1. A node broadcasts a reference message to receivers i and j.
2. Receivers i and j record their local time upon arrival of the reference message, t_i and t_j.
3. Receivers i and j exchange their locally observed arrival times and thus calculate their relative clock offsets $\theta_i = t_i - t_j$.

To compensate for the remaining uncertainties, in RBS receivers calculate an average offset from multiple reference messages. To estimate the clock drift instead of averaging the offsets, linear regression can be used.

The main limitation of RBS is its dependency on the availability of physical layer broadcast. The main strength of RBS is that it does not require low level OS access and therefore is applicable on a wide range of hardware. The authors of [18] claim RBS to be two times more accurate than NTP.

Timing-sync Protocol for Sensor Networks

Compared to RBS, the Timing-sync Protocol for Sensor Networks (TPSN) [32] is a sender-receiver synchronization protocol that is used to synchronize a multi-hop WSN to an external reference time source. TPSN is less computational complex than NTP.

TPSN creates and maintains a hierarchical topology in the network. Every node in this structure is assigned a level. The node connected to an external reference time source acts as the root node and has level 0. For other nodes the level is defined as the length of the path from that node to the root node.

Once TPSN has established this structure the synchronization phase begins. Starting from synchronizing nodes of level 1 with the root node, nodes of level i are synchronized with nodes of level $i - 1$.

The synchronization of two nodes is a sender-receiver synchronization, calculating offsets by message exchange and estimating delays from the round trip time in the same way as NTP does (see chapter 2.1.1).

TPSN uses time stamping at MAC layer and in [32] the authors argue, that this way the uncertainties are reduced to the transmission time. TPSN has an advantage over RBS if there is a close coupling between radio and application layer (as there is for many small sensor nodes), because then instrumenting the MAC layer eliminates the uncertainties at the sender side, like RBS, but still has the benefit of round trip time measurements to estimate the propagation time. The shortcoming of TPSN is its lack of clock drift estimation and compensation.

	Send	Access	Transmission (incl. encoding)	Propagation
RBS	No	No	No	No
TPSN	No	No	Yes (estimating)	Yes (estimating)
FTSP	No	No	Yes (estimating)	Yes (estimating)

Table 2.2: Error sources before a synchronization message reaches the receiver side. The table shows which protocol is prone to which message delay reason. If a protocol is prone to a message delay reason, the table shows if the protocol estimates the delay.

Flooding Time Synchronization Protocol

The Flooding Time Synchronization Protocol (FTSP) [31] is a sender-receiver synchronization protocol, like TPSN, that utilizes periodic flooding of synchronization messages and a dynamically updated topology. It uses MAC layer time stamping like TPSN and additional error correction mechanisms including clock drift estimation.

Table 2.2 and Table 2.3 show which protocol is affected by which uncertainties. In contrast to TPSB and RBS, FTSP does not suffer from time stamping uncertainties at the MAC layer, that is jitter of interrupt handling and decoding time (see Chapter 2.1.2).

This is achieved by using radio broadcast like RBS, but including an explicit time stamp, which is the estimated global time at transmission of a given byte. The receivers record the local time of the reception of that given byte. The jitter is reduced by recording a time stamp after each byte boundary of a packet after the SYNC bytes on sender and receiver side.

	Reception			
	Interrupt handling	Decoding	Byte alignment	Receive
RBS	Yes	Yes	Yes	No
TPSN	Yes	Yes	Yes	No
FTSP	Yes (estimating)	Yes (estimating)	No	No

Table 2.3: Error sources when the signal arrives at the receiver. The table shows which protocol is prone to which message delay reason. If a protocol is prone to a message delay reason, the table shows if the protocol estimates the delay.

This time stamps than are normalized and the minimum is taken as estimate of the interrupt handling time error. The average of these normalized and interrupt handling time error corrected time stamps is used to reduce the jitter of the encoding and the decoding time. Linear regression is used to estimate clock drift.

In Chapter 3 the applicability of the discussed synchronization protocols will be analyzed. But first we will give an overview of architectures for context awareness in order to analyze if and how time synchronization is done in these architectures.

2.2 Architectures for context awareness

The architectures discussed in this chapter, as all software architectures, try to fulfill a set of functional and non-functional requirements. Functional requirements are requirements directly related to the function of the software. Non-functional requirements ”constitute the justifications of design decisions and constrain the way in which the required functionality may be realized" [33].

In [33] it is noted that there are different well known classification schemes for non-functional requirements, but that even these schemes are inconsistent. Out of the long list of non-functional requirements mentioned in the literature the following are especially important for context aware architectures:

- Performance – The ability to fulfill the system's task fast and uninterrupted.
- Scalability – The ability to fulfill the system's task without reduced performance even if the number of fulfilled tasks simultaneously increases.
- Reliability – The ability to correctly fulfill the system's tasks with stable performance.
- Maintainability – The possibility to change and extend the system while using relatively few development resources.
- Reusability – The possibility to reuse the system or parts of it without the need to change large portions of the system.

Even if these requirements are especially important for context aware architectures, they are not specific for context aware architectures. Nevertheless, the authors of the works discussed in this chapter tend to characterize these non-functional requirements as specific for context aware architectures and therefore refuse to reuse the solutions provided by established middleware approaches and communication protocols.

The authors of context aware architectures seem to agree on a set of challenges that make the fulfillment of the functional and non-functional requirements difficult. These challenges are rooted in the dynamic, heterogeneous and distributed nature of context aware applications:

- Changing connectivity – Context aware applications usually involve several sensors, either located in the user's environment or integrated in the user's personal devices, like the user's smart phone. To not limit the user's mobility and to achieve the invisibility of ubiquitous computing, at least the user's personal devices, if not all involved devices, have to use some kind of radio connection to communicate with other devices. Loss of connection is an inherent characteristic of radio connections as well as changing bandwidth.

- Dynamic resource availability – Directly dependent on the changing connectivity and the user's mobility, environmental devices visible to the user's personal devices will appear and disappear all the time.

- Multitude of different sensor interfaces – Sensor devices will be produced by numerous vendors for a multitude of purposes. It cannot be assumed that all these devices follow some standard interface definition or protocol.

- Different sensor characteristics – In the same way as the multitude of sensors will have different interfaces, they will have different characteristics, like different accuracy, different sampling frequency, different reliability, and so forth.

- Limited resources – The goal of almost invisible but ubiquitous device availability makes it inevitable that device size is limited and therefore the processing power and storage capacity will be limited. Furthermore, mobile devices are dependent on batteries and that will limit the processing power even more, as well as it will limit communication bandwidth.

From these challenges functional requirements are arising. As for the non-functional requirements, the functional requirements are addressed with specific solutions by most authors, rather than using established technology. The functional requirements are:

- Encapsulation of heterogeneous sensors/ transparency – The two challenges *different sensor interfaces* and *different sensor characteristics* demand the central functionality of architectures for context awareness, the encapsulation of the sensor interfaces. Once encapsulated and then providing a uniform interface, the sensors can be easily used by application developers, even in graphical development approaches.

- Communication model – The challenge of *changing connectivity* needs to be addressed by a scalable communication model with sufficient performance. Generally, depending on the application's needs and the sensor's characteristics, either publish/subscribe or polling approaches are necessary.

- Storage and History – Seldom a single sensor value is of any significance to an application, normally the distribution and trend of the values is used to get more significant information. Therefore, context values have to be stored and the storage has to provide fast and scalable access to the history of the values. For the prediction of future context the history is needed to extract patterns and trends.

- Resource discovery – The challenge *dynamic resource availability* has to be addressed by providing a resource discovery mechanism, which allows querying for available devices.

- Context processing/ data flow – To enable the calculation of high-level context, the architecture approach has to offer a way to define a data flow of context processing steps, where several sensors are aggregated. Facing the challenge of *limited resources* it should be possible to distribute the processing over the available resources. The architecture should handle this in a transparent way.
- Rule engine – In order that applications don't have to hard code how to react to context changes, it is beneficial to define rules that define what actions to trigger when specified conditions are met. These rules can have simple forms, like first-order logic, and be user defined, or else can be learned using machine learning techniques. In any case the task of the rule engine is to observe the context sources and evaluate corresponding to the rules.
- Security – Since context aware applications are envisioned to be ubiquitous and thus will steadily affect our everyday life, it is important that these applications are secure and cannot easily be manipulated by third parties.
- Privacy – Context data is by definition of high interest to the user and usually taken from the user's devices or from devices in the user's proximity. Thus the data is the user's private data and has to be protected.

In the following, the important context aware architecture approaches are discussed.

2.2.1 The Context Toolkit

Dey et. al. proposed the Context Toolkit [6, 2] as one of the first middleware approaches to enable the development of context aware applications. Their approach is inspired by the widget concept known from GUI toolkits and thus they foresee the delivery of building blocks to build context aware systems more easily.

The Context Toolkit's main goals where to overcome the difficulties imposed by the distributed nature of context aware applications and the heterogeneity of the used sensors. In detail in [34] four major difficulties are identified:

1. Context information may be acquired from unconventional sensors. While in 1999 the authors claimed GPS to be an unconventional sensor, today there are still new unconventional sensors occurring. For developers of context aware applications a middleware should abstract from the specifics of those sensors and enable the use in a standard way, i.e. a well know application programming interface (API).

2. Raw sensor information has to be abstracted to make sense. That is, for example an acceleration sensor reading may not have any meaning to the developer at all, while a high level context like 'running' can be much more easy to use.

3. Context may be acquired from multiple distributed and heterogeneous sources. Distribution and heterogeneity are a major challenge for every software system and thus it is a basic middleware service to cope with both.

4. Environmental changes must be detected in real time and the history of context is valuable.

The widget concept has three main benefits that help to cope with the difficulties of context acquisition:

1. Hiding specifics/ complexity of used sensors.

2. Managing details of the abstraction of context information.

3. Providing reusable building blocks that can be combined like GUI widgets to compose context sources.

A context widget, as a building block, allows applications to register to be notified in case of a context change. Widgets are built from their more basic components that can also be combined: generators, interpreters and servers. Generators are the components really acquiring the context from a sensor, interpreters abstract and filter context, and servers aggregate and store context information.

The Context Toolkit achieves platform independence by using a communication model entirely based on HTTP and XML message exchange. Applications can both subscribe to a widget and poll a widget. If the subscription mechanism is used, applications can specify conditions to prevent unnecessary notifications.

Even though the need for real time context information is acknowledged by Dey et. al. [6, 2], there is no architectural support to enable time synchronization or any other facilities needed to enable real time processing of context.

2.2.2 CARISMA

The Context-Aware Reflective Middleware System for Mobile Applications (CARISMA) [35] is a middleware approach that enables the development of mobile applications that react to context changes. However, the context considered by CARISMA is limited to resource availability, e.g. memory, bandwidth and battery power. Compared to the Context Toolkit, the context is not necessary for the applications to operate, it is merely influencing the way an application is carrying out a specific task. For example depending on the available bandwidth a message is transmitted as text, voice or video.

Generally, it is the purpose of a middleware to hide these specifics to the application (transparency). Carpa et. al. argue that an application may have valuable information and thus CARISMA exposes an abstraction of the middleware state to the application and enable the application to change the middleware behaviour through reflection.

There is no mention of time synchronization support for the distributed mobile devices CARISMA handles.

2.2.3 MiddleWhere

MiddleWhere [36] is a middleware approach that enables ubiquitous computing applications to incorporate location information. Thus, in MiddleWhere the only context considered is the location of a user or object. MiddleWhere specifically supports the fusion of location information from different sources that may have different resolution and confidence. As none of the fused sensors is capable of giving an absolute accurate location information, the location information is associated with a probability value.

MiddleWhere has the following properties:

1. Fusion of multiple location sensing technologies. MiddleWhere fuses data from different sensor sources with different accuracy and resolution. This information may be conflicting, thus MiddleWhere calculates a spatial probability distribution.

2. Handling, temporal nature of location information. Location information is only valuable if it is fresh. MiddleWhere therefore reduces the confidence in location information as the information gets older.

3. Hybrid location Model. MiddleWhere supports coordinate as well as symbolic location models.

4. Push and Pull interaction. Applications can actively pull for location information as well as to register to be informed in case of location changes.

5. Region based and object based locations. MiddleWhere can keep track of the location of objects and which objects are located in a particular region.

6. World model. MiddleWhere keeps a spatial database to model the physical world.

7. Spatial relationships between objects. MiddleWhere can express whether an object is in close proximity to another object or if an object is collocated with another object.

Even if Ranganathan et. al. recognize the temporal nature of location information, MiddleWhere's only tribute to that is the reduction of confidence in location information over time. There is no mechanism that ensures correct time information and handles conflicts originating from different and inaccurate time stamps. However, all error and conflict resolving calculations proposed in [36] rely on correct time information.

2.2.4 Solar

Chen et. al. propose Solar [22] as a general purpose platform for handling of context processing, that is, not only a special context like location is considered but all kinds of context information. Solar foresees raw sensor data to be processed to get high level context. In particular context data can be filtered, aggregated, and abstracted.

As a middleware Solar provides middleware services to *collect* raw data from a multitude of sensors, *process* the raw data and *disseminate* the high level context to many applications. Solar strives to enable scalable, secure, and privacy preserving context processing. Therefore, the communication model of Solar is based on the subscription of context event streams. Solar provides an attribute based naming scheme for event sources to enable the discovery of event sources.

Event sources can be sensors as well as what Solar calls operators, which can filter, aggregate, and further process context and have a similar interface as sensor sources. Thus, event streams can be combined in a flexible way to process context in several steps and reuse operators. One event stream that may be subscribed by an application is the leaf of a tree structure of operators.

To not only reuse the operators at design time, but also save processing power, Solar can combine the tree structures into an acyclic directed graph to reuse tree parts that otherwise will be needed twice. Applications can define tree structures in an XML based language and Solar will instantiate the needed tree to provide the context event stream to the application.

In a Solar deployment there are two kinds of devices, a *star* and multiple *planets*. The star handles the deployment of trees to the planets and the subscriptions of the applications. A planet will run multiple operators. It is the task of the star to distribute the operators equally to the planets and thereby consider the available processor load and bandwidth. Sensors and applications themselves are not part of the Solar system. How time synchronization is handled by Solar is not mentioned by the authors.

2.2.5 Gaia

The Gaia middleware is a middleware approach to enable the application development for smart homes [37], where applications have to cope with a distributed, heterogeneous environment of networked devices that may appear and leave in a highly dynamic way. Therefore, Gaia is based on the principle structure of operating systems. It consists of a core, that handles the dynamic life cycle of the other Gaia components, and on top of that core there are the following basic services:

- Spaces Repository Service. A repository that stores information about the devices and software in a physical space (e.g. a room).
- Event Manager. The Event Manager provides a publish/subscribe communication infrastructure.
- Context File System. A storage for context information that has the hierarchical organization of file systems.
- Presence Service. This service discovers devices that are present in a physical space.
- Context Service. This service allows applications to query for context information.

Additionally to the basic components Gaia provides an application framework that is envisioned to ease the development of applications for smart spaces. The framework is based on the Model-View-Controller pattern and provides mobility, adaption and dynamic binding of applications. The distributed communication is handled using CORBA. There is no time synchronization service described for Gaia.

2.2.6 Sentient Object Model

Biegel and Cahill present the Sentient Object Model in [38]. The Sentient Object Model is a programming model and a framework to enable the development of context aware applications. The authors emphasize the close coupling of objects, which they claim is beneficial in dynamic mobile environments, and the uniform interface of the so called sentient objects.

A sentient object is an entity encapsulating the following functionality:

- Abstraction of sensors and actuators. The developer is relieved of the error prone handling of low level APIs of various hardware devices.
- Probabilistic sensor fusion mechanism including filtering of sensor events. The sensor fusion is based on Bayesian Networks to model uncertainties and dependencies between sensors.
- Rule based reasoning. The developer can specify event-condition-action rules and the inference engine can identify relevant context information out of a context hierarchy and thus can be more efficient.

Thus, a sentient object is an autonomic object that is aware of its context and can act proactively upon it. Sentient objects can communicate with one another using an event based mechanism. Since sentient objects provide a uniform interface, whether they are sensors or actuators or are aggregating other sentient objects, they can be easily combined using a visual interface. All event filtering and rule based reasoning is hidden to the other objects and handled by an internal control logic. Time synchronization is not regarded.

2.2.7 Hydrogen

The Hydrogen Context framework [39] is an architectural approach intended to enable the development specifically on mobile devices such as smart phones. Hofer et al. argue that mobile devices pose the following problems that need to be addressed by context frameworks, additionally to the general purpose of context frameworks like separation of concerns, encapsulation of sensor APIs and interoperability in a heterogeneous environment:

- Limitation of network connections. Mobile devices may have limited bandwidth available, e.g. due to higher traffic costs, and connections may suffer frequent disconnects.

- Limited computing power. While development of processing power and memory capacity of mobile devices seems to follow Moore's law they are still not as powerful as desktop computers or servers. Additionally, mobile devices are limited by available battery power, which directly influences the processing power.

Based on these limitations of mobile devices Hofer et al. formulated five requirements they see for a context framework:

1. Lightweightness. The architecture has to use the available resources on mobile devices economically.

2. Extensibility. In case of Hyderogen extensibility means the ability to connect to remote sensor sources.

3. Robustness. The architecture has to be robust against disconnects.

4. Meta-Information. Context information, especially from remote sensors, has to be enriched with information about the location and precision of the sensor.

5. Context-Sharing. The architecture has to allow the sharing of sensor data with remote devices.

Hydrogen addresses these requirements with a three-layered architecture. The basic layer, called the adaptor layer, handles the retrieval of sensor data from different sensors. This sensor data is then handed to the second layer, the management layer, which basically stores the context and provides access to the data for local and remote applications. Local applications reside in the top layer, the application layer. The applications can query contexts from the management layer or subscribe to context information. Again, time synchronization is not regarded.

2.2.8 RCSM

The Reconfigurable Context-Sensitive Middleware (RCSM) [25] is a middleware approach for pervasive computing applications. Yau et al. found two main characteristics of pervasive computing applications, context sensitivity and ad hoc communication. RCSM addresses these two characteristics and additionally supports the development of applications that have both characteristics at the same time.

In general one basic feature of middleware is transparency, which is to hide specifics from the applications, e.g. a middleware for database access hides the specifics of accessing databases from different vendors. In contrast, pervasive applications need a balance between transparency and context awareness. It is a major goal of RCSM to provide this balance. Therefore, RCSM has the following main features:

- Object oriented development framework. RCSM provides an object oriented framework like middleware approaches for programming language independent interprocess communication do (e.g. CORBA). Objects of RCSM consist of two parts, the object implementing the context independent functionality in any programming language and a context sensitive interface. The context sensitive interface is specified in a meta language and compiled by RCSM into a runtime container object for the context independent object. The meta language can be used to specify the contexts that should be considered (subscribed to) and which actions (functions of the context independent object) should be triggered.

- Object request broker. RCMS's object request broker handles ad hoc communication and service discovery. It provides platform independent interfaces that hide the intricacies of specific transport protocols. Generally it follows the concepts of CORBA.

The gathering of context from sensors sources, further processing of context, and time synchronization is not covered by RCSM.

2.2.9 Software framework of Henricksen and Indulska

Henricksen and Indulska [23] presented a software framework for context aware applications along with a context modeling language and a way to formally express user preferences. This software framework supports two programming models. The first supported programming model is branching, which is based on the user preference specifications. Applications select actions based on ratings assigned to a set of alternatives by the preference model. The second programming model is triggering, which basically is a publish/subscribe communication model.

The software framework that enables the two programming models is divided into six layers from bottom to top:

1. Context gathering layer. On this layer the context from sensors is gathered and processed to higher level

representations. This layer is connected over an event notification scheme with sensors and the layer on top.

2. Context reception layer. This layer provides a bidirectional mapping from the gathered context to a context model.
3. Context management layer. This layer is a repository of context models; it can be distributed.
4. Query layer. This layer provides an interface for applications and the adaption layer to query the management layer for context information.
5. Adaption layer. The adaption layer is a repository of preferences and event trigger specifications. It also evaluates the specifications using the query layer.
6. Application layer. This layer provides a toolkit to applications that enables the use of the branching and the triggering programming models.

The specifics of the context gathering layer, and thus time stamp retrieval, are not detailed in [23].

2.2.10 CASS

In [40] Fahy and Clarke present the Context-Awareness substructure (CASS) middleware. CASS's main design goals are the support of high-level context inference and the separation of context inference and context based behaviors from application code. To enable the handling of a large number of sensor nodes and to overcome the memory and processor constraints of mobile devices. CASS is a server based approach. This server based approach comes at the cost of the need for permanent network connections to the server. Fahy and Clarke state that they solve this problem with local caching, however that would lead to old and maybe deprecated context information on the mobile devices.

The requirements Fahy and Clarke identify for CASS are:

- Supporting a large number of sensors and other context sources.
- Storing the context history.
- Supporting context interpretation and higher-level abstraction.
- Event based communication model.
- Extensibility.
- Transparent use of distributed sources.
- Separation of context inference and context based behaviors from application code.

On the CASS server there is a database to store the context as well as user data, application data and domain knowledge. This database based approach offers the possibility to use SQL to query context information. Additionally, the database also stores rules and the context interests of applications.

Also on the server there is an inference engine that uses forward caching to infer new facts from known facts and rules. The rules are contained in a knowledge base and define action goals that should be triggered when the specified high-level contexts occur. Time synchronization is not regarded by the authors.

2.2.11 SOCAM

The Service-oriented Context-Aware Middleware (SOCAM) [41] is a service-oriented middleware approach based on

OSGi, which is built for rapid prototyping of mobile context-aware applications. In SOCAM contexts are represented as predicates in OWL. The components of SOCAM are:

- Context Providers that encapsulate the APIs of the heterogeneous sensors. Context providers have to register at a service registry, so that they can be discovery by other services. Context providers publish events in form of OWL descriptions when the contexts are changing.

- Context Interpreters are special context providers that are providing high-level contexts. They contain a knowledge base and use different context reasoners, as for example RDFS reasoners or OWL reasoners. The knowledge base provides an API to let others services query, add, delete, and modify contexts.

- Service Location Service where context providers and context interpreters can register and then can be found by applications.

- Context-Aware Mobile Services are applications that can either query context providers for context or register to an event notification scheme to be informed of context changes. Users can specify a set of rules in first order logic that express when to trigger a services action.

SOCAM, unlike the most other approaches, achieves middleware functionality that is common in distributed environments, like network communication abstraction, by using the OSGi functionality and concentrates on the context-aware specific middleware services, however, without regarding time synchronization.

2.2.12 Software framework of Korpipää et al.

In [42] Korpipää et al. present a context management framework that is targeted at dealing with noise, faulty connections, drift, miscalibration, wear and tear and other factors causing uncertainties when fusing sensor data. As a communication model a blackboard approach is used. On top of the Symbian OS Korpipää et al. implemented a middleware with four main components:

- Context Manager that is a central server component, even if it runs locally on the mobile device, where the other components act as clients. The context manager is blackboard based and stores all context information. The clients can either query for context, subscribe to low level contexts or use high-level contexts.

- Resource Servers connect to sensors and other context sources and post the contexts to the context manager's blackboard. The resource server's task is to provide context abstractions to higher levels that are useful and have low enough frequency to not stress the system. There are four phases of context processing taking place in the resource servers. First, measuring the raw data. Second, pre-processing a set of data from a time interval and calculate generic features. Third, extracting specific features and fourth, quantizing and semantic labeling of the data. The quantizing and semantic labeling is done using fuzzy sets or crisp limits.

- Context Recognition Services can register at the context manager to share high-level contexts.

- Applications that are served with context information in an event based manner.

The approach uses naïve Bayes classifiers as a supervised learning approach to classify contexts because of the computational efficient context classification that is robust to uncertainties. Korpipää et al. recognize the importance of time information to relate context events and also the potential influence of drift, but except for the claim that the naïve Bayes classifiers are robust, they do not provide any detail if they foresee any time synchronization.

2.2.13 CAPNET

The Context-Aware Middleware for Mobile Multimedia Applications (CAPNET) [43] combines middleware services for context-awareness and middleware services that support multimedia services, such as storage, capturing, rendering and adaption of media for different resource limited mobile devices. Along with several requirements for the multimedia capabilities, CAPNET fulfills the following requirements of context-aware middlewares:

- Support for a variety of sensor devices.
- Support for distribution of sensors
- Transparency of context processing
- Context storage
- Control of context data flow

The CAPNET middleware layer is located on top of a layer of existing technologies that offer remote procedure calls, database services and service orientation. The CAPNET middleware includes components for service discovery, component management, user interface, multimedia and context. The task of the context component is to abstract the different sensor device interfaces to a uniform interface, to store the context and to deliver context information by means of an event notification scheme. Time synchronization, again, is not mentioned.

None of the approaches discussed in this Chapter regards time synchronization, therefore, in this thesis we analyze the relevance of time synchronization for context awareness and propose an approach towards time synchronization for context awareness. Additionally, adding information about time synchronization into quality of context is proposed in Chapter 7.

2.3 Quality of context

An integral feature of context is its uncertainty. Sensors are subject to failure and noise, also processing of sensor data is a process involving approximation and that may cause ambiguity. Thus, it is important for applications receiving the context data to also receive meta information additional to the actual value.

Gray and Salber propose a notion of quality of context (QoC) [44]. QoC, in their definition, is an information about the quality that consists out of six single parameters:

- Coverage – The amount or range of data that can be sensed, e.g. for a temperature sensor this will be the temperature range like $-10°$ C - $50°$ C.
- Resolution – The smallest perceivable element, e.g. for a temperature sensor the resolution may be $0.5°$ C.
- Accuracy – The range of deviation, e.g. for a temperature sensor the accuracy may be $\pm 0.1°$ C.
- Repeatability – How likely will the sensor deliver the same measurement, if the physical circumstances are exactly the same.

- Frequency – The sample rate.
- Timeliness – The temporal accuracy; range of potential delay of delivery.

Later, Buchholz et al. proposed a slightly different definition where they name the, in their eyes, five most important parameters but emphasize that there may be many more parameters, depending on the specific context [45]. The five parameters named by Buchholz et al. are:

- Precision – The precision is the same as accuracy in the definition by Gray and Salber.
- Probability of Correctness – Denotes the probability that a single context information is correct, based on previous behavior of the sensor hardware. This parameter is related to the repeatability parameter in the definition by Gray and Salber.
- Trust-worthiness – Like the probability of correctness this parameter describes how likely it is that a context information is correct, but in this case based on the assessment of the context provider's personal trust-worthiness or potential harmful tendency.
- Resolution – The resolution is the same as in the definition by Gray and Salber.
- Up-to-dateness – This parameter describes how old a context information is. That is different from timeliness in the definition by Gray and Salber. Timeliness describes the range of potential delay before a context source delivers a context information, up-to-dateness how old a single context value actually is.

Additional to the refinement of the parameter definition, Buchholz et al. argue different reasons why QoC is needed. They see a need to use QoC in service level agreements additionally to QoS if context information delivery is the purpose of the service. If context aware applications have a choice between different context sources to acquire the desired context, they should be able to base their choice on QoC parameters. Also the composition of multiple context sources to generate high-level context information is dependent on the QoC of the fused context sources. Caching of context information may affect the QoC, especially parameters like up-to-dateness or timeliness.

Users may use QoC information to rate to quality of recommendations based on contexts with a certain QoC. Also they can express privacy policies using QoC. For example they can allow access to their location with reduced accuracy/precision and less up-to-date.

Contrary to the notion of Buchholz et al., that there may be many more QoC parameters depending on the context source, Zimmer defines only four parameters that should apply to all contexts in general [46]:

- Spatial Origin – That is the location of the sensors source.
- Age – Like up-to-dateness this parameter denotes how old a context information is and additionally how old the oldest context that was used to derive that information is.
- Reliability – That is the same as probability of correctness. Zimmer states that it may be hard to calculate the reliability and proposes to use fuzzy-rules defined by experts.
- Degree of Relationship – This parameter expresses how

a context information is related to other contexts, e.g. because the other contexts are fused to derive this context or the other contexts partly use the same sensor sources.

Sheikh et al. rename up-to-dateness to freshness and also add another two parameters to the list of QoC parameters [47]:

- Spatial resolution – That is the precision of the spatial origin.
- Temporal resolution – That is the precision of the time stamp attached to the context information. The definition of timeliness by Gray and Salber covers this definition but extends it to the delivery delay, which causes ambiguity in Gray and Salber's definition [44].

Sheikh et al. argue three reasons for the use of QoC. First, they consider QoC information an additional cause for application adaptation. Second, they find QoC useful to enable higher middleware efficiency, e.g. if a sensor with lower computational cost provides lower, but still sufficient QoC a costly sensor may be replaced. Last, they see a use for QoC parameters in the expression of privacy [47].

Manzoor et al. give a more precise definition to actually calculate some of the QoC parameters defined by the other authors [48]. They define up-to-dateness with respect to a maximum lifetime of a context information and decrease the up-to-dateness value when the actual lifetime reaches near the maximum. Also they define trust-worthiness with respect to the space between sensor and measured real world entity.

Kim and Lee propose to express accuracy as whether a value is in a confidence interval that is statistically estimated [49]. However, this approach seems more suited to calculate the probability of correctness than to express the accuracy/-precision.

In [50] Krause and Hochstatter observe that there is still a challenge to model the QoC parameters, especially because there is 'an endless range of possibilities' to express the QoC parameters. They also note that QoC parameters do not only apply to context information derived from a single sensor source but also to aggregated contexts. Furthermore, they argue that QoC and the worth of a context information for a particular application may be two things and therefore they extend the definition of QoC to:

> *Quality of Context (QoC) is any inherent information that describes context information and can be used to determine the worth of the information for a specific application. This includes information about the provisioning process the information has undergone (history, age), but not estimations about future provisioning steps it might run through.* [50]

2.4 Human activity recognition

Human activity recognition is used as a sample context aware application in this thesis in order to evaluate the influence of time synchronization on reasoning accuracy. Therefore, here we will outline the state of the art in human activity recognition.

More than ten years ago research on activity recognition began using audio and video data [51, 52]. Later, the use of body worn sensors became popular [53, 54, 55, 56, 57].

Depending on the number of used sensors and used sensor devices as well as on the heterogeneity of the devices, the problem of time synchronization becomes more relevant:

- Single sensor – If only one sensor is used to recognize an activity, the relative ordering provided by the timestamps from the sensor device is sufficient and no time synchronization is necessary. An example for this class of activity recognition approaches is using the acceleration sensor in a single smart phone [58, 59, 52, 57].

If training data is labelled manually using a second device the synchronization of the sensor device and the labeling device is important to mark the transition point between two activities.

- Single device – Most activity recognition approaches are using a single device equipped with multiple sensors, e.g. [60, 55, 61, 51, 62]. The relative ordering provided by the device time is also sufficient in that case. If a second labeling device is used to establish ground truth the time synchronization may be an issue.

  The different sensors attached to the device may however have different sampling frequencies and for some sensors it may take longer to detect a physical fact than for other sensors (timeliness). In that case sampling time may need coordination.

- Homogeneous devices – Approaches using multiple sensor devices of the same type, attached to the human body at different positions, in a wireless sensor network (WSN) have to secure time synchronization between the sensor devices. This will be discussed in detail in Chapter 3.

  Even if the accuracy of time synchronization available for wireless sensor networks (see chapter 2.1.2) is sufficient, to the best of our knowledge there is no clue how time synchronization is achieved in the published research work in this field, e.g. [56] or [54].

- Heterogeneous devices – The vision of ubiquitous computing encompasses the use of a multitude of sensors in the user's proximity and personal devices of a user, like the smart phone, together. These devices may use different communication technologies to connect to each other. It cannot be assumed that all involved devices offer sufficient ways to synchronize their clocks or have clocks of sufficient accuracy at all. In Chapter 3 the issues caused by unsynchronized clocks will be discussed.

Today most modern smart phones are equipped with accelerometers and several groups published results showing good reasoning accuracy using these accelerometers [58, 59, 63].

Throughout this thesis acceleration sensors like the ones in smart phones and their fusion with other sensor data will be used in experimental settings to show various time synchronization related issues.

So far, none of the published result considers the influence of time synchronization and the clock accuracy of smart phones.

Chapter 3

Reasons for timing issues

In Chapter 2.1 the principle of electronic clocks is described along with time synchronization protocols. In the following reasons why clocks involved in the acquisition of context may not be synchronized or why time stamps may not be accurate are discussed. Some of these reasons can be addressed using the known time synchronization protocols, hitherto the other reasons are not addressed.

To analyze the issues that may be caused by unsynchronized clocks we make the following definitions:

Definition 4. *The time t_x is the time of the event e_x recorded by the device sensing event e_x.*

Definition 5. *Given Definition 4, the clock c_x is the clock from which t_x is taken. The offset of the clocks c_x from clock c_y is $t_{\text{offset}}^{c_x,c_y}$.*

Definition 6. *Given Definition 5, when there is no offset two events e_1 and e_2 occur at the same time if $\Delta t \geq |t_1 - t_2|$. Δt depends on the applications need for accuracy.*

If there is an offset, an event e_1 occurs before e_2 if $t_1 + \Delta t + t_{\text{offset}}^{c_1,c_2} < t_2$.

Definition 6 defines a relative temporal ordering of events. It defines if an event happens before or after another event and if two events happen at the same time. The order is relative and not absolute because it is defined relative to the involved clocks, whether these clocks are synchronized to any external source providing an absolute time or not.

3.1 Clock drift and possibility of clock adjustment

As already explained in Chapter 2.1, electronic clocks are based on the properties of quartz (silicon dioxide). Quartz, formed into a small tuning fork oscillates at 32.768 Hz or 2^{15} cycles per second. Counting these cycles allows to generate a time of day.

Oscillators are sensitive to temperature, pressure and other factors, e.g the precision of the form of the tuning fork. Thus oscillators cannot maintain a precise frequency with infinite accuracy, the offset between two clocks will always increase over time, which is called *clock drift*.

Clocks have a certain *stability*, which is 'how well it can maintain a constant frequency', and clocks have a certain accuracy, which is 'how well its time compares with national standards. ...The time offset of clock i relative to clock j is the time difference between them $T_{ij}(t) \equiv T_i(t) - T_j(t)$, while the frequency offset of clock i relative to clock j is the frequency difference between them $R_{ij}(t) \equiv R_i(t) - R_j(t)$' [27]. In other words, because of insufficient clock stability, the clock will drift and thus the clock's accuracy will decrease.

Wireless sensor networks (WSNs) are often associated with low cost sensor devices that may cause device quality related and other problems, e.g. unstable oscillators, limited energy and limited communication bandwidth [16]. While there are several clock synchronization approaches prosed that can archive sufficient accuracy [16, 17, 18, 19, 21, 20], context aware architectures often combine data from several sensors, that are not part of the same WSN, and thus are likely to be not synchronized at all.

From this follows that it cannot be assumed that all clocks involved have sufficient accuracy and thus it cannot be assumed that time stamps are accurate. In fact time stamps must be regarded as potentially wrong, as long as it is not known what provisions are made to ensure correct time stamps.

Assuming the involved sensor devices provide some means to adjust the clock, the algorithms described in Chapter 2.1 can synchronize the device clocks. However, many applications use sensors integrated in modern smart phones, e.g. human movement detections applications make use of acceleration sensors in smart phones. Unfortunately, up to now, it is not possible to use the time synchronization algorithms on smart phones for the following reasons:

- It is not possible to set the time programmatically neither using JavaME nor on Android and iOS.
- Connecting to Internet time servers requires the user to allow the use of a connection, normally opening a user dialog on the phone.
- The user simply may not want his phone's clock time to be set.

If smart phone clocks cannot be synchronized, is it save to use time stamps from smart phones? First of all, in CDMA networks time is distributed to the phones, in GSM that is not the case. While the Network Identity and Time Zone (NITZ) specification is part of the GSM specification [64], for example in Germany only one of the four major network providers supports this feature (tested on Nokia E71 phone).

```
time.hrz.uni-kassel.de:  stratum 1,
offset 0.000671, synch distance 0.002717,
refid 'DCFp'
```

Figure 3.1: Output of ntptrace of the time server.

Also, at least the smart phones we have tested (see Table 3.1) require the user to activate the setting on the phone or allow the user to disable it. Setting the clock by hand will not be more accurate than to the nearest second. To verify that smart phone clocks have a significant offset and are drifting we carried out the following experiment.

3.1.1 Experimental setup

To test the offset and drift of smart phone clocks, the clocks are compared to the clock of a desktop computer. Therefore, the desktop computer's clock is synchronized using NTP to `time.hrz.uni-kassel.de`. This time server is a stratum 1 NTP time server located in the same university network than the desktop computer (see Figure 3.1). The server uses the DCF77 radio signal as reference source that receives the time from the 'Physikalisch-Technische Bundesanstalt' in Braunschweig, Germany [65]. This national institute uses cesium clocks to provide the legal time in Germany.

The clock of the desktop computer has, according to the `ntptime` command, an offset of -2.611 μs to the time server and the estimated error is 78 μs (maximum error 133961 μs).

On the smart phones a program implemented in Java (Java Micro Edition) is used. This program sends a time stamp every 0.31 seconds ($\approx$ 32,25 Hz) to a server program running on the desktop computer (retrieved on the smart phone using Java's `System.currentTimeMillis()` function). When the desktop computer receives the time stamp the offset is calculated, thus the network sending time is not estimated and subtracted, but can safely be expected to be in the order of milliseconds not seconds. The average ping time of 50 pings to the desktop computer utilizing the slowest connection used in the experiments is 57.451 ms. Two different connections are used in the experiments, WLAN with the access point connected over Gigabit-Ethernet to the desktop computer, and UMTS. The UMTS/GSM network provider used has the 'Network Identity and Time Zone' (NITZ) extension enabled in his network.

A total of 17 experiments with 9 different smart phone models are carried out. In every experiment 60,000 samples are taken. The smart phones are taken from their users without adjusting the time before the experiment.

3.1.2 Results

Table 3.1 summarizes the results. It can be seen that in the best case the mean offset is 2.01 seconds, in the worst case for two phones it is roughly two minutes (if the two phones that are one hour off and the one that is one year off are neglected, because it can be assumed the users of these phones never tried to set the time right). The Nokia and Samsung models have a variance of more than 19 seconds. The Sony Ericsson model and the one from LG have a variance below 5 seconds. That difference can be seen when the figures Figure 3.6 and Figure 3.8 or Figure 3.2 and Figure 3.4 are compared; the models that showed greater variance can be identified by the greater spikes in the graphs. The figures Figure 3.7, Figure 3.9, Figure 3.3, and Figure 3.5 show histograms of the time offset for four chosen phone models.

All smart phone model's clocks that are tested drift. The figures Figure 3.6, Figure 3.8, Figure 3.2, and Figure 3.4 show a red line calculated with a least squares polynomial fit algorithm. The gradient of this red line estimates the clock

model	connection	variance (sec)	mean (sec)	gradient (sec/hour)
LG-KM900	WLAN	4.61	3735.36	0.48
Nokia5800d-1	WLAN	1943.10	-6.04	-13.77
NokiaE71-1	UMTS	11530.60	-6.12	-1.05
NokiaE71-1	UMTS	95.82	-4.39	-1.39
NokiaE71-1	UMTS	37944.95	-7.77	-3.96
NokiaE71-1	WLAN	123.87	-2.01	-0.81
NokiaE71-1	UMTS	30.17	78.55	-1.45
NokiaE71-1	UMTS	44.02	2.81	-1.30
NokiaE71-1	UMTS	686.19	8.41	-1.52
NokiaE72-1	WLAN	520.02	-23.90	-2.47
NokiaE72-1	WLAN	456.29	-24.57	-3.81
NokiaN95	WLAN	600.75	44.56	-9.96
NokiaN97-1	WLAN	200.19	118.73	-0.11
NokiaN97-4	WLAN	19.60	-31536037.36	-1.01
S8000	WLAN	4578.00	3674.08	0.43
SonyEricsson-U1i	WLAN	1.45	4.99	0.27
SonyEricsson-U1i	WLAN	0.36	22.01	0.02

Table 3.1: Overview of time differences and clock drift on the tested smart phones.

drift. One phone model drifts 13.77 seconds/hour (worst case). Two phone models show a drift below 1 second/hour. The rest of the tested phones drifted between 1 second/hour and 10 second/hour.

Two times the Nokia E71 shows a significant higher variance than in the other experiments. These times the phone used a UMTS connection and was carried around during the experiments; during all other experiments the phone was left unmoved on a desk. Thus the greater variance can be attributed to network delays. The drift observed in the two experiments with moving phone still fits in with the other experiments.

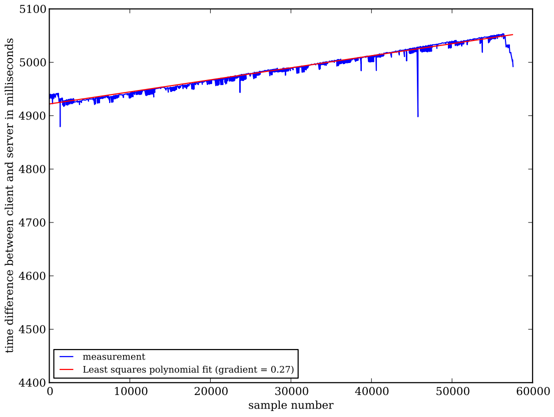


Figure 3.2: Time offset desktop computer synchronized with NTP to Sony Ericsson U1i (Satio).

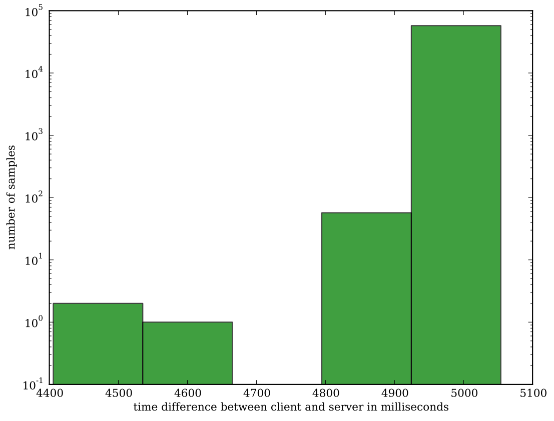


Figure 3.3: Histogram of time offset desktop computer synchronized with NTP to Sony Ericsson U1i (Satio).

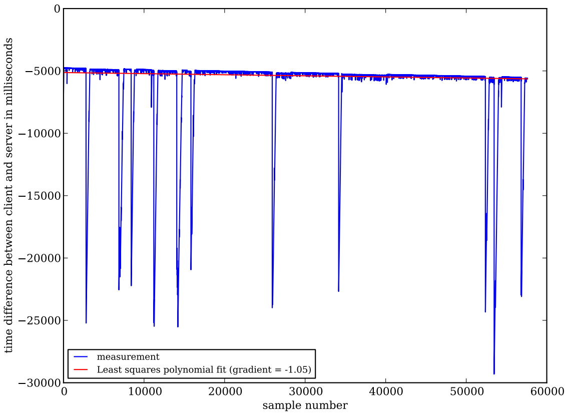


Figure 3.4: Time offset desktop computer synchronized with NTP to Nokia E71-1.

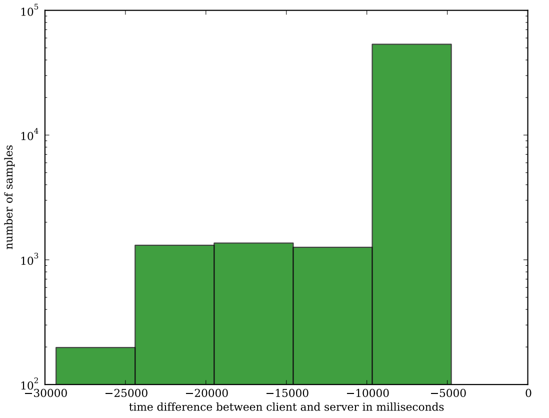


Figure 3.5: Histogram of time offset desktop computer synchronized with NTP to Nokia E71-1.

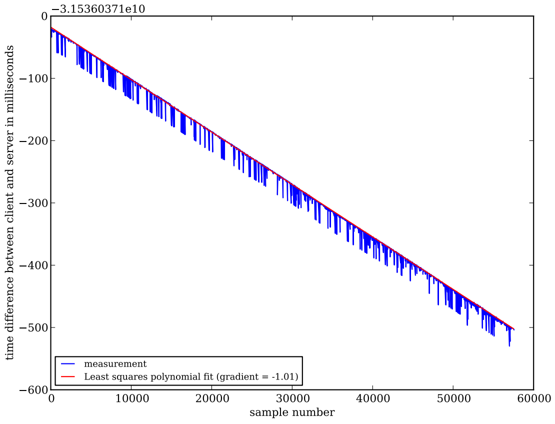


Figure 3.6: Time offset desktop computer synchronized with NTP to Nokia N97-4.

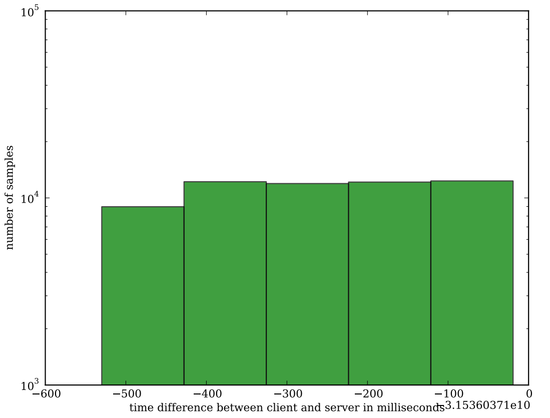


Figure 3.7: Histogram of time offset desktop computer synchronized with NTP to Nokia N97-4.

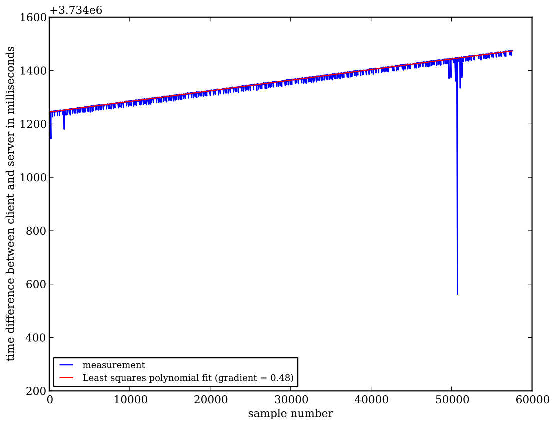


Figure 3.8: Time offset desktop computer synchronized with NTP to LG-KM900 (Arena).

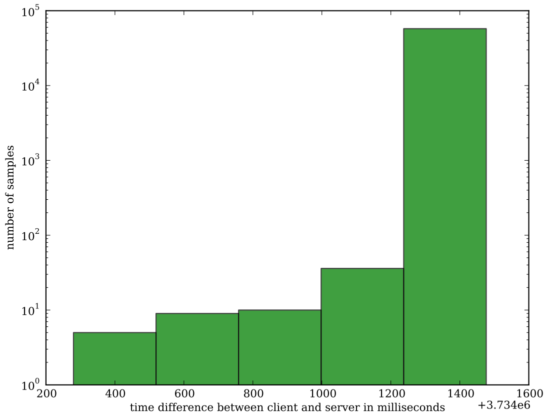


Figure 3.9: Histogram of time offset desktop computer synchronized with NTP to LG-KM900 (Arena).

3.1.3 Discussion of results

The experimental results show that due to the reasons discussed in this chapter, it cannot be safely assumed that clocks of smart phones are synchronized to the nearest second. The users of the phone may not set the clock to correct time at all and if they try to, it will be impossible to set it to the exact millisecond. Even the phones tested that use the UMTS connection where the network provider disseminated the time through the network are not significantly more accurate (best case 2 seconds off). The observed clock drift in the order of seconds per hour makes it worse, even if the clock is correct to the millisecond at one point in time, it takes only hours to drift off. In Chapter 4 the effect of an offset in that order will be discussed in detail. The offset generally is an important information without which the relative temporal ordering of events is not known (see Definition 6).

3.2 Construction related differences

Even if we assume that the clocks of the used sensor devices are synchronized, there are reasons, why the devices will not deliver sensor data all at the same frequency. Furthermore sample data that is to be aggregated, will not be sampled at the same time by all devices. The 'same time' here is used to refer to a defined period of time wherein all time stamps are regarded as if they are the same, depending on the specific application's need for accuracy (see Definition 6).

Some sensors may take more time than others to sense a change in the physical environment. To this phenomenon from now on is referred to as sensor delay.

Definition 7. *The time t_{rx} is the time when event e_x really occurred; thus $t_{rx} \leq t_x$. The sensor delay is defined as $t^x_{\text{delay}} = t_x - t_{rx}$.*

Definition 8. *Given Definition 7, when there is no offset two events e_1 and e_2 occur at the same time if $\Delta t \geq |t_{r1} - t_{r2}|$. Δt depends on the applications need for accuracy.*

If there is an offset an event e_1 occurs before e_2 if $t_{r1} + \Delta t + t^{c_1,c_2}_{\text{offset}} < t_{r2}$.

The time an event really occurs is $t_{rx} = t_x - t^x_{\text{delay}}$.

The sensor delay is caused by the fact that sensors have different construction-related physical capabilities. In dynamic environments where several sources of sensor data may be dynamically discovered and aggregated, it is not safe to assume that all sensors have the same physical capabilities even if they are of the same general type. Furthermore, sensors of different types clearly have different physical capabilities. For example temperature sensors are known to suffer from thermal inertia and therefore the sensor delay is relatively big. However, these construction related delays are deterministic and thus the sensor delay can be taken into account when calculating the relative temporal order of events.

3.3 Processing load of sensor devices and network sending time

The sensor nodes have additional tasks to fulfill. These additional tasks also consume time that cannot be used to retrieve sensor values. For example smart phones are often equipped with accelerometer sensors and other sensors, however they clearly have additional processor load. Normally, there is no way to influence the scheduling algorithms on the device in order to ensure measurements at a fixed frequency.

Additionally, it depends on the operating system how fast a sensor can be accessed, e.g. on the current implementation of the Android OS it is not possible to specify an exact sampling frequency, instead one can choose between four constants for the sensor delay (fastest, game, normal, and ui) [1]. It is not specified to which sampling frequency a constant refers to and it can vary on different devices and software versions.

The additional processor load and the operating system constraints thus add another time delta to the overall time from the real occurrence of an event until it is passed to the application. The time delay caused by processing load and operating system specific constraints is nondeterministic and hence it cannot simply be used in the calculation of the relative temporal ordering. Also, the maximal possible sampling frequency is constrained by the processing load, caused by the sampling itself and the additional tasks, in relation to the processing power. For example, the SunSpot sensor node used in the experiments in Chapter 4 is not able to sample faster than 320 Hz when there is now other processing load, but if every single samples is to be send separately to a second device that additional processing load reduces the maximal sampling frequency to 250 Hz.

If the application that should receive the event is not running on the device that has sensed the event, on top of sensor delay and the time it takes for the operating system to access the sensor, the time it takes to send the event notification to the application's device is to be taken into account. This time is also nondeterministic.

Definition 9. *The time it takes from the real occurrence of an event until it reaches the application* Δt_{notify} *is composed of the sensor delay* t^x_{delay}*, the processing delay* $t^x_{\text{processing}}$ *and the network delay* t^x_{network}*.* $\Delta t_{\text{notify}} = t^x_{\text{delay}} + t^x_{\text{processing}} + t^x_{\text{network}}$

[1] `http://developer.android.com/reference/android/hardware/SensorManager.html`

3.4 Context processing delays

The common understanding of context reasoning is that sensor data is to be processed in several steps and that it is necessary to infer high level contexts out of the raw data [15, 22, 23, 24, 25]. Figure 3.10 shows the typical data flow of context data from sensor to high-level context. These pre-processing and abstraction steps all consume processor capacity and in distributed scenarios are foreseen to be carried out on different devices with respect to the device capabilities, which consumes time while sending data over the network. Therefore, the time it takes before sensor data reaches the application is not only determined by the sensor itself, but also by the time it takes to process the data. This time is the sum of the time consumed by the pre-processing and abstraction steps and the network delays and can vary from value to value and sensor to sensor.

The time every pre-processing and abstraction step takes is influenced by the CPU load and the scheduler and thus is non-deterministic. The network delays are influenced by several factors, such as the bandwidth and thus also are non-deterministic. The sensor's sampling frequency may be faster than the slowest pre-processing or abstraction step, consequently the rate at which the samples reaches the application may be slower than the sensor's maximal sampling frequency.

In Chapter 4 the issues caused by the nondeterminism of the processing load of sensor devices and the context processing delays will be analyzed and their effect on context reasoning accuracy will be tested.

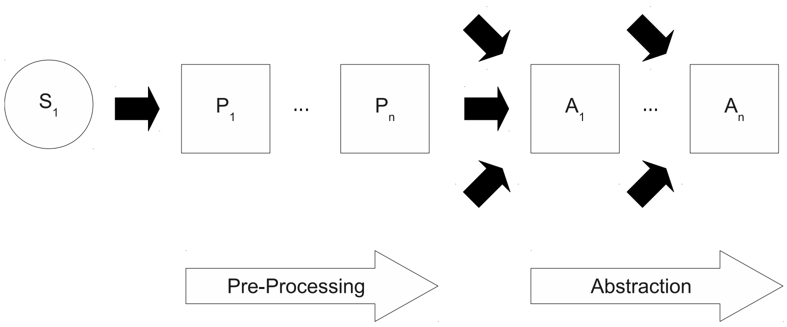


Figure 3.10: Typical data flow of sensor data. Retrieved from sensor S_1 the date is pre-processed in several steps $(P_1 \ldots P_n)$ and afterwards to gain high-level context it is processed in order to abstract it $(A_1 \ldots A_n)$. The abstraction steps may fuse data from several sensors or other abstraction steps.

Chapter 4

Analysis of the influence of time synchronization on reasoning accuracy

Understanding a user's situation is an essential feature of context aware applications, which are an important part of the vision of ubiquitous computing [1, 6]. In many such applications data of several sensors is aggregated to gain an understanding of the user's situation, often called context reasoning, which is commonly done based on rules (e.g. 'if $temperatureSensor > 20°$ C and $lightSensor > 10.000$ lx than summer day'). We differentiate between user defined rules that are simple rules defined by humans, and classification learning, which uses labelled training data for machine learning of classifiers and thus is able to do more complex reasoning.

In the following chapter the influence of time synchronization on reasoning accuracy for reasoning based on user defined rule will be investigated. Afterwards, in Chapter 4.2 the influence of time synchronization on context reasoning based on classification learning algorithms will be analyzed and evaluated for six commonly used algorithms.

4.1 Influence on accuracy for user defined rules

User defined rules are rules that describe the reasoning logic in context reasoning processes and that are manually defined by a user or domain expert. These rules are of an 'if .. then ..' form where the 'if' part can contain the operators '$<$', '$>$', '$=$', '$\geq$', '$\leq$', 'and', and 'or'. These operators are used on the sensor values (e.g. 'if $temperatureSensor > 20°$ C and $lightSensor >$ 10.000 lx than summer day').

For user defined rules the synchronization of sensor clocks is important to infer a relative ordering of sensor readings. Figure 4.1 shows a schematic sketch of a moving person. In the top sketch the sensors attached to the thighs have synchronized clocks, in the lower sketch the clocks are out of sync. It is possible to infer the movement of the person from the sensor orientation, if the clocks are in sync, if the clocks are not in sync it is no longer possible to infer the movement. To infer the movement it has to be clear if two sensor readings happen at the same time or one after another (see Definition 6). In the sketch the time frame Δt, in which two time stamps have to be in to be regarded as happening at the same time, is marked with dotted blue lines. The size of Δt is application specific and for human movement detection is accepted to be determined by a maximum frequency of 10 Hz [61].

In a scenario of a smart meeting room it is beneficial to know whether a person is standing, walking around or sitting. This would, for example, enable to infer (along with other parameters) if a meeting has started and which person currently is doing a presentation at the white board.

To detect whether a person is standing, walking around or sitting, we use accelerometers placed in a person's pockets.

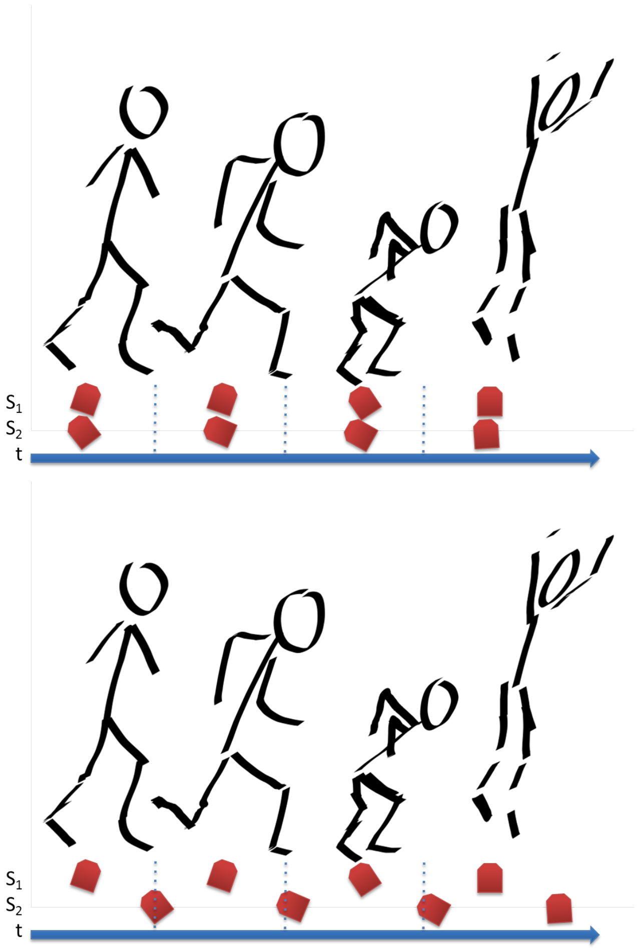


Figure 4.1: Schematic sketch of a moving person. The red boxes show the sensor's orientations when the sensors are attached at the person's thighs. The dotted blue lines mark the begin of a new time frame. In the top sketch the sensor's clocks are in sync and in the bottom sketch not.

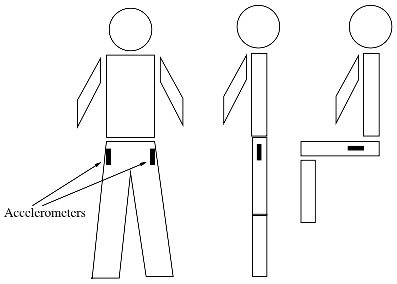


Figure 4.2: Positioning of accelerometers in the pockets (left). Standing person with upright accelerometers (middle). Sitting person with accelerometers rotated 90 ° (right).

Many smart devices today have integrated accelerometers. One accelerometer in a pocket on the side of each leg is sufficient (Fig. 4.2), as further shown in this chapter. The three-axis accelerometer allows to calculate the angle of the upper leg relative to the earth gravity. Thus, it can be distinguished between three general situations; if both legs are pointing in the direction of the earth gravitation, a person is probably standing, if both legs are angled a person is probably sitting and if both legs' angles are alternating, a person is walking.

To exactly distinguish the three situations, the time synchronization is important. To detect alternating or simultaneous movements the temporal order of measurements is clearly important. Human movement happens in a frequency range from 0–10 Hz [61]. This means to classify values as happening simultaneously, their time stamps have to differ less than 0.1 sec. There are several possible reasons why sensors could not be synchronized to that extent.

One reason could be that the two used devices (e.g. one smart phone and one special sensor device) have different physical characteristics. If both devices are theoretically able to deliver measurements with the same frequency, it is still possible that the devices are busy with other tasks. Another common reason for delayed measurement delivery is the need to further process the data and send it over several network hops. In that case, processor load and network delays lead to unpredictable delays. For a complete discussion of the reasons see Chapter 3

Generally, aggregated sensor data should always be captured around the same time (constrained by the sensor's characteristics and the application's needs), otherwise the data may be inconsistent. None the less, existing approaches for context aware systems typically simply assume that the time when the different sensor data is captured does not matter for the calculations. A common approach is to use publish/subscribe communication (e.g. [14]) to inform interested parts of a context aware application whenever a value changes (see Chapter 2.2). For slowly changing values this may not be an issue, but for fast changing sensor data it is critical to maintain correct temporal relations.

There are several sensor types whose values change much more frequently and for which a single sensor value is of no meaning, but the change over time is of importance (e.g. video, audio, vibration and acceleration [62]). For these kind of sensors a publish/subscribe based architecture may not be the best solution, because the frequent change will result in constant streams of change events. Particularly, the time constraints for these kinds of sensors are much more tight; if a sensor's values change every second, tolerating a time difference of two seconds is not feasible. If these time constraints are not fulfilled, the data may be useless to some applications or otherwise less valuable.

In the following the timing issues raised by these differences are analyzed. When it is sensed whether a person is standing or sitting, to be certain of the position of both thighs, it is

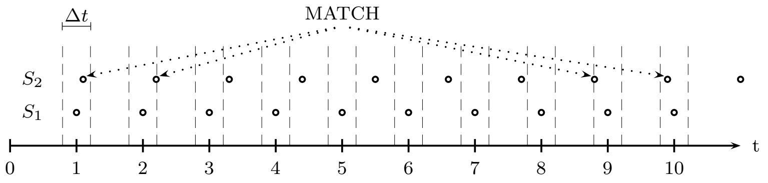


Figure 4.3: The first 10 samples of S_1 (bottom) and S_2 (top).

necessary to get the information from both accelerometers at the same time and to allow only a small difference Δt. Otherwise it will be unclear if a differing angle is caused by the time delay of one sensor, or by really different angles, like if a person only lifts one leg. To ensure this is relatively simple when both sensors use identical sampling rates and are synchronized. If the two sensors however are not able to sample at identical rates, maybe because they use different technologies, the problem becomes harder.

Without loss of generality, we assume the first sensor S_1 is the faster one and samples at a rate R_1 of one sample per second and S_2 is slower by factor x and thus has sampling rate $R_2 = xR_1$. Two samples s_1 from S_1 and s_2 from S_2 have fitting timestamps if the time t_{s1}, that is the time when s_1 is measured, differs only Δt from t_{s2}. For example if x is 1.1 and S_1 samples once per second then S_2 samples every 1.1 second. This results in two time series of samples: $T_{s1} = (1, 2, 3, 4, ...)$ and $T_{s2} = (1.1, 2.2, 3.3, 4.4, ...)$. If $\Delta t = 0.2$ then only some $t_{s1} \in T_{s1}$ are fitting. Figure 4.3 shows the first 10 samples of both sensors and marks the fitting timestamps.

To calculate to percentage p_{S1} of fitting timestamps in T_{s1}, first the percentage p_{S2} of samples in T_{s2} that have fitting timestamps has to be calculated. In T_{s2} for a $\Delta t = 0.2$ only sample times where the first decimal place is $0, 1, 2, 8$ or 9 are fitting because in T_{s1} there are only $0's$ as decimal places. This, out of the first 100 samples, is exactly the case if $t_{s1} \in (1, 2, 8, 9, 10, 11, 12, 18, 19, 20, ..., 90, 91, 92, 98, 99, 100)$. From this follows that only 40 of 100 timestamps from S_2 are fitting with a timestamp from S_1. When 100 samples from S_1 are taken, there are only $100/x$ samples taken from S_2, out of which 0.4 have fitting timestamps. Thus p_{S1} is $\frac{100}{1.1}0.4 \approx 36\%$

The percentage of samples of the slower sensor which have fitting timestamps with a sample from the other sensor p_{S2} depends on x, with $x = \frac{R_2}{R_1}$, and the allowed time difference Δt. The first decimal place of a sample s_2 with sample time t_{s2} has to be equal or less than the allowed difference Δt or equal or greater $1 - \Delta t$. If T_{s2}^{fit} is the set of fitting sample times in T_{s2}, $T_{s2}^{fit} \subseteq T_{s2}$, than $\forall t_{s2}(\exists t_{s1} \in T_{s1} \wedge |t_{s2} - t_{s1}| \leq \Delta t \Rightarrow t_{s2} \in T_{s2}^{fit})$. Furthermore there is a $t_{s1} \in T_{s1}$ with $|t_{s2} - t_{s1}| \leq \Delta t$ when $(t_s2 \mod x) \leq \Delta t \vee (1 - (t_s2 \mod x)) \leq \Delta t$.

If y is the first decimal place of x ($x = z + \frac{y}{10}, z \in \mathbb{N}, y \in N, N = (1, ..., 9)$) than out of 100 samples there are $p_{s2} = 10|M|$ ($M \subseteq N, \frac{n}{10} \leq \Delta t \Rightarrow n \in M$) fitting timestamps. To calculate $|M| = f(\Delta t, y)$ use the following equation (1):

$$f(\Delta t, y) = \begin{cases} 2(10\Delta t) + 1 & y \in (1, 3, 7, 9) \\ 2(10\Delta t - (10\Delta t \bmod 2)) + 2 & y \in (2, 4, 6, 8) \\ 5 & y = 5 \end{cases} \tag{4.1}$$

Table 4.1 shows the result of $f(M, y)$ for $y \in (1, 2, ..., 9)$.

For uneven y in the progression ($1y \bmod 10, ..., 10y \bmod 10$) every number $\in (1, 2, ..., 10)$ appears exactly once. For even y only even numbers appear in the progression and therefore only even increases of Δt can add more fitting timestamps. The multiplier 2 in Equation 4.1 is because adding Δt can lead to fitting timestamps as well as subtracting Δt. Fig. 4.5 shows the percentage of fitting timestamps in T_{s2} and T_{s1}. The percentage of fitting timestamps in T_{s1} depends on x, because during the time 100 samples for T_{s1}

Δt	$y =$ even	$y =$ uneven ($x \neq 0.5$)	$x = 0.5$
0.1	20	30	50
0.2	60	50	50
0.3	60	70	50
0.4	100	90	50
0.5	100	100	100

Table 4.1: Percentage of fitting timestamps from the slower sensor.

are sampled, there are only $\frac{100}{x}$ samples in T_{s2} of which only p_{s2} have fitting timestamps with a sample from T_{s1} (4.2).

$$p_{S1} = \frac{100}{x} p_{S2} \tag{4.2}$$

As the figures Figure 4.4, Figure 4.5, Figure 4.6, and Figure 4.7 show, even if the first sensor is not twice as fast as the second sensor ($1 < x < 2$), in the worst case ($\Delta t = 0.1$) only $\approx$ 18.2% of samples from the first sensor have fitting timestamps from the second sensors. With $x \geq 2$ it gets even worse (Fig. 4.7). The other $\approx$ 82.8% of the samples from sensor one are a potential waste of energy and network bandwidth. If these samples are used by a reasoning algorithm anyway, the accuracy may decrease! For applications that require the use of two sensors and define a maximum time difference between two samples, one from each sensor, a way to avoid the unnecessary sampling would save energy and network bandwidth.

In the example of sensing the angles of both thighs of a person, when one sensor indicates that a leg is angled and the other sensor shows the opposite, the time difference is important to decide if the differing measurements are caused by differing sampling rates or by the person really lifting only

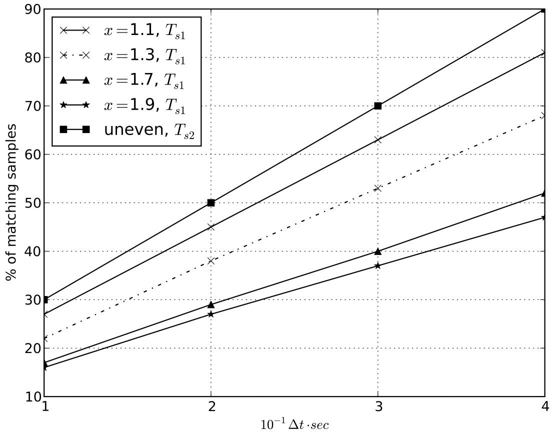


Figure 4.4: Shows % of fitting timestamps for uneven first decimal place of x in relation to Δt ($1 < x < 2$).

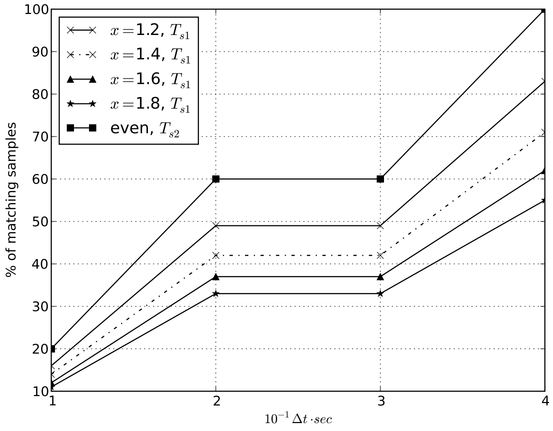


Figure 4.5: Shows % of fitting timestamps for even first decimal place of x (except $x = 1.5$) in relation to Δt ($1 < x < 2$).

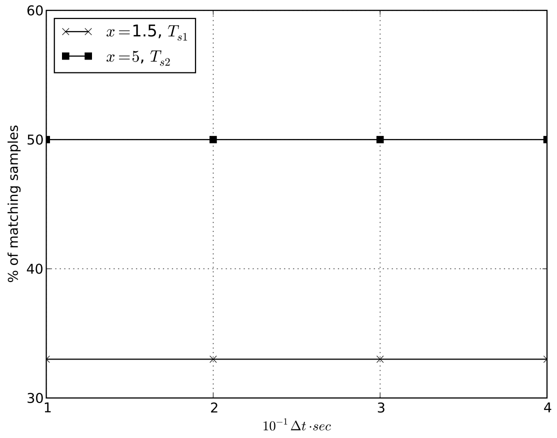


Figure 4.6: Shows % of fitting timestamps for factor $x = 1.5$ in relation to Δt.

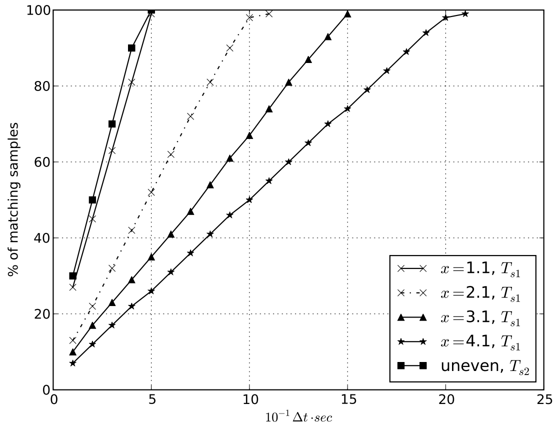


Figure 4.7: Shows % of fitting timestamps for uneven first decimal place of x in relation to Δt.

one leg instead of sitting down.

4.2 Influence on accuracy for classification learning

Context reasoning based on classification learning contrary to reasoning based on human user defined rules, offers the benefit of saving the work of rule definition and can reveal relationship between context and sensor values that may not be evident to the human user. Therefore, it is an often used approach in context reasoning. However, learned classifiers are dependent on proper training data and are prone to over fitting [66]. In the following the effect of time synchronization of the sensors producing the training data and the test data will be evaluated.

To evaluate the influence of time synchronization on reasoning accuracy of classification learning algorithms the following scenario, in which user activity is classified based on acceleration data, is used.

In a scenario where context aware applications are to support office workers, it is beneficial to know the current activity of a user, e.g. if it can be detected if a user is currently using his desktop computer, the power saving modes of this computer can be adjusted accordingly. To access the situation of a user sitting in front of and working with his computer, we attached one SunSPOT sensor node per arm, equipped with a three-axis accelerometer, to the user's wrists (see Figure. 4.8). The accelerometer data is used to distinguish five different activities: typing on the keyboard, hand writing, moving the mouse, using the phone, and drinking.

The experiments show that it is possible to distinguish the five activities with reasonably good accuracy (82.09 % in the worst case) when the sensor data timestamps are synchronized (Figure 4.9, synchronization is achieved by manually aligning

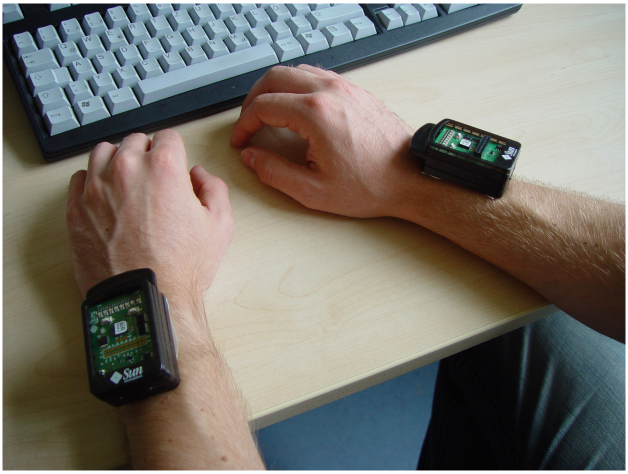


Figure 4.8: Positioning of sensor nodes with accelerometers, one per wrist.

the data after the recording). However for the several reasons discussed in Chapter 3 it cannot be assumed that the clocks of the two sensors are in sync.

Especially important for base-level classification algorithms is the fact that it is unlikely that the time difference between sensors remains stable over time (e.g. due to clock drift and skew or changing network and CPU load) and therefore classifier performance may change over time. Experiments described in Chapter 4.2.1 show that the accuracy is highly dependent on the time shift between sensors after recording the training data. Chapter 4.2.2 discuses the results of the experimental evaluation and Chapter 4.3 concludes the findings.

4.2.1 Experimental setup

The experiment setup consists of two SunSPOTs devices [15] placed at both wrists of the user (Figure 4.8). The acceleration data is measured at a sampling rate of 32 Hz and the performed activities are annotated using an additional application running on a Nokia Internet Tablet N800. The collected data from all three devices is then combined and synchronized manually to prepare the training data needed for activity recognition. The recognition system uses classification learning algorithms to build the activity models from the training data, based on the approach used in previous investigations on activity recognition using smart phones [59]. The built model is integrated in the recognition system as the designated classifier.

In the training phase, the preparation of the classifiers requires the transformation of raw acceleration data into selected features to be used as attributes. For the recognition needed in the experiment, we have selected three time domain (mean, variance and standard deviation) and two frequency domain (energy and information entropy of the fast Fourier transform) transformations. The test data is also transformed into all five features to be used as input data for the desired recognition using the built classifiers during the recognition phase. The evaluations are made based on the results of the obtained recognitions for both SunSPOT devices. For the purpose of comparison, we have also repeated the evaluations with the sampling rate 16 Hz and different combinations of window lengths as well as overlapping percentages used for the preparation of the features (see Table 4.2).

Based on the obtained training data from both SunSPOTS, classifiers are build using selected base-level classification algorithms. These algorithms are Bayesian network (BN), K-nearest neighbor (IBk), decision tree (J48), rule-based classifier (JRip), naïve Bayes (NB) and sequential mini-

			Number of instances per data set			
Freq.	Window size	Overlap	20100805	20100809	20100811	20100812
16	32	0.50	1192	2186	1504	713
16	32	0.75	2384	4372	3008	1425
16	64	0.50	595	1092	751	356
16	64	0.75	1190	2184	1502	711
32	64	0.50	1189	2179	1502	712
32	64	0.75	2377	4358	3003	1424
32	128	0.50	594	1089	750	355
32	128	0.75	1187	2177	1500	710

Table 4.2: All algorithms were evaluated with eight different parameter combinations for the parameters frequency [Hz], window size [samples] and overlap [samples]. The table shows the resulting number of instances in the training data for every one of the four recorded data sets ('20100805', '20100809', '20100811' and '20100812').

mal optimization (SMO). They are frequently used in various accelerometer-base activity recognition investigations and have shown relatively good recognition accuracies [59, 67, 57].

Unfortunately, there is no way to predict the accuracy of a certain classification learning algorithm used on a certain data set other than actually testing it with the data set using techniques like cross-validation [66]. The choice of a classification learning algorithm therefore can only be done by testing the different algorithms. In previous investigations, the above classification algorithms have shown accuracies higher than 90 %, particularly the IBk, J48 and JRip classifiers [59]. In this investigation, we want to investigate the influence of the time shift on the recognition process. Therefore, the data of the second sensor is shifted respectively 25, 50, 75, 100, 125, 150, 175 and 200 samples to create un-synchronized test data and to test this data with classifiers built from the synchronized training data.

4.2.2 Results

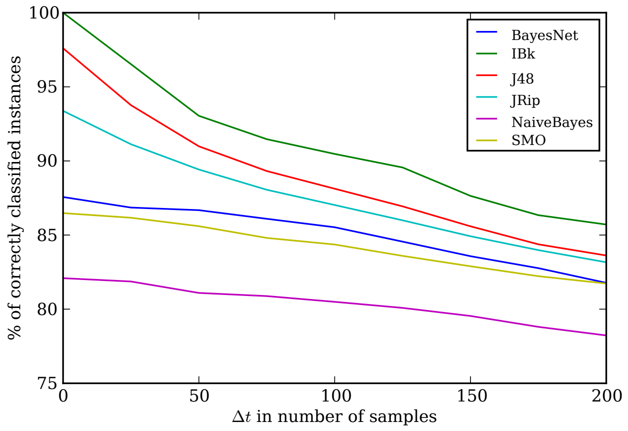


Figure 4.9: Overview of the average performance of all six classifiers

Theoretically, if a classifier is built using training data from both sensor devices without any time shift during the training phase, a delay of sensor data delivery from one of the sensor devices during the recognition phase causes changes in the aggregated data needed for the recognition. If the delay shift is big enough, the built classifiers will fail in producing the expected recognition and accuracy. Experiments show that for all six algorithms used (see Chapter 4.2.1) the accuracy goes down even for a small time shift of 25 samples ($25\,\text{samples} \div 32\,{}^{\text{samples}}/_{\text{sec}} \approx 0.78125\,\text{sec}$). For the maximum tested time shift of 200 samples accuracy goes down by 15 percentage points in the worst case (Table 4.3, see the figures Figure 4.16 – Figure 4.39 which show the results as box plots, where the blue box extends from the lower to upper quartile values of the data, with a red line at the median.). An overview of the average performance of all algorithms is shown in Figure 4.9.

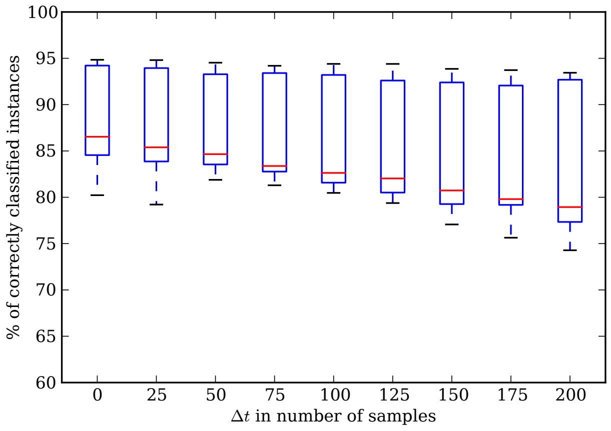


Figure 4.10: Average performance of Bayesian Network classifier (BN)

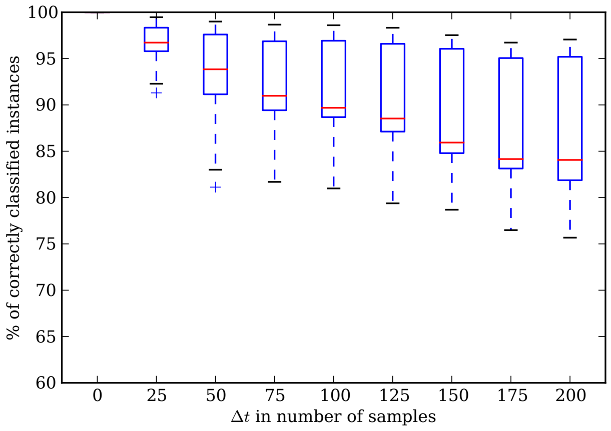


Figure 4.11: Average performance of K-nearest neighbour classifier (IBk)

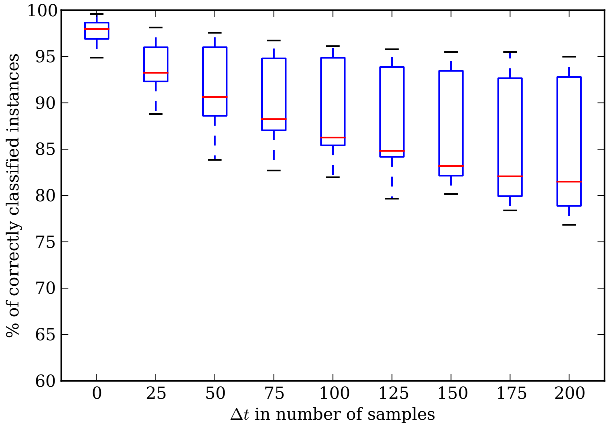


Figure 4.12: Average performance of decision tree classifier (J48)

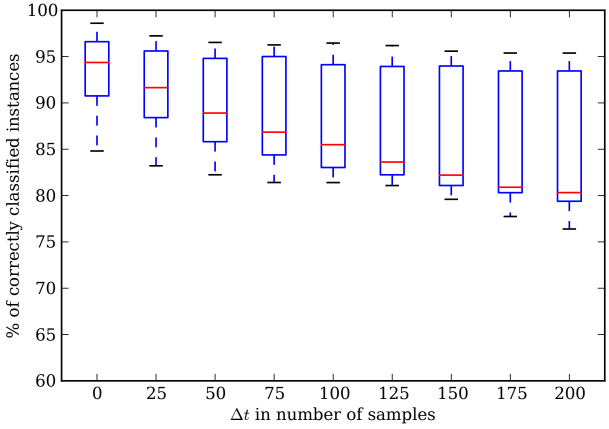


Figure 4.13: Average performance of rule based classifier (JRip)

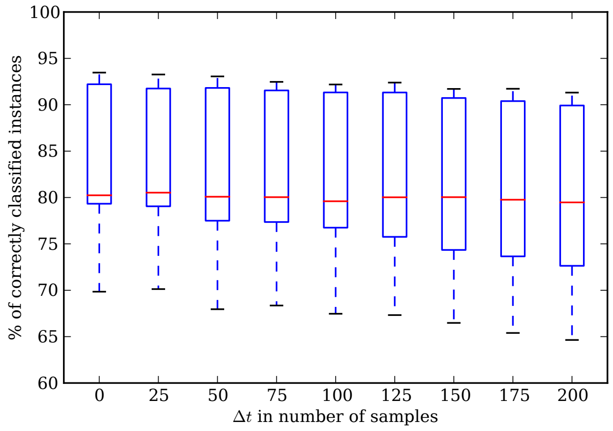


Figure 4.14: Average performance of naïve Bayes classifier (NB)

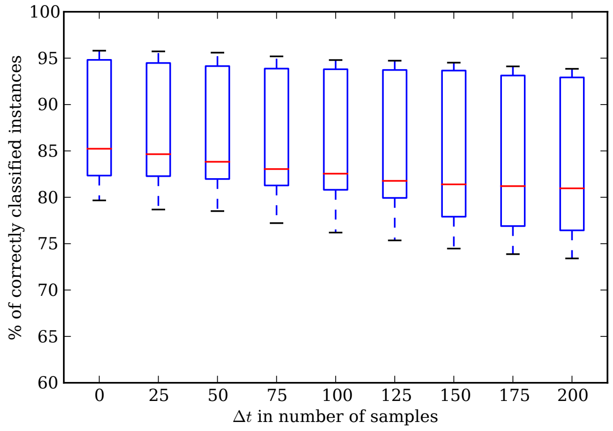


Figure 4.15: Average performance of sequential minimal optimization (SMO)

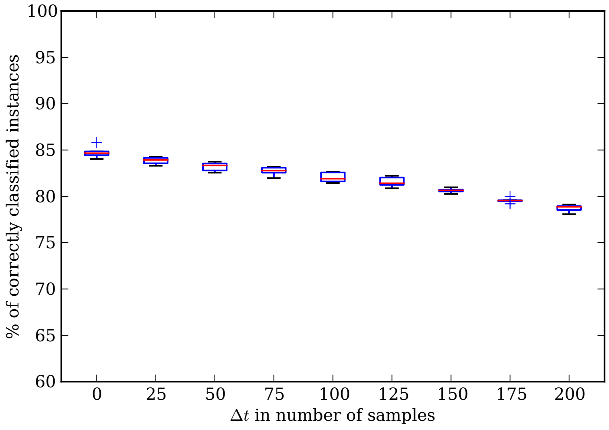


Figure 4.16: Performance of Bayesian Network classifier (BN), data set 20100809

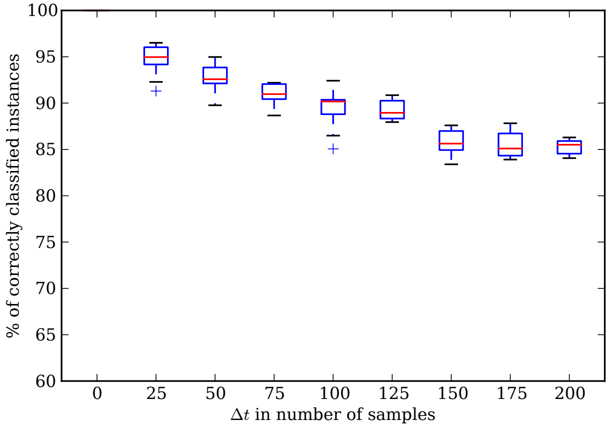


Figure 4.17: Performance of K-nearest neighbour classifier (IBk), data set 20100809

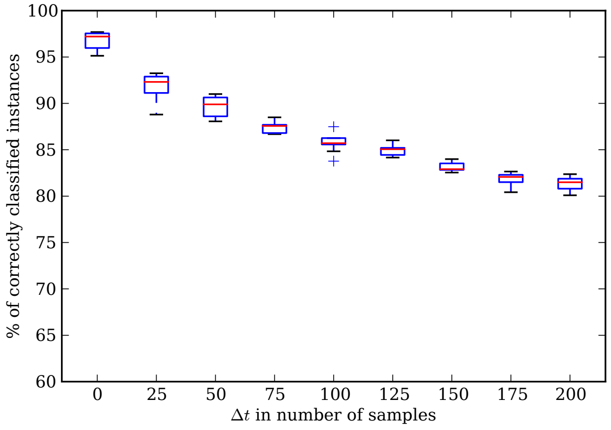


Figure 4.18: Performance of decision tree classifier (J48), data set 20100809

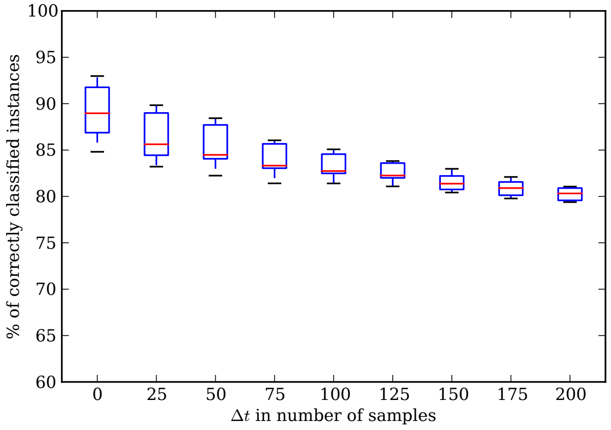


Figure 4.19: Performance of rule based classifier (JRip), data set 20100809

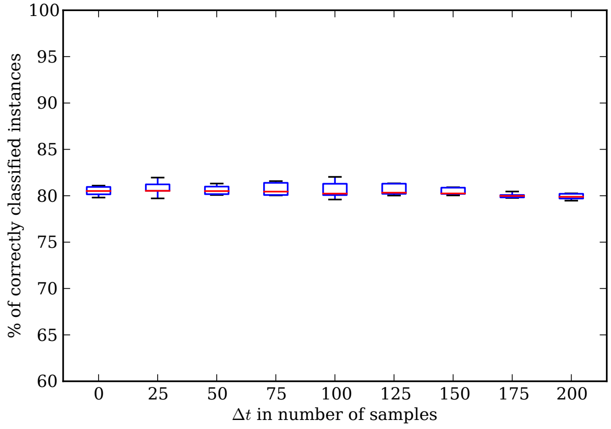


Figure 4.20: Performance of naïve Bayes classifier (NB), data set 20100809

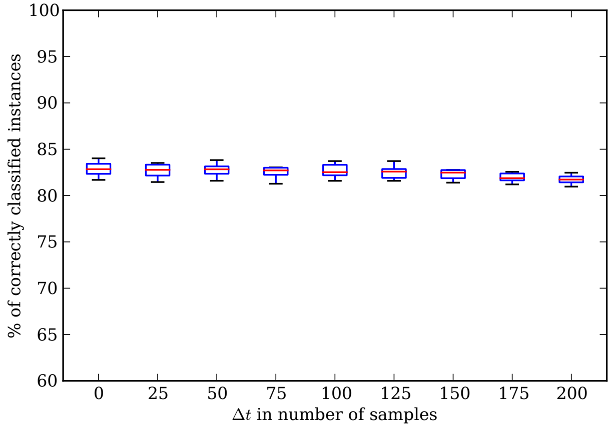


Figure 4.21: Performance of SMO classifier, data set 20100809

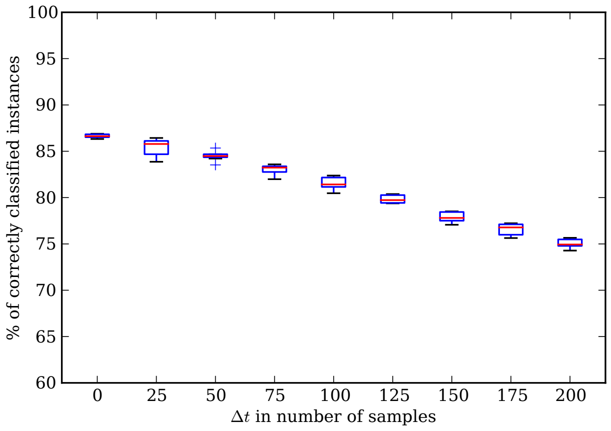


Figure 4.22: Performance of Bayesian Network classifier (BN), data set 20100805

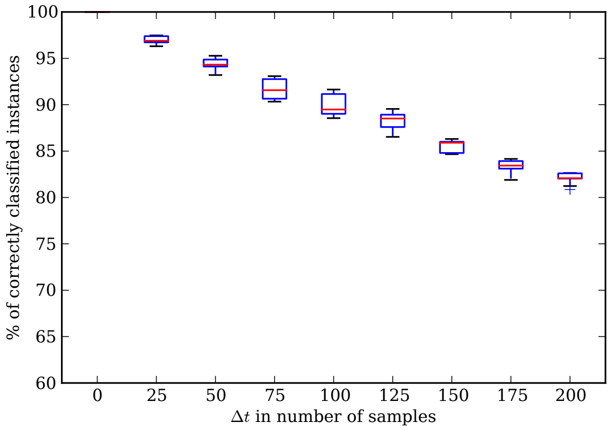


Figure 4.23: Performance of K-nearest neighbour classifier (IBk), data set 20100805

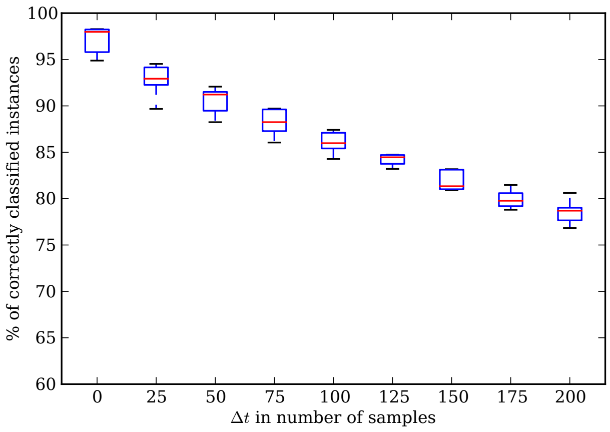


Figure 4.24: Performance of decision tree classifier (J48), data set 20100805

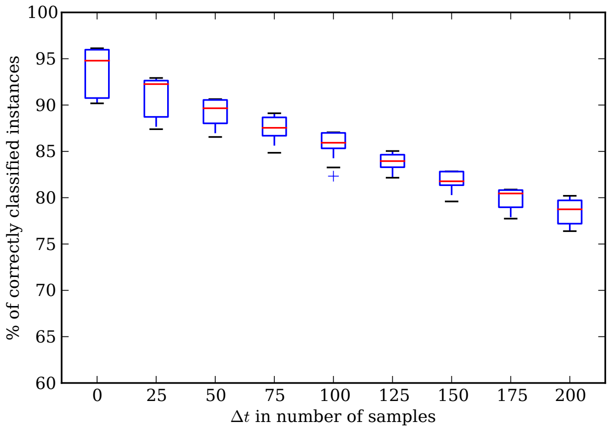


Figure 4.25: Performance of rule based classifier (JRip), data set 20100805

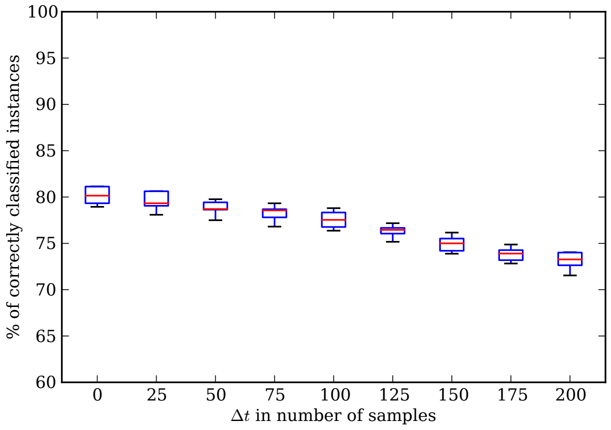


Figure 4.26: Performance of naïve Bayes classifier (NB), data set 20100805

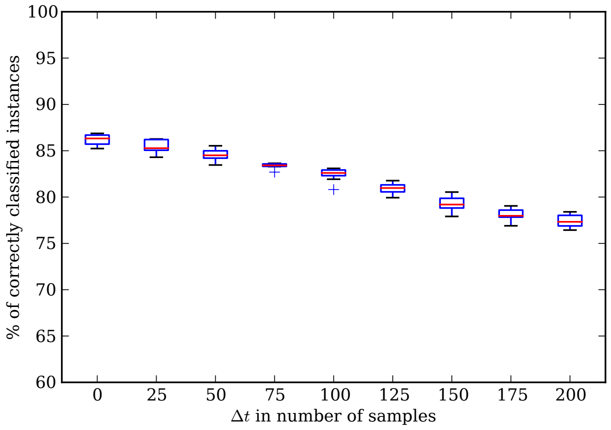


Figure 4.27: Performance of SMO classifier, data set 20100805

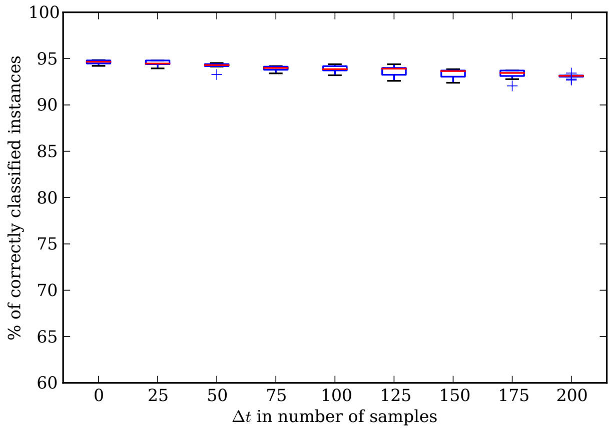


Figure 4.28: Performance of Bayesian Network classifier (BN), data set 20100811

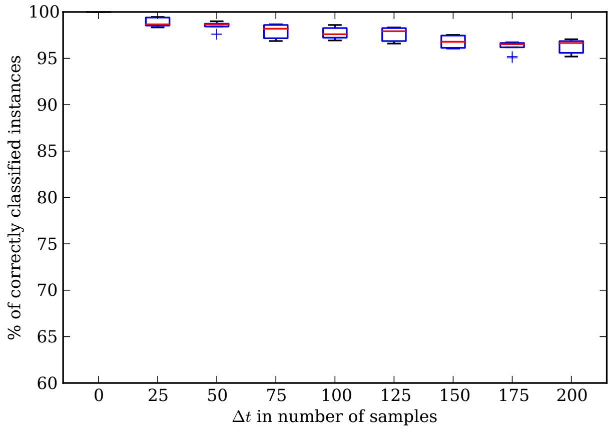


Figure 4.29: Performance of K-nearest neighbour classifier (IBk), data set 20100811

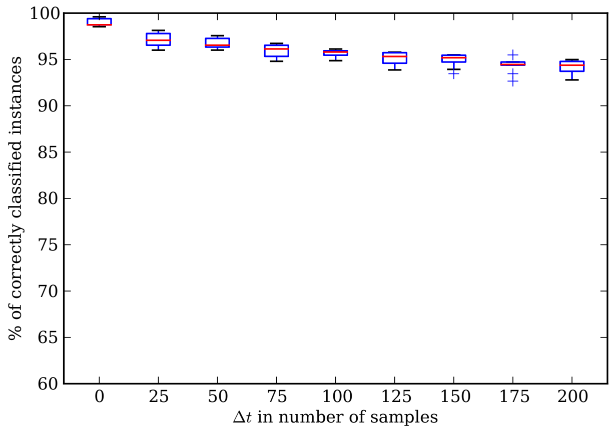


Figure 4.30: Performance of decision tree classifier (J48), data set 20100811

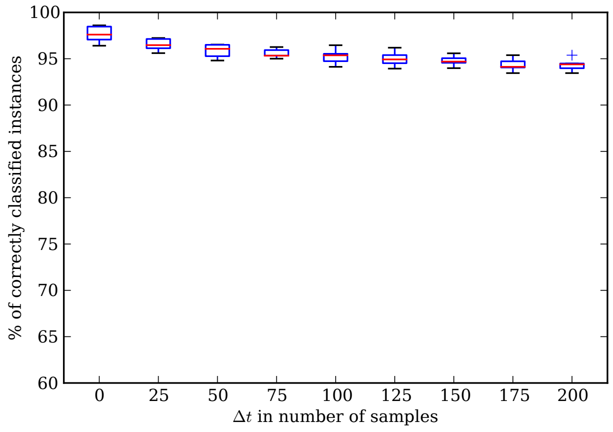


Figure 4.31: Performance of rule based classifier (JRip), data set 20100811

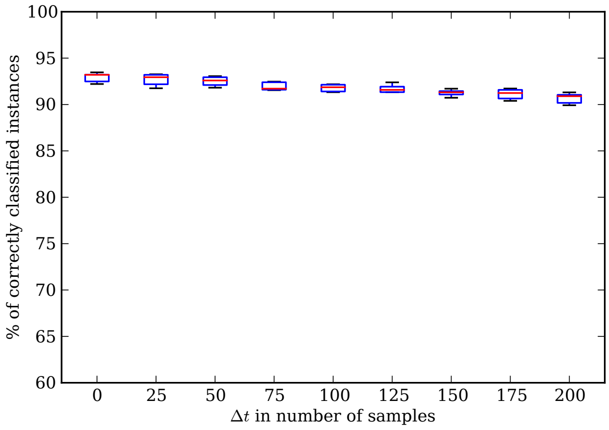


Figure 4.32: Performance of naïve Bayes classifier (NB), data set 20100811

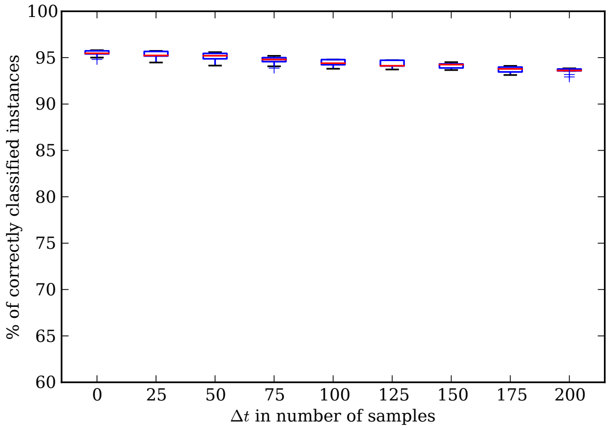


Figure 4.33: Performance of SMO classifier, data set 20100811

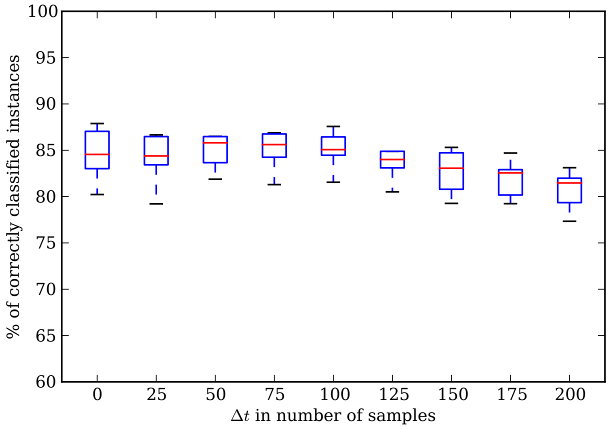


Figure 4.34: Performance of Bayesian Network classifier (BN), data set 20100812

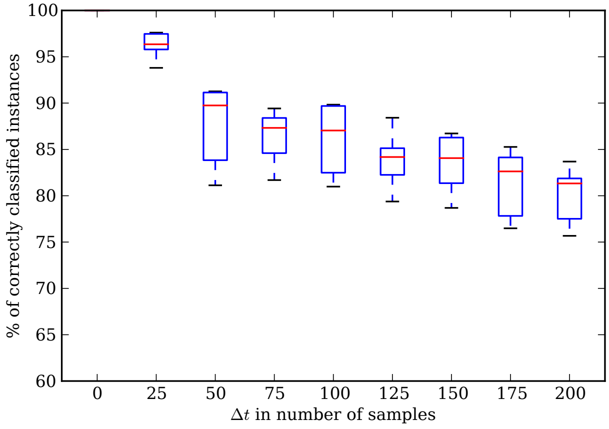


Figure 4.35: Performance of K-nearest neighbour classifier (IBk), data set 20100812

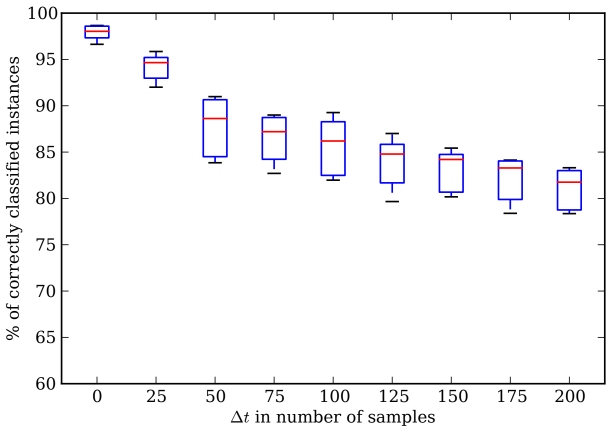


Figure 4.36: Performance of decision tree classifier (J48), data set 20100812

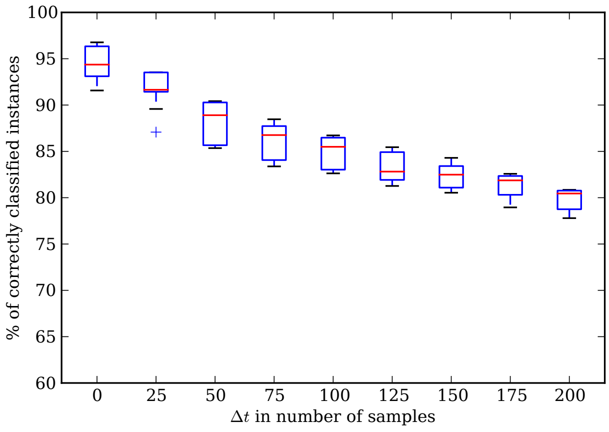


Figure 4.37: Performance of rule based classifier (JRip), data set 20100812

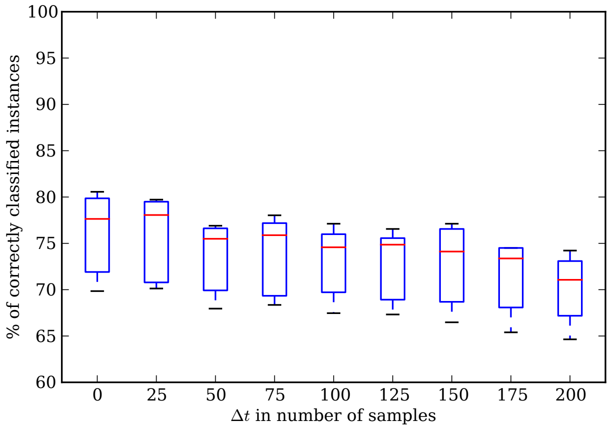


Figure 4.38: Performance of naïve Bayes classifier (NB), data set 20100812

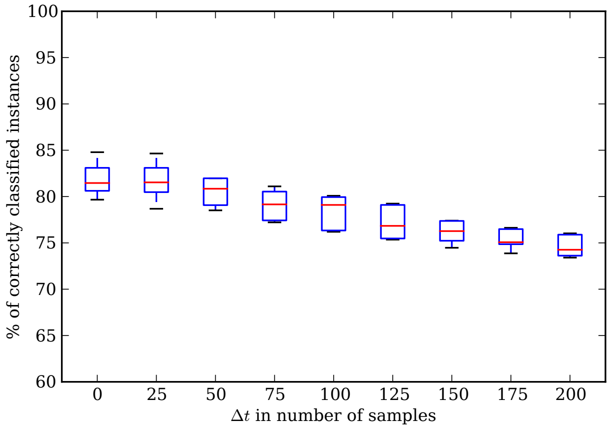


Figure 4.39: Performance of SMO classifier, data set 20100812

	Time shift in number of samples								
Algorithm	0	25	50	75	100	125	150	175	200
BayesNet	87.57	86.85	86.68	86.10	85.53	84.56	83.57	82.77	81.78
IBk	100.00	96.53	93.04	91.47	90.47	89.56	87.64	86.34	85.71
J48	97.60	93.76	90.99	89.32	88.12	86.93	85.59	84.37	83.62
JRip	93.38	91.13	89.42	88.05	87.03	85.99	84.92	83.98	83.16
NaiveBayes	82.09	81.87	81.10	80.88	80.49	80.09	79.54	78.81	78.23
SMO	86.48	86.17	85.60	84.80	84.36	83.59	82.89	82.22	81.73

Table 4.3: Average correctly classified instances in % for different algorithms in relation to the time shift. The average is calculated out of four data sets with a total length of 1:33 hours and eight parameter settings per algorithm (see Table 4.2).

The IBk classifier is an instance-based classifier. It compares the distance (usually Euclidean distance) between instances in the training data to find the nearest neighbor for the desired recognition. Therefore, it performs 100 % accurate on the un-shifted data but performs on average 4.47 percentage points worse for a shift of 25 samples and 14.29 percentage points worse for a 200 samples shift (Figure 4.11, note that in the un-shifted case training and test data is the same).

Both BN and NB are classifiers that use probability between the attributes to classify the activity. Figures 4.10 and 4.14 show that the accuracy of both classifiers also suffer from time shifts. NB's accuracy only goes down by 3.86 percentage points in the worst case, but also has the worst accuracy of all tested algorithms even with no time shift. NB's accuracy with no time shift is less than the worst case accuracy of IBk, J48 and JRip.

The decision tree classifier J48 is suitable for data instances that are describable with attribute-value pairs. It uses a divide-and-conquer strategy to discover the relationship between the attributes and the respective activities. The rule-based classifier JRip uses "if-else" relationships for the recognition. Both J48 and JRip classifiers are able to produce descriptive models. Both achieve high accuracy in case of no time shift, with J48 slightly better and both suffer from time shifts (Figures 4.12 and 4.13) but are in the worst case still better than BN, NB and SMO.

The SMO algorithm is an optimized method to apply the support vector machine (SVM) method on the training data for classification. A SVM classifier finds hyperplanes in linearly separable data, where a hyperplane is defined as a linear function that separates the data into two groups in space (in the experiments a polynomial kernel is used). It performs only slightly better than NB, with 86.48 % accuracy for no time shift and going down 4.75 percentage points in the worst case (Figure 4.15).

All in all, the results show that classification learning algorithms may not be susceptible to time synchronization as long as the timing differences remain stable, but are vulnerable if there is a time offset between the involved sensor nodes after the training phase, which may be due to the clock drift observed in Chapter 3. This shows that, for classification learning algorithms, architectural support for handling time synchronization in context aware application can be beneficial, as shown for user defined rule based reasoning (Chapter 4.1).

4.3 Conclusion

The results show that for user defined rule based reasoning, if the sampled data is not synchronized it may lead to inconsistencies and therefore bad reasoning accuracy. We also evaluated the influence of time synchronization on classification learning algorithms. The evaluation shows that classification learning algorithms may be less susceptible to timing differences if timing differences remain stable after the training phase, but are as vulnerable to changes in time synchronization as user defined rule based reasoning is.

Of course not all applications of context awareness are affected by the observed problems. In order to be affected, an application has to have at least some of the following characteristics:

- Data of more than one sensor is aggregated.
- The order of events is important.
- Single sensor readings are of no meaning, rather change over time is observed.
- The sensed information is changing twice as fast as or faster than the changes can be sensed.

We believe that not all but many context aware applications have these characteristics, e.g. recognition of movements using multiple sensors, as we have shown, suffers from insufficient time synchronization. For these applications the identified influence factors on time synchronization analyzed in Chapter 3 will influence the accuracy of reasoning approaches. The results support the claim that architectures for context aware applications need to support ways of handling the uncertainties caused by insufficient time synchronization or ways to avoid insufficient time synchronization.

Chapter 5

Requirements of an architecture for context awareness

In Chapter 2.2 an overview of the state of the art of context aware architectures is given. From the literature the following characteristics of context aware applications that have to be taken into account during the design of an architecture, are subtracted:

- Changing connectivity
- Dynamic resource availability
- Multitude of different sensor interfaces
- Different sensor characteristics
- Limited resources

For a detailed description of these characteristics see Chapter 2.2. Most architectures are targeted at special sensor nodes and even if they all foresee the support for heterogeneous devices, the use of a common smart phone adds some challenges to the design of context aware architectures. One could argue that smart phones are just another heterogeneous sensor device. However, the smart phone on the one hand has less limited resources than most special purpose sensor devices, on the other hand smart phones have limited API support and thus their utilization in a context reasoning process is limited.

Smart phones have limited API support or even restrictions on who can access the API for good reasons, the smart phones primary function is to act as the user's private device, mainly targeted at fulfilling the user's communication needs. The utilization in context reasoning processes thus can only be an additional task that should not interfere with the main task and has to protect the user's privacy. For example, most users will not tolerate having their smart phone's clock set by an external application, at least not without the user's confirmation.

The availability of the smart phone's resources may be more restricted than that of the resources of the special purpose sensor devices. Thus, all characteristics and requirements have to be addressed in a way that ensures minimal interference with the smart phone user's interest in the primary tasks of his smart phone and the integrity and privacy of his data, which is stored on the phone.

In the following, the functional requirements and software qualities that should be fulfilled by a context aware architecture are discussed, keeping in mind the special challenges of smart phone utilization and the time synchronization needs identified in the previous chapters.

5.1 Functional requirements

First, the functional requirements of a context aware architecture are discussed. To fully understand the functional requirements, the process of context reasoning has to be decomposed to single processing steps.

The processing steps are (see Figure 3.10):

1. Retrieval of the raw sensor data – The first step of every context reasoning process is to retrieve the sensor data. This can either happen using a polling approach or a notification based approach (publish/subscribe). The data retrieved from the sensor is raw, that is it is not yet processed in any way and thus often of little meaning to any human user.

2. Pre-processing of the raw data – Raw data almost always needs pre-processing before it can be used in the next steps of the reasoning process. For example filtering data or calculation of features using time series of data is regarded as pre-processing in this thesis.

3. Calculation of low level context data – In order to make sense for human users the pre-processed data has to be further processed. In this step data from sensors is translated to a more meaningful unit, e.g. from voltage to °C for a temperature sensor. In this step data from several sensors can be aggregated.

4. Abstraction to high-level context data – Low level context data can be aggregated and further processed in order to retrieve high level context data, also the calculation of high level context can take user preferences into account. For example the aggregation of multiple environmental sensors, like temperature, light, and humidity sensors, can be used to derive a high level context describing the room climate; however, what is a comfortable room climate depends on the user's preferences.

The step 4 is not necessarily occurring only once, but can be repeated several times, while the calculated output becomes evermore high level. All steps consume processing resources and thus time and energy can be saved if the processing steps are distributed in the network of available pervasive devices. It is possible that all steps are performed on a single device, like a smart phone, but pervasive environments will offer a multitude of available devices. These devices will differ in terms of processing power, energy supply and other resource's availability; hence a design goal of a context aware architecture has to be to enable the distribution of the processing load in a way that ensures the timely processing while saving resources, especially of the user's devices and of battery powered devices.

5.1.1 Encapsulation of sensor interfaces

It is a common understanding that one main functional requirement of context aware architectures is to encapsulate the interfaces of the sensors. The sensors used in pervasive environments are heterogeneous in terms of vendors and capabilities, and thus will not have a uniform interface. In order to ease the development task these differing interfaces are to be encapsulated to enable easy reuse of the sensors.

All architectural approaches discussed in Chapter 2.2 strive to encapsulate the sensor interfaces. However, there is no detailed explanation in the literature how the encapsulation is exactly done and how a uniform interface may look like or which functionality it exposes.

Generally, in the theory of object oriented programming an interface of an object describes the properties and the behavior of the object [68]. Hence, the properties and behavior common to all sensors has to be defined. Moreover, it may be necessary to identify classes of sensors that can share a uniform interface, because sensors may be different in a way that may not allow to encapsulate them with one uniform interface.

5.1.2 Distributed processing and load distribution

As already outlined in the beginning of this chapter, in future pervasive environments there will not only be a multitude of environmental sensors but all kinds of computing resources available. The resources may belong to and be carried around by the users or be installed in the environment or even be moving around autonomously. Some of these devices may require the user to pay for the use of the provided resources, some may be intentionally harmful.

It is therefore a functional requirement of context architectures to enable the discovery and utilization of the resources. Above all, it is necessary that the resources are used as efficient as possible, regarding monetary costs, energy costs, processing time, data integrity, and security.

To achieve this, the first requirement is that processing steps are performed by loosely coupled software objects that can be moved from one resource to another. That implies that not only the sensors have to have uniform interfaces, but also the algorithms used in the processing. If the architecture should be programming language independent, the algorithms have to be described using a separately defined document format, the interpretation of those documents then has to be implemented in every programming language that should be supported.

When the used algorithms are movable in some way, there need to be a way to query the resources for their status, for example processing load and remaining battery power. And at last, there need to be one instance that orchestrates the distributed data flow. That is, applications will have to specify the context they are interested in and the orchestrating instance will assemble the needed processing steps and distribute them on the available resources. When resources disappear the compensation has to be handled by finding alternative resources and changing the data flow accordingly.

Out of the architectures discussed in Chapter 2.2 only the SOLAR approach foresees a distribution support that fulfills these requirements [22].

5.1.3 Dynamic environment support

The dynamic characteristic of pervasive environments, in which new resources may appear and disappear all the time, imposes some additional difficulties that an architecture should support to handle. As already implied in the introduction to this chapter, the dynamic is due to the movement of users, carrying around several devices and entering physical spaces in which devices are deployed. Additionally, devices may be able to move around themselves and also devices may disappear because batteries are dying or because of hardware failures of the devices.

Connectivity is another always varying factor in pervasive environments. Depending on the current location different communication channels may be available, at different costs. The communication channels will offer different bandwidth, depending on technical restrictions and current load.

To face the difficulties of these dynamic, pervasive environments, an architecture for context awareness has to support the discovery of resources. That is, if a user enters a new physical space, the user's devices need a means of learning about the available devices in that space and how to utilize them. Additionally, during the stay of the user, the information about available resources has to be continuously updated as resource availability will change over time. All architectures proposed in the literature and discussed in Chapter 2.2 acknowledge this requirement.

Secondly, there need to be architectural support for the seamless change of communication channels. The architecture should be able to provide an abstraction of the available communication technologies and free the application developer from the burden to deal with the changing availabilities and costs. The architecture should choose the technology used to fulfill the application's needs transparently and handle handover and such.

While there are approaches in the literature to fulfill this last requirement, it cannot be found in detail in the literature published about architectural approaches towards context awareness (Chapter 2.2).

5.1.4 Time synchronization

The one requirement that is not recognized by any of the architectures described in Chapter 2.2 is the support for time synchronization. One of the main contributions of this thesis is the analysis of the impact of time synchronization on reasoning accuracy. In Chapter 3 there are argued several reasons that can cause time synchronization related issues. In Chapter 4 an in-depth analysis of the impact on reasoning accuracy concludes that the accuracy may significantly suffer from clock drift and other time synchronization related issues. Therefore, as it is the purpose of a middleware architecture to transparently handle system specific difficulties, architectures for context awareness should also enable the handling of time synchronization issues.

Handling the issues can happen in several ways, depending on the involved resources and their capabilities and on the application needs. First of all, the involved resources may not offer any time information at all. In that case, at least the applications, if they require time information or specify a desired synchronization accuracy, should be informed about the lack of those.

Secondly, almost all devices will be offering time information, but many will not provide any means of adjusting the clocks as it is required by synchronization algorithms. In that case the architecture has to establish a relative time ordering without adjusting the resource's clocks, e.g. by maintaining a table of relative clock offsets.

And last, if it is possible to establish a relative time ordering, the accuracy may be sufficient for some applications and not sufficient for other applications. In that case, the architecture has to either increase the accuracy if possible or to inform the application of the insufficient accuracy. Therefore, some way to specify the accuracy requirements of applications has to be specified.

In Chapter 6 an architectural approach that will especially address the time synchronization needs depicted here, along with the aforementioned requirements, is proposed. Before proposing that approach, the definition of the requirements has to be completed by discussing the software qualities (or non-functional requirements) of an architecture for context awareness.

5.2 Software qualities

Software qualities, also called non-functional requirements, are characteristics that a software should have that are not describing the software's primary functions but rather attribute characteristics to how the primary functions should be performed. Generally, every software should fulfill as many of the software qualities as possible, for a detailed list of the many software qualities see [33]. Out of the long list of software qualities, the following three are especially important and challenging when it comes to software architectures for context awareness.

Security and privacy are also often listed under software qualities and both are arguably of very high importance for context awareness; however they are out of the scope of this thesis.

5.2.1 Maintainability

Maintainability is a quality of software that expresses how easy or difficult it is to maintain the software. Maintenance of software here refers to bug fixing, extending and changing the software. It is hard to find an absolute measure of maintainability, however it can be measured as the time a particular developer needs to fulfill a maintenance task compared to the time the same developer needs for the same task when a different software is maintained.

While maintainability is a desirable feature of every software, it is of special importance for software used in ubiquitous environments, like context aware applications. Due to the high number of involved resources, users and applications and the high dynamic, the software architectures in ubiquitous systems are complex, which increases the possibility of bugs. Also the variety of user preferences and the fast evolving technologies cause a frequent change of the requirements a software architecture has to address and thus maintenance tasks will be frequent.

There is no easy receipt how to design software so that it is maintainable [69], though there are the following principles that are regarded as best practice to most likely achieve good maintainability:

- Modularization – The partition of software into modules or components that are loosely coupled allows software maintainers to work on the single components that are smaller than the whole software, without the need to completely understand the other components. [70]
- Use of software patterns – The use of well know solutions for recurring problems that are derived from the observation of practitioners (so called software patterns) ensure that maintainers recognize the patterns and thus can understand the code more easily. [71]

- Use of standards – Software that uses standards, e.g. standard data exchange formats or standard communication protocols, also ensures that maintainers are able to more easily understand the code, because they may know the standard or will find additional documentation resources and alternative implementations of the standard.
- Following conventions – Code that is not following conventions will be irritating and harder to read for developers that are accustomed to the convention.
- Documentation – Clearly, the more the code is documented the more likely a maintainer will understand the code and the intention of the original developer.

Unfortunately there still is a lack of standards for context awareness, e.g. there are no standards for describing, storing and exchanging context, and there are no standards to describe sensor interfaces and capabilities. To cope with foreseen future multitude and variety of sensors, the establishment of those standards will be inevitable.

5.2.2 Scalability

Scalability of software is how well the software performs when the number of involved resources, users, and requests increases. Performance here is measured in response time and memory consumption. As for maintainability the main determining factor for scalability is the sheer number of devices foreseen in ubiquitous environments, software that only performs with a small number of devices is useful for testing and proof of concepts, but simply useless in real deployments of context aware applications.

To achieve high scalability the following measures can be taken:

- Efficient algorithms – Doubtless, if running an algorithm once is consuming less CPU resources and energy, a hundred times running this algorithm will also consume fewer resources. Note that the savings may not be linear, for example on the one hand caching of data may cause that two times running an algorithm is faster than the run time of one run times two, and on the other hand scheduling algorithms and interrupt times may cause an opposite effect.

- Redundant (centralized) infrastructure – Every resource has an upper limit where it is no longer able to server any more requests, thus if parts of an infrastructure are central to the system, they limit the overall capacity of a system, unless they are redundant and the load can be distributed. The ability of transparent load balancing needs to be planned at design time, otherwise the introduction of those techniques may be costly. For example, if a list of resources is foreseen to be held in every device's memory, at a certain amount the device's capabilities will be exceeded, therefore the distributed storage along with a selective update mechanism has to be planned.

- Asynchronous communication protocols – If system components handle communication synchronously, the components have to wait for answers and are blocked, i.e. they cannot perform any other tasks while waiting for the answer. This blocking of resources that could be otherwise of use will reduce a systems performance and hence the scalability. On the contrary, asynchronous communication allows for the queuing of massages and the dispatching of multiple workers to answer the messages that will ensure a more efficient use of the systems resources.

- Self-awareness and self-adjustment – Of the self-*requirements known from autonomic systems [72], the self-awareness and self-adjustment requirements are two requirements that will aid the scalability. If a system knows its internal status, especially load, there is a chance that an ubiquitous system may adjust, e.g. by acquiring additional resources or releasing resources from the environment.

To ensure that the proposed measures have the desired effect on scalability continuous testing during the implementation phase is inevitable.

5.2.3 Reliability

The last software quality that should be emphasized here is reliability. Reliability of software is how often the software performs correctly, correctly meaning an answer is given to a request and that answer is the answer the developer intended. More important, reliability from a user's perspective is that a software behaves as expected.

The fulfillment of user expectations is of special importance, given to what extent context aware applications will involve in the user's everyday life. In fact the success of context aware applications depends on the reliability of these applications; if users are expected to trust a software that will make decisions on their behalf, and likely they will not be able to understand the internals of the software, they will only do so, if the software behaves as they expect and will not fail them.

Reliability is hard to achieve for complex systems, but the following measures can be taken to raise the likelihood that a software system is reliable:

- Testing – Continuously testing every piece (or unit) of the software during the development process is vital to ensure reliability. Unit testing frameworks, often integrated in IDEs, help developers to automate these tasks. However, the test cases have to be specified and it is important that functions are tested, especially when unusual input values are given.

- Two-person integrity – Whether it is during the implementation, as in peer-programming, or afterwards, it is always beneficial if a second person carefully looks on what the first person has implemented.
- Quality assurance – Additionally to the two aforementioned measures, a well-defined quality assurance process that is to follow during design and implementation of a software can increase reliability.
- Software reuse – New implementations may contain new bugs and will need extensive testing, while the reuse of software that has earned merits in real deployment and has already undergone testing by real users, potentially with several bug fixes applied, will proof more reliable in many cases. Additionally, the size of the developer community or company behind a software product may (but must not) hint to the software's maturity.

Of the four mentioned measures to increase the likelihood of high reliability, only the last is possible to take during the design phase of a context awareness architecture. The other three are methodical measures that have to be taken by the developers using the architecture.

In the following chapter an architecture addressing the described requirements and focusing on a solution of the time synchronization issues is proposed and evaluated.

Chapter 6

Directed acyclic graph based reasoning

In Chapter 2.2 an overview of approaches towards context awareness architectures is given. None of these architectures considers the need to handle time synchronization of sensor devices. In the Chapter 3 We discuss the various reasons that can cause a lack of time synchronization and in Chapter 4 it is shown that insufficient time synchronization and the issues caused by this will affect the reasoning accuracy negatively. Also, we discuss why the time synchronization approaches known from WSNs and also that the Network Time Protocol will not solve the problems in every case (see Chapter 2.1).

In the following we discuss a conceptual approach towards an architecture that can handle the time synchronization in cases where traditional approaches, as discussed in Chapter 2.1 will not work. Additionally, where appropriate, hints are given, how the approach will fulfill the other requirements discussed in the previous chapter, still this chapter focuses on the time synchronization.

While the approach is only conceptual, it is partially implemented and experiments are made to prove the claims regarding the time synchronization. This partial implementation is done in Java, however, the approach is and has to be independent of programming languages. Therefore, we restrict the discussion to communication paradigms and protocol level.

The analysis of the approach is based on the general abstraction of the reasoning process as shown in Figure 3.10. The sensor data flows through several processing steps; every step consumes time and at some steps sensor data from several other processing steps or sensor sources is aggregated. At this aggregation points it is important that only sensor data is combined in the calculation that is originating from the same time frame (see Chapter 4).

This chapter is outlined as follows: First an alternative approach taken from the research field of streaming databases is discussed for the purpose of comparison. Then the approach central to this thesis is detailed. At last, the two approaches are compared theoretically and experimentally.

6.1 Stream barrier approach

In order to compare the approach of this thesis, a second approach is considered. This second approach is taken from the research area of streaming databases. To the best of our knowledge, this approach has never been considered in the research on context awareness, and none of the architectures discussed in Chapter 2.2 considers time synchronization at all. However, to those familiar with streaming databases, the use of so called stream barriers is an obvious solution to some of the time synchronization issues in context awareness that are discussed in this thesis.

Generally, the data flow of sensor data from sensor devices, through several processing steps to the application, is very similar to the data flow in streaming databases. When database content is streamed it will also be processed in several steps before it reaches a subscriber interested in the content. The content here is the same as sensor data and the processing steps, in streaming database mainly filtering and translation steps, also can aggregate data from several streams.

In streaming media and streaming database applications time stamps are attached to the data. At aggregation points so called stream barriers are inserted. At these stream barriers streams are delayed until other streams catch up [73]. This is done using a simple algorithm that compares the time stamps. Sensors can publish the data in streams to the subscribers as usual. The stream barrier buffers all streams. Than it takes the heads of the streams to check if their time stamps differ only by the allowed Δt. If not, the oldest sample will be replaced with next sample from the corresponding buffer to check again. This way only samples with fitting time stamps are used for the reasoning.

This approach can be used if the sensors do not support polling. The downside of this approach is that only a certain percentage of samples have fitting time stamps (as described in Chapter 4). Obviously, for this approach the sensor devices still need synchronized clocks. The approach proposed in the following can handle unsynchronized sensor devices.

6.2 DAG approach

To enable synchronized processing we propose a directed acyclic graph (DAG) based approach [74]. Processing steps are nodes in a DAG, sensors are sources and applications connect to the sinks.

Definition 10. *A DAG is a directed graph without cycles. Nodes without ingoing edges are sources and nodes with no outgoing edges are sinks. The depth of a node is the longest path from a source to that node and the height is the longest*

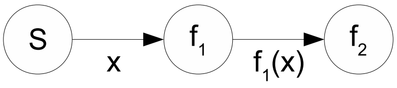


Figure 6.1: Part of a DAG with sensor S and two processing steps.

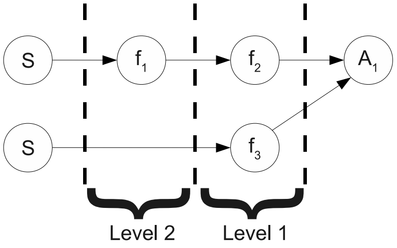


Figure 6.2: Reasoning-stages in a distributed acyclic graph that represents devices in a distributed environment. Two sensors are processed in two processing levels before the application A_1 gets the data. The first level has only one calculation f_1.

path from that node to a sink. The length of a DAG is the length of the longest path.

If two nodes are connected by a directed edge, the result of the calculation from the first node is an input parameter for the calculation of the second node. For example, in Figure 6.1 the calculation result of f_1 is the input for the calculation of f_2: $f_2(f_1(x))$.

To enable synchronized processing, nodes in the graph are organized in processing levels. A processing level contains all nodes that have the same height (see Figure 6.2). The level of each node can be calculated, for example, by the simple recursive algorithm.

Algorithm: Node level calculation

```
calcSubgraph(Node n, int height){
   for (iterator i =
         nodes.getIterator();
      i.hasNext()){
         subgraph.add(n, height+1);
         calcSubgraph(((Node)
            i.next())
               .getParentNodes(),
               height+1);
       }
    }
calcSubgraph(A1,0);
```

The reasoning calculation is done in processing turns. In the first turn the data of all involved sensors is captured. Afterwards, starting form the highest level (i.e. the node with the longest path to the application/sink) the input values for nodes of the next level are processed. The next samples are triggered by the nodes of the highest level not before all node of this level have completed their calculations. This is a polling approach in contrast to the normally used publish/subscribe.

For instance, if the data from one sensor is processed by processing steps f_1 and afterwards f_2 and data from another sensor necessitates only one processing step f_3, f_2 and f_3 are processed after f_1 is already completed, since f_1 belongs to a lower level. The calculation in one processing turn can be distributed among several processing units.

W.l.o.g., we assume that processing steps have the same time complexity, since processing steps of higher time complexity can easily be modeled by multiple (sub-) processing steps. If two devices are available, distinct processing steps can be computed in parallel. The system captures the data of all sensors of which data is aggregated at the same time and afterwards starts the calculation of processing turns. At any processing level, only data captured at the same time instance

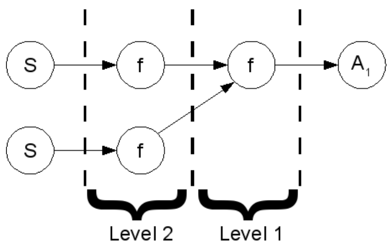


Figure 6.3: DAG where data from one sensor needs two processing steps and from the other only one processing step.

arrives, and the following processing steps are informed about the new input data only when the calculation of all nodes at one level is finished.

The turn based, synchronized processing ensures that a calculation of a succeeding turn is delayed until all calculations in the preceding turn are completed. All captured samples have fitting time stamps from the other sensors and no samples are unnecessarily taken. Figure 6.3 illustrates this property. An undefined state is therefore impossible, since all calculations are based on sensor data that is captured at the same time instant. The remaining sources for erroneous results for values of identical contexts are then erroneous measurements or errors in the processing units. The possibility that discrete context values are calculated due to non-synchronized context processing is eliminated by the DAGR approach.

Consequently the sampling rate is as slow as the slowest node in the DAG. Samples are automatically taken at the same time frame only differing by the network sending time. This leads to only fitting time stamps as long as there are no network timeouts. If there are timeouts, all sensor data has to be requested a second time.

The downside of this algorithm, along with the dependency on the slowest sensors, is that the sensors have to support polling of data and that a controlling instance is needed. This instance can be part of a processing step, but surely needs some extra implementation effort.

The benefit of the approach is that to retrieve time stamps from heterogeneous devices, this devices only need to support polling of data. There is neither a way needed to register for sensor updates, nor is there a need for time stamps originating at the sensor device. Hence, the clock synchronization of the involved sensor devices is irrelevant, the DAGR approach ensures relative ordering, and thus synchronized processing, by protocol design.

We have partially implemented both approaches, DAGR and the stream barrier approach, to evaluate the effect on energy consumption. Chapter 6.3 presents the evaluation results.

6.3 Evaluation

In Chapter 4 the issues caused by insufficient time synchronization are analyzed; in this chapter we evaluate how the DAGR approach solves the identified issues and compare the results to the stream barrier approach. Additionally, the energy consumption of both approaches is compared. In order to evaluate the two approaches, the following scenario is handled by both approaches.

To sense whether a person is standing or sitting, two accelerometers are used. One accelerometer is placed in a person's pocket on the side of each leg. The accelerometers are part of the SunSPOT sensor node [75]. These small battery (3.7 V, 720 mAh) powered devices run a JavaVM directly on Amtel AT91RM9200 (ARM9) hardware and have, amongst others, a built-in accelerometer (ST Microsystems LIS3L02AQ, 2g sensitivity) and light (Toshiba TPS851) sensor. The devices are able to send data over IEEE 802.15.4

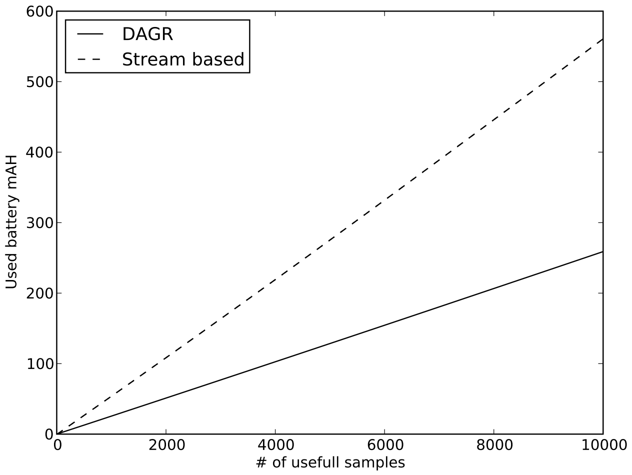


Figure 6.4: Comparison of energy consumption per useful samples over time.

radio connections to a base station.

The three axis of the accelerometers are used to calculate the tilt Θ of the axis pointing to the earth when a person is standing a_{earth} ($\Theta = \arcsin \frac{a_{earth}}{|\bar{a}|}$, total acceleration is $|\bar{a}|$). To find out which axis is pointing to the earth the gravitational acceleration helps. When the person is standing still, the axis pointing to the earth is the only axis that measures acceleration of approximately 1g. The light sensors are used as control instance, only when it is "dark" around the SunSpot, it is assumed that the device really is in a pocket (see Figure 6.5). There are some other cases when a person angles both legs, those cases are not considered, however, to extend the algorithms to detect that the person is jumping, the total acceleration $|\bar{a}|$ can be considered; to detect that a person lays down a second sensor attached to the upper body will be sufficient.

In our experiment, to sense whether a person is standing or

```
if ((-0.5 < sample2.getTilt())
   & (sample2.getTilt() < 0.5)
   & (-0.5 < sample.getTilt())
   & (sample.getTilt() < 0.5)
   & sample2.getLight() < 1
   & sample.getLight() < 1) {
   System.out.println("Sitting");
} else {
   System.out.println("Standing");
}
```

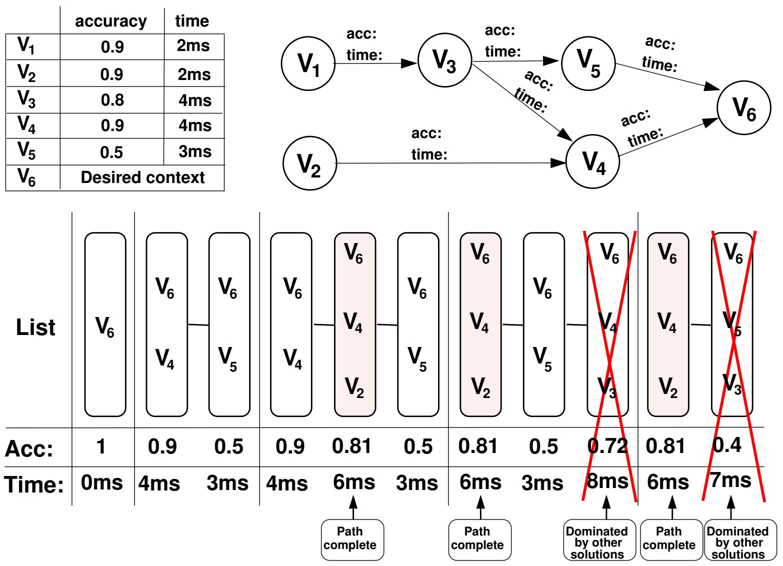
Figure 6.5: Simple algorithm to calculate if a person is standing or sitting.

sitting, the two sensors' sample rates differs by factor 1.5. Such a difference may be possible for several reasons (see Chapter 3). With the DAGR approach both sensors are constantly polled for new samples and therefore all samples' time stamps taken are fitting (the directed acyclic graph has no more levels here). With the stream barrier based approach $\approx 66\%$ of samples taken from the faster sensor did not have fitting time stamps ($\Delta t < 0.5R_1$, see Chapter 4). The energy consumption average is 0.17mAh over an experiment duration of 12 hours for both approaches. Figure 6.4 shows that the energy consumption per useful sample consequently is significantly better for the DAGR approach.

6.4 Optimized DAG creation

Given the idea of directed acyclic graph reasoning, one main concern is how to derive the graph that describes the reasoning process [76]. Furthermore, when several solutions for context composition exist, how to determine the desired context with the optimum solution for all objective functions?

Typical objectives are to minimize the computation time, energy consumption or error probability as well as to maximize



	accuracy	time
V_1	0.9	2ms
V_2	0.9	2ms
V_3	0.8	4ms
V_4	0.9	4ms
V_5	0.5	3ms
V_6	Desired context	

List	V_6	V_6 V_4	V_6 V_5	V_6 V_4	V_6 V_4 V_2	V_6 V_5	V_6 V_4 V_2	V_6 V_5	V_6 V_4 V_3	V_6 V_4 V_2	V_6 V_5 V_3
Acc:	1	0.9	0.5	0.9	0.81	0.5	0.81	0.5	0.72	0.81	0.4
Time:	0ms	4ms	3ms	4ms	6ms	3ms	6ms	3ms	8ms	6ms	7ms
					Path complete		Path complete		Dominated by other solutions	Path complete	Dominated by other solutions

Figure 6.6: Illustration of the operational principle of the branch and bound algorithm [76].

the reliability of context providers. In practical applications, several objectives can impact the decision for an optimum alternative. When various objectives exist, it is looked for Pareto optimal solutions that are superior or equal to all other solutions in at least one objective.

6.4.1 Derive the directed graph

For a device to derive a graph that describes feasible context reasoning sequences of context providers in the device's proximity, the available context providers (sensor devices) are to be discovered.

For example FAME2, a distributed middleware for mobile devices, which can be used in conjunction with DAGR, enables such service discovery [77]. When services are discovered they provide a service description that details the service provided. On obtaining a request, we assume that context providers forward information about their service as well as information about services in their proximity which can also be extended to several hops. We also assume that this information is forwarded by service descriptions each context provider offers.

From this information about accessible context providers, services, properties and local proximity of further context providers, the requesting node creates a graph $G = (V, E)$ in which the vertices represent distinct context providers (see Fig. 6.6).

Every vertex has one or more ingoing ports and one outgoing port that represent the input and output context types accepted by the context provider. Edges are directed between two vertices that are in direct proximity from outgoing to ingoing port when the context provided at the outgoing port matches the context required at the ingoing port. Edges (v_i, v_j) are labelled with the costs (related to the distinct objectives) of utilizing context provider i. Figure 6.6 depicts an example graph.

The cost for creating this graph can be estimated by the maximum number of edges in the graph. For n nodes (or context providers) this is at most $\frac{n-1}{2}n = O(n^2)$ when the graph is fully connected.

The algorithm then takes a greedy approach to extend the cheapest path iteratively by one node. When a path is described completely, all paths that are more expensive in all objectives are disregarded.

6.4.2 Derive the optimum reasoning sequence

The constructed graph may contain several sequences of context providers that lead to a desired output sequence. Since only (one of) the optimum solutions needs to be utilized, distinct solutions have to be identified and weighted according to the edge weights.

Assuming that the maximum number of distinct input ports for one context provider is k, the maximum number of ingoing edges is l and the maximum length of a context reasoning sequence is L, not more than L^{l^k} distinct solutions exist. This is bounded from above by $O(n^{n^n})$ since the number of nodes n restricts all these aspects. This worst case complexity time is not acceptable.

In order to optimize the computation time the use of a branch-and-bound algorithm is advised (Figure 6.6).

Algorithm: Branch and Bound

```
Input:
  Ordered, labeled Graph (V,E),
  Start-Vertex: n v
  Objective functions: n f ,..., f 1
Initialise:
  Add vertex n v into a list that is ordered
    by the objective functions

Boolean operate = TRUE;
While (operate){
  Take first path from list
  if (path complete){
     Scan through list until first Pareto
       dominated solution is found and
       remove this and all succeeding
       solutions
     exit
  } else {
     Choose and mark the cheapest unmarked edge
  if (further unmarked edges exist){
     Copy solution
     Extend it by the edge
     Insert new path (solution) into list
```

```
       (with respect to ordering)
  } else
     Insert new path (solution) into list
      (with respect to the ordering)
  }
}
```

The algorithm first starts at the final node – the desired context to be provided by the context providers. All patterns that result in a context other than the desired one can be ignored.

The algorithm then takes a greedy approach to extend the cheapest path iteratively by one node. When a path is described completely, all paths that are more expensive in all objectives are disregarded. Although the worst case computation time is not improved by this approach, it will in typical scenarios exclude a considerable amount of possible paths that cannot be extended to Pareto optimal solutions so that the actual runtime can be improved.

6.5 Conclusion

We analyzed the issue of differing sampling rates in a context aware application where accelerometers are used to sense whether a person is standing or sitting. This difference in timing leads to samples not fitting a corresponding sample from related sensors for a maximum allowed time difference. The theory shows that in the case of a maximum allowed difference of $\frac{1}{10}$ of the sampling rate (accuracy needed to detect human movement), only $\approx 20\%$ of the samples have fitting time stamps in the worst case ($\approx 50\%$ in the best case). This is problematic, because detection of context changes may depend on occurrences of certain patterns around the same time at multiple involved sensors (e.g. movement detection with accelerometers). If the sampled data is not synchronized it may lead to inconsistencies.

Two possible ways to handle the timing issues have been discussed. One approach is based on polling and the organization of the reasoning process in an acyclic directed graph. This allows the application to adapt to the sensor's capabilities and orchestrate the sensing to lead to a synchronized system. The second approach is based on publish/subscribe and uses buffering at barriers to let the streams of sensor data catch up. The second approach cannot avoid the occurrence of samples that are not fitting any other sample's sample time but finds the fitting time stamps in the streams.

The evaluation of the measured data confirms the theory of Chapter 4. Also, the DAGR approach shows to demand 2/3 less energy per useful sample.

To show the practical relevance of the findings, we implemented a scenario of sensing if a person is standing or sitting. Two accelerometers are able to sense this. One is able to operate at higher sampling rate than the other. Both approaches are feasible in this scenario. The polling approach (directed acyclic graph based reasoning - DAGR) is significantly less energy demanding when retrieving a pair of samples with fitting time stamps. The evaluation by measured data proves the theoretical findings.

Chapter 7

Time locality as a new parameter for Quality of Context

Given the dependency of reasoning accuracy on proper time synchronization (viz. Chapter 4), it is obvious that applications and application developers should get information about the time synchronization properties of a reasoning process. There are already a set of parameters defined that strive to rate the quality of a context date. The parameters are known as Quality of Context parameters (QoC). Chapter 2.3 gives a detailed summary of the work published in the field of QoC.

In short, Gray and Salber name six parameters to rate the quality of a context information obtained from a sensor, that are the **coverage** of the sensor, the **resolution** to denote the smallest perceivable element, the construction-related **accuracy** of the sensor, the **repeatability** to state how stable the measurements are, the **frequency** of the sensor updates and the **timeliness**, which is how old a measurement is [44].

Buchholz et al. later refined this definition and identified five QoC parameters, that are the **precision**, the **resolution**, the **probability of correctness**, the **trustworthiness** and the **up-to-dateness**, which is how old a measurement is [45].

Additionally, Sheikh et al. distinguish between temporal and spatial resolution [47]. Temporal resolution is the same as frequency and spatial resolution is the same as in the definition of Bucholz et al. [45]. In [50, 44, 48, 49, 46] similar definitions are given.

None of those definitions considers the degree of time synchronization as a parameter of QoC. The only time related parameters are expressing how old a context date is, how fast a sensor delivers new data, and the delay between the real occurrence and the recognition by computers [78]. Furthermore QoC parameters, so far, are only defined for one sensor. In ubiquitous computing, where data often is derived by fusion of data from several sensor nodes, it should also be possible to rate the aggregated data's QoC.

To enable the consideration of time synchronization, we propose a new parameter for QoC: *time locality.* The less the time difference between two samples from different sensors is, the better is the *time locality*. Better *time locality* means it is less likely that the physical fact changed between the two measurements. To achieve good time locality, time series have to be in sync. In the previous chapters we analyze the problems caused by insufficient time locality, show the effect on reasoning accuracy experimentally, and test two possible solutions to achieve high time locality. *Time locality* is the first identified parameter related to fused data.

Definition 11. *Two sensor value's timestamps from sensor S_1 and S_2 are exactly matching when their time stamps t_{s1} and t_{s2} only differ by exactly Δt or less. The time locality parameter τ expresses a degree of matching for timestamps. τ is 1 if two sensor value's timestamps match exactly and decreases with bigger time differences (Figure 7.1). The decreasing slope*

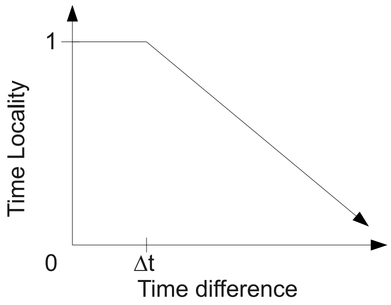


Figure 7.1: Time locality τ is 1 as long as the time difference in less or equal Δt. Afterwards it decreases until it strives to zero.

can be defined by the application depending on the maximum tolerated time difference.

There are two characteristics of sensors where *time locality* is of particular importance when the data of more than one sensor is aggregated:

Often a physical fact is measured by several redundant sensors to improve the accuracy. If those sensors' frequency differs, some may take more time to recognize a change of the physical fact. E.g. if one infrared sensor and one sonar sensor are used to detect approaching persons, the sonar sensor will detect the person before the infrared sensor can (the sonar has greater range). Though, the measurement time stamps have to be compared to clearly decide if a person is approaching or departing. If *time locality* is not considered for redundant sensors the reasoning may lead to ambiguous results.

Also, whenever it is important to consider the ordering of events, e.g. distinguish between alternating movement and simultaneous movement, it is important to consider the *time locality*. With lower value of *time locality* τ, the probability of incorrect ordering increases. For example, when the two accelerometers in the experiment in Chapter 6.3 are used to recognize if a person is sitting by considering the angles, sitting can be confused with walking if it is not clear if both legs are angled alternating or simultaneously; however, if $\tau = 1$ it can be clearly distinguished (see also Figure 4.1).

Chapter 8

Conclusion and Outlook

In the previous chapters we have discussed the issues related to time synchronization in context aware applications, which are an important part of the vision of ubiquitous computing. The main contributions of this thesis are:

- an analysis of known time synchronization approaches and their applicability in context aware applications,
- an analysis and evaluation of the reasons that cause time differences between devices, an evaluation of common reasoning algorithms for their susceptibility to time synchronization related decreases of reasoning accuracy,
- a conceptual approach to cope with the time synchronization issues along with a new architecture for context awareness,
- and a proposal to incorporate time synchronisation information into Quality of Context.

In the following chapters we summarize the contributions of this thesis. Chapter 8.1 details the contributions and results of the research conducted for and described in this thesis.

Throughout the chapters Chapter 8.1.1 to Chapter 8.1.4 every major contribution is summarized. In Chapter 8.2 open issues and suggestions how to build on the results of this thesis are discussed.

8.1 Conclusion

Even if we still do not see large scale real world deployments of context aware applications, such applications and architectures to support them are researched for more the 10 years now. However, none of the published approaches considers the role of proper time synchronization between devices gathering context data.

This lack of consideration may be due to one of the reasons, unawareness of the issues, a believe that the issues can be easily solved, or a viewpoint that these issues are inherent to context aware applications and must not be solved. The intention of this thesis is to raise the awareness of this issues and to show that they can be solved, even if known synchronization approaches are not applicable at all.

In the previous chapters we have presented the result of our research. These results show that time synchronization is a serious issue for context aware applications and that this cannot be solved with known synchronization algorithms but through a change of communication models in context aware architectures. These results are summarized in the following.

8.1.1 Applicability of known synchronization approaches

In Chapter 2 known time synchronization protocols are described. All protocols, whether it is NTP or synchronization protocols specially developed for sensor networks, are based on techniques to estimate the non-deterministic parts of the time it takes to send a time stamp from one device to another for comparison. Thus, the device clocks can be adjusted to a reference source using the time received from that source corrected with the estimate send time.

Beside some minor obstacles, in ubiquitous environments there often is no means of clock adjustment available. For example, smart phones, which are commonly regarded as the first true ubiquitous devices and therefore are object of extensive research as devices to enable context aware applications, often offer no API that allows for clock adjustment. Also, even more important, smart phones are personal user devices, as many devices in ubiquitous environments will be. Users may not want their devices' clocks to be adjusted whenever they enter a physical space that makes use of their devices, by whatever external entity.

8.1.2 Reasons for timing issues

In Chapter 3 the reasons for timing issues are discussed. There are four major reasons for timing issues in context aware applications. The first reason is that the clocks are often not synchronized with the desired accuracy. Sensors are often associated with low cost components that may cause device quality related and other problems, e.g. unstable oscillators. As argued in Chapter 2 known synchronization approaches may not be applicable. The experiments described in Chapter 3 show that clock drift of clocks in modern smart phones is in the order of seconds per hour. Also the offset observed in tested phones is 2 seconds in the best case.

The second reason for timing issues is that sensors have different construction-related physical capabilities. In dynamic environments where several sources of sensor data may be dynamically discovered and aggregated, it is not safe to assume that all sensors have the same physical capabilities, even if they are of the same general type. Furthermore, sensors of different types clearly have different physical capabilities and different measurement characteristics like different sampling rates. Thus, some sensors may take more time than others to sense a change in the physical environment.

Third, the sensor nodes have additional tasks to fulfill. For example, smart phones are often equipped with accelerometer sensors and light sensors, however, they clearly have additional processor load. Normally there is no way to influence the scheduling algorithms on the devices to ensure measurements at a fixed frequency. Therefore, the maximum sampling frequency is restricted by the additional processing load. Furthermore, it may not be possible to ensure sampling at fixed and stable sampling frequency.

At last, the common understanding of context reasoning foresees the pre-processing of sensor data in several steps and the inference of high level contexts out of the raw data [15, 22, 23, 24, 25]. These processing steps all consume processor capacity and in distributed scenarios are foreseen to be carried out on different devices with respect to the device capabilities, which consumes time while sending data over the network. Therefore, the time it takes before sensor data reaches the application is not only determined by the sensor itself, but also by the time it takes to process the data. This time is the sum of the time consumed by the processing steps and the network delays and can vary from value to value and sensor to sensor. This also restricts the maximum sampling frequency and hinders a stable sampling frequency.

8.1.3 Impact of time synchronization on reasoning accuracy

In Chapter 4 the impact of time synchronization on context reasoning accuracy is analyzed. The evaluation shows that for both, manually defined reasoning rules and classification learning techniques, time synchronization has a significant impact on the reasoning accuracy.

Six commonly used classification learning algorithms, namely Bayesian network (BN), K-nearest neighbor (IBk), decision tree (J48), rule-based classifier (JRip), naïve Bayes (NB), and sequential minimal optimization (SMO), are tested for their dependency on time synchronization. This is done in a scenario where two acceleration sensing devices placed at a user's wrists are used to recognize different user activities, such as typing on keyboard, writing, using mouse, and using phone.

For all algorithms the evaluation with different time offsets between the two sensing devices, introduced after the training phase, showed significant decreases of reasoning accuracy. The evaluated time offsets are chosen based on the findings from the previous evaluation of typical offsets of smart phone clocks.

Additionally, the impact of different sampling frequencies in situations where applications demand that only context data from the same specified time frame is aggregated is studied. It shows that even small differences in sampling frequencies, like those that are found to possibly occur due to the reasons described in Chapter 3, lead to significant number of sample pairs that do not fit in the required time frame and are thus unusable in the worst case.

8.1.4 Synchronization approach and new Quality of Context parameter

In Chapter 6 a novel approach to enable synchronized processing of context along with a new architecture based on this concept is proposed. This new architecture enables the synchronized processing of context data, without the need for time stamps originating from the sensing devices.

While architectures for context aware applications commonly use publish/subscribe based communication models (see Chapter 2.2), the approach proposed in this thesis shifts the communication model to a polling based model. This, along with an organization of context processing steps in a directed acyclic graph, allows to control the overall sampling frequency and is independent of devices' time stamps.

The proposed approach is evaluated and compared with a second approach that depends on device originating time stamps, but can be used in cases where a polling based communication model is not feasible. Both approaches ensure that only data with time stamps from defined time intervals are aggregated. The second approach, however, cannot control the sampling frequencies and thus suffers from the problem described in Chapter 3 that a significant numbers of samples may be useless. For example, we showed theoretically and experimentally that in the case of a maximum allowed time difference of $\frac{1}{10}$ of the sampling rate, only $\approx 20\%$ of the samples have fitting timestamps in the worst case.

The polling based approach, because it allows to control the sampling frequencies, can reduce the energy consumption of context gathering. In an experiment we show that the approach demands 2/3 less energy per sample pair fitting in the specified time interval, compared to the second approach.

8.2 Outlook

Although this thesis shows the importance of time synchronization for context reasoning applications and proposes a solution to solve these issues, there still remain interesting research questions that are out of scope for this thesis, but which we would like to see tackled in the future. To complete this thesis, here we will summarize the most important of these questions.

First, there is a special characteristic of sensor networks that could be used in case devices allow to adjust the clocks. Normally, time synchronization is based on the estimation of the time it takes to transfer a reference time to the device to be adjusted. This estimation is basically done using averaged time from one or several rounds of sending time stamps back and forth. In sensor networks, there are external reference points that can be used for offset calculation: the sensor events. For example, if all involved devices have a light sensor and are in the same room, when the light is switched on, it will be possible to compare the time stamps the devices attach to this event and calculate the offset between the device clocks. Because there is no estimation of network sending times from round trip times involved that should allow for high synchronization accuracy.

Another interesting research topic is, in situations where the time synchronization is not possible, how the reasoning algorithms can be extended to be less prone to timing issues. Furthermore, the experiments shown that some algorithms are less prone than other algorithms but offer worse overall performance (see Chapter 4), it will be interesting to further investigate if this algorithms can be improved to offer better performance.

At last, we propose the use of time synchronization information as part of Quality of Context (see Chapter 7). As for all Quality of Context parameters, it is still unclear how applications should react to the information that time synchronization is not sufficient. Possibilities include switching to another sensor source, estimate the value from context history, switch the used algorithm, trigger a synchronization process if possible, and simply cancel the context calculation.

Bibliography

[1] M. Weiser, “The computer for the 21st century,” *Scientific American*, vol. 9, pp. 66–75, 1991.

[2] A. K. Dey, *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, Georgia Institute of Technology, 2000.

[3] P. Dourish, “What we talk about when we talk about context,” in *Personal and Ubiquitous Computing*, vol. 8, 2004.

[4] B. N. Schilit and M. M. Theimer, “Disseminating active map information to mobile hosts,” in *IEEE Network*, vol. 5, pp. 22–32, 1994.

[5] B. N. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.

[6] A. K. Dey, “Understanding and using context,” *Personal Ubiquitous Comput.*, vol. 5, no. 1, pp. 4–7, 2001.

[7] J. Mäntyjärvi, *Sensor-based context recognition for mobile applications.* PhD thesis, VTT Technical Research Centre of Finland, 2003.

[8] K. Henricksen, *A Framework for Cotnext-Aware Pervasive Computing Applications.* PhD thesis, School of

Information Technology and Electrical Engineering at the University of Queensland, 2003.

[9] H. Lieberman and T. Selker, "Out of context: computer systems that adapt to, and learn from, context," *IBM Syst. J.*, vol. 39, pp. 617–632, 2000.

[10] S. Sigg, *Development of a novel context prediction algorithm and analysis of context prediction schemes.* PhD thesis, University of Kassel, Chair for Communication Technology, ComTec, 2008.

[11] J. Pascoe, "Adding generic contextual capabilities to wearable computers," in *Proceedings of the second International Symposium on Wearable Computers*, pp. 92–99, 1998.

[12] A. K. Dey, G. D. Abowd, and A. Wood, "Cyberdesk: A framework for providing self-integrating context-aware services," in *Knowledge-Based Systems*, vol. 11, pp. 3–13, 1998.

[13] G. Chen, *Solar: Building A Context Fusion Network for Pervasive Computing.* PhD thesis, Hanover, New Hampshire, 2004.

[14] G. Chen and D. Kotz, "Context aggregation and dissemination in ubiquitous computing systems," in *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pp. 105–114, IEEE Computer Society, 2002.

[15] B. N. Klein and K. David, "Basic approach of timing in context aware architectures verified by concrete advantages," in *Proceedings of the 2010 10th IEEE/IPSJ International Symposium on Applications and the Internet*,

SAINT '10, (Washington, DC, USA), pp. 113–116, IEEE Computer Society, 2010.

[16] B. R. Hamilton, X. Ma, Q. Zhao, and J. Xu, "Aces: adaptive clock estimation and synchronization using kalman filtering," in *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pp. 152–162, ACM, 2008.

[17] S. Yoon, C. Veerarittiphan, and M. L. Sichitiu, "Tiny-sync: Tight time synchronization for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 3, no. 2, 2007.

[18] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, 2002.

[19] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Continuous clock synchronization in wireless real-time applications," in *SRDS '00: Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, pp. 125–132, IEEE Computer Society, 2000.

[20] W. Su and I. F. Akyildiz, "Time-diffusion synchronization protocol for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 384–397, 2005.

[21] S. PalChaudhuri, A. K. Saha, and D. B. Johnson, "Adaptive clock synchronization in sensor networks," in *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*, pp. 340–348, ACM, 2004.

[22] G. Chen and D. Kotz, "Solar: A pervasive-computing infrastructure for context-aware mobile applications," tech. rep., Dartmouth College, 2002.

[23] K. Henricksen and J. Indulska, “A software engineering framework for context-aware pervasive computing,” in *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom’04)*, pp. 77–86, IEEE Computer Society, 2004.

[24] B. N. Schilit, *A system architecture for context-aware mobile computing.* PhD thesis, Columbia University, New York, 1995.

[25] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, “Reconfigurable context-sensitive middleware for pervasive computing,” *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 33–40, 2002.

[26] M. Berchtold, C. Decker, T. Riedel, T. Zimmer, and M. Beigl, “Using a context quality measure for improving smart appliances,” in *ICDCSW ’07: Proceedings of the 27th International Conference on Distributed Computing Systems Workshops*, p. 52, IEEE Computer Society, 2007.

[27] D. Mills, “Modelling and analysis of computer network clocks,” tech. rep., Electrical Engineering Department University of Delaware, 1992.

[28] Intel Corp., *Intel ICH Family Real Time Clock (RTC) Accuracy and Considerations under Test Conditions - Application Note – AP-728*, 2007.

[29] R. Love, A. Oram, and J. Read, *Linux System Programming.* O’Reilly Media, 2007.

[30] D. Mills, “On the accuracy and stability of clocks synchronized by the network time protocol in the internet system,” *ACM Computer Communication Review*, vol. 20, pp. 65–75, 1990.

[31] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 39–49, ACM, 2004.

[32] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 138–149, ACM, 2003.

[33] L. Chung and J. do Prado Leite, "On non-functional requirements in software engineering," in *Conceptual Modeling: Foundations and Applications* (A. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, eds.), vol. 5600 of *Lecture Notes in Computer Science*, pp. 363–379, Springer-Verlag, 2009.

[34] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: aiding the development of context-enabled applications," in *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pp. 434–441, ACM, 1999.

[35] L. Capra, W. Emmerich, and C. Mascolo, "Carisma: Context-aware reflective middleware system for mobile applications," *IEEE Transactions on Software Engineering*, vol. 29, pp. 929–945, 2003.

[36] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, and M. D. Mickunas, "Middlewhere: a middleware for location awareness in ubiquitous computing applications," in *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, Middleware '04, pp. 397–416, Springer-Verlag, 2004.

[37] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A middleware infrastruc-

ture for active spaces," *IEEE Pervasive Computing*, vol. 1, pp. 74–83, 2002.

[38] G. Biegel and V. Cahill, "A framework for developing mobile, context-aware applications," in *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, pp. 361–365, IEEE Computer Society, 2004.

[39] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger, "Context-awareness on mobile devices - the hydrogen approach," *Hawaii International Conference on System Sciences*, vol. 9, pp. 10–11, 2003.

[40] P. Fahy and S. Clarke, "Cass – a middleware for mobile context-aware applications," in *Workshop on Context Awareness, MobiSys*, pp. 304–308, 2004.

[41] T. Gu, H. K. Pung, and D. Q. Zhang, "A middleware for building context-aware mobile services," in *In Proceedings of IEEE Vehicular Technology Conference (VTC)*, pp. 2656–2660, 2004.

[42] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E.-J. Malm, "Managing context information in mobile devices," *IEEE Pervasive Computing*, vol. 2, pp. 42–51, 2003.

[43] O. Davidyuk, J. Riekki, V.-M. Rautio, and J. Sun, "Context-aware middleware for mobile multimedia applications," in *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pp. 213–220, ACM, 2004.

[44] P. D. Gray and D. Salber, "Modelling and using sensed context information in the design of interactive applica-

tions," in *EHCI '01: Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, pp. 317–336, Springer-Verlag, 2001.

[45] T. Buchholz, A. Küpper, and M. Schiffers, "Quality of context: What it is and why we need it," in *Proceedings of the Workshop of the HP OpenView University Association 2003 (HPOVUA 2003)*, 2003.

[46] T. Zimmer, "Qoc: Quality of context - improving the performance of context-aware applications," in *Advances in Pervaisive Computing. Adjunct Proceedings of Pervasive 2006.*, APC, 2006.

[47] K. Sheikh, M. Wegdam, and M. van Sinderen, "Middleware support for quality of context in pervasive context-aware systems.," in *PerCom Workshops*, pp. 461–466, IEEE Computer Society, 2007.

[48] A. Manzoor, H.-L. Truong, and S. Dustdar, "On the evaluation of quality of context," in *EuroSSC '08: Proceedings of the 3rd European Conference on Smart Sensing and Context*, pp. 140–153, Springer-Verlag, 2008.

[49] Y. Kim and K. Lee, "A quality measurement method of context information in ubiquitous environments," in *ICHIT '06: Proceedings of the 2006 International Conference on Hybrid Information Technology*, pp. 576–581, IEEE Computer Society, 2006.

[50] M. Krause and I. Hochstatter, "Challenges in modelling and using quality of context (qoc)," in *Mobility Aware Technologies and Applications* (T. Magedanz, A. Karmouch, S. Pierre, and I. Venieris, eds.), vol. 3744 of *Lecture Notes in Computer Science*, pp. 324–333, Springer-Verlag, 2005.

[51] A. Madan, R. Caneel, and A. Pentland, “Groupmedia: Wearable devices to understand social context,” in *Second International Conference on Mobile Systems, Applications and Services (Mobisys) 2004 ACM/USENIX workshop on Context Awareness*, 2004.

[52] T. Starner, J. Weaver, and A. Pentland, “A wearable computer based american sign language recognizer,” in *ISWC ’97: Proceedings of the 1st IEEE International Symposium on Wearable Computers*, pp. 130–137, IEEE Computer Society, 1997.

[53] T. Westeyn, P. Presti, J. Johnson, and T. Starner, “A naive technique correcting time-series data for recognition applications,” in *Wearable Computers, 2009. ISWC ’09. International Symposium on*, pp. 159–160, 2009.

[54] N. Kern, B. Schiele, and A. Schmidt, “Recognizing context for annotating a live life recording,” *Personal Ubiquitous Comput.*, vol. 11, no. 4, pp. 251–263, 2007.

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

[56] J. Mantyjarvi, J. Himberg, and T. Seppanen, “Recognizing human motion with multiple acceleration sensors,” in *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, vol. 2, pp. 747–752, 2001.

[57] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, “Activity recognition from accelerometer data,” in *Proceedings of the 17th conference on Innovative applications of artificial intelligence*, vol. 3, pp. 1541–1546, AAAI Press, 2005.

[58] S. L. Lau and K. David, "Movement recognition using the accelerometer in smartphones," in *Future Network & Mobile Summit 2010*, pp. 1–9, 2010.

[59] S. L. Lau, I. König, K. David, B. Parandian, C. Carius-Düssel, and M. Schultz, "Supporting patient monitoring using activity recognition with a smartphone," in *The Seventh International Symposium on Wireless Communication Systems (ISWCS'10)*, pp. 810–814, 2010.

[60] Y. Cho, Y. Nam, Y.-J. Choi, and W.-D. Cho, "Smart-buckle: human activity recognition using a 3-axis accelerometer and a wearable camera," in *Proceedings of the 2nd International Workshop on Systems and Networking Support for Health Care and Assisted Living Environments*, pp. 1–3, ACM, 2008.

[61] J. Lester, B. Hannaford, and G. Borriello, ""”are you with me?" – using accelerometers to determine if two devices are carried by the same person," in *Pervasive Computing* (A. Ferscha and F. Mattern, eds.), vol. 3001 of *Lecture Notes in Computer Science*, pp. 33–50, Springer-Verlag, 2004.

[62] S.-I. Yang and S.-B. Cho, "Recognizing human activities from accelerometer and physiological sensors," in *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pp. 100–105, 2008.

[63] G. Bieber, A. Hoffmeyer, E. Gutzeit, C. Peter, and B. Urban, "Activity monitoring by fusion of optical and mechanical tracking technologies for user behavior analysis," in *PETRA '09: Proceedings of the 2nd International Conference on PErvsive Technologies Related to Assistive Environments*, pp. 1–6, ACM, 2009.

[64] 3GPP, “Network identity and time zone (nitz), 3gpp ts 22.042,” tech. rep., 3GPP, 2011.

[65] D. Piester, P. Hetzel, and A. Bauch, “Zeit- und normalfrequenzverbreitung mit dcf77,” *PTB-Mitteilungen*, vol. 144, pp. 345–368, 2004.

[66] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques.* San Francisco, CA: Morgan Kaufmann, 2. ed., 2005.

[67] L. Bao and S. S. Intille, “Activity recognition from user-annotated acceleration data,” *Pervasive 2004*, pp. 1–17, 2004.

[68] T. Budd, *Understanding Object-Oriented Programming Using Java.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1999.

[69] M. Riaz, E. Mendes, and E. Tempero, “A systematic review of software maintainability prediction and metrics,” in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM ’09, pp. 367–377, IEEE Computer Society, 2009.

[70] G. Pour, “Component-based software development approach: new opportunities and challenges,” in *Technology of Object-Oriented Languages*, vol. 26, pp. 376–383, 1998.

[71] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns.* Addison Wesley, 2001.

[72] E. Vassev and M. Hinchey, “The challenge of developing autonomic systems,” *Computer*, vol. 43, pp. 93–96, 2010.

[73] J. Krämer and B. Seeger, “Semantics and implementation of continuous sliding window queries over data streams,” *ACM Trans. Database Syst.*, vol. 34, no. 1, pp. 1–49, 2009.

[74] B. N. Klein, S. L. Lau, A. Pirali, T. Löffler, and K. David, "Dagr - dag based context reasoning: An architecture for context aware applications," in *Proceedings of the 2008 Eighth International Workshop on Applications and Services in Wireless Networks*, pp. 20–25, IEEE Computer Society, 2008.

[75] R. B. Smith, "Spotworld and the sun spot," in *IPSN*, pp. 565–566, 2007.

[76] B. N. Klein, S. Sigg, K. David, and M. Beigl, "Dag based context reasoning: Optimised dag creation," in *Proceedings of Workshop on Context-Systems Design, Evaluation and Optimisation. ARCS 2010 - Architecture of Computing Systems*, pp. 177–182, 2010.

[77] B. Wuest, O. Drögehorn, and K. David, "The fame2 platform concept: Moving platforms to the mobile.," in *International Conference on Internet Computing* (H. R. Arabnia and O. Droegehorn, eds.), p. 423..433, CSREA Press, 2004.

[78] Y. Bu, T. Gu, X. Tao, J. Li, S. Chen, and J. Lu, "Managing quality of context in pervasive computing," in *QSIC '06: Proceedings of the Sixth International Conference on Quality Software*, pp. 193–200, IEEE Computer Society, 2006.

In context-awareness, sensors in the vicinity of the user are utilized to infer a user's situation. There is no single sensor that is able to correctly and completely access a user's situation, context awareness is always dependent on a multitude of sensor information to be aggregated to at least estimate a user's situation. It cannot be safely assumed that the clocks of all these sensor devices are accurately synchronized.

In this book

- we discuss the issues related to time synchronization in context aware applications,
- analyze known time synchronization approaches and their applicability in context aware applications,
- analyze and evaluate the reasons that cause time differences between devices,
- evaluate common reasoning algorithms for their susceptibility to time synchronization related decreases of reasoning accuracy,
- propose a conceptual approach to cope with the time synchronization issues along with a new architecture for context awareness,
- and propose to incorporate time synchronization information into Quality of Context.

ISBN 978-3-86219-226-7