KTH Information and
Communication Technology

Sensing and Making Sense

Designing Middleware for Context Aware Computing

Martin Jonsson

A Dissertation submitted to
The Royal Institute of Technology
in partial fulfillment of the requirements for
the degree of Doctor of Technology

May 2007

The Royal Institute of Technology
School of Information and Communication Technology
Department of Computer and Systems Sciences

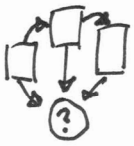
DSV Report series No. 07-009
ISBN 978-91-7178-702-6
ISSN 1101-8526
ISRN SU-KTH/DSV/R - 07/9 – SE

Abstract

Computing devices are becoming wireless, smaller and embedded into other artifacts. Some of them are mobile while others are built into the environment. The novel technologies are also becoming more dependent of communication with other computing devices over different kinds of networks. These interconnected devices constitute locally and globally distributed service environments that will enforce new requirements on the design of software systems. These new type of environments provide both opportunities for new types of applications as well as a number of new problems that will have to be addressed. One approach that have been suggested both in order to provide new functionalities and as a possible solution to some of the problems is to try and collect and incorporate aspects of the *context* of a person, activity or device as a part of the computer system.

This thesis addresses some issues that have to be considered when designing this kind of systems. In particular the thesis examines how to design middleware that can support the creation of context aware applications. As a part of this work several instances of such systems have been implemented and put in use and tested in various applications. Some key problems with respect to the design of context information middleware are also identified and examined. One question that is addressed concerns the clash between the need for internal representations of context information and the goal of middleware to support a broad range of applications. Another problem that has been addressed concerns how to create means for context aware service discovery in ubiquitous computing environments. Various mechanisms that address these problems have been implemented and tested. Finally the work addresses issues related to the role of the users in this kind of systems. Implementations and experiments have been performed where users take more active roles in aspects of system maintenance and adaptation as well as in the interpretation and representation of context information.

Acknowledgements



For supervision, always precise and clear:
Calle Jansson

For collaboration around the Context Shadow system: Patrik Werle, Johan Mattsson, Fredrik Kilander and Peter Lönnqvist.

For collaboration in the ACAS project:
Li Wei, Fredrik Kilander, Andreas Wennlund, Theo Kanter, Chip McGuire.

For collaboration on and extensions to the Spots system:
Tommy Westman, Li Wei, Johan Mattsson
+ all nice students who have been forced to use, evaluate and extend the Spots system as part of their course work

For reading and commenting: Ylva Fernaeus, Johan Mattsson, Tessy Cerato-Pargman, Markus Bylund, Patric Dahlqvist

For creative expansions of research into the realms of pyssel and voodoo: Ylva

For putting up with my continuous outbursts of rambling - my roommate and friend: Johan

For keeping the jolly DSV-spirit high:
Patric, Johanarna, Maria, Jakob, Lisa, Karin, Hillevi, Fatima, Klas, Tony, Wah Sui and others

For easing the rather turbulent landing of this thesis through refreshments and entertainment: Oliver, Hanna, Johan, Oskar, Lisen, Ludde and all other wonderful friends.

For being there all the time and always helping out:
My parents

Slutligen, min lilla familj

Hanna - Du är bäst! Det hade aldrig gått utan dig

Alba - Nu kan du få hoppa på min dator igen

Alfons - Nej, pappa ska inte bli en sån doktor som skär i barn

TACK!

Table of Contents

Chapter 1
Introduction

Our understandings of what a computer is and how it can be used are dissolving and being reshaped as a plethora of new computerized artefacts is appearing. Some are small and mobile like mobile phones, digital cameras and PDAs, and others might be invisibly integrated into our environments. An overall trend is that we are going from a situation where a single person interacts with a single computer to situations where each person owns and carries several computing devices, and also shares a number of devices with other people. A lot of research have been conducted in this realm, which is often referred to as ubiquitous or pervasive computing (Weiser 1991), both on technology development as well as to revisit theories on interaction with computers. An important aspect of the technological development is that it makes computer technology increasingly available. On one hand the *physical availability* increases as due to an inhabitation of computers in our everyday physical environments. On the other hand the *virtual availability* increases due to the fact that the computing resources are becoming accessible to a higher extent through various forms of computer networks.

One example of how the increasing virtual availability is manifested is the increasing use of *service oriented* software architectures that is gaining ground in these new distributed computing environments (Papazoglou et al. 2003, Espinoza 2002). The basic assumption in this approach is that computational and hardware functions can be encapsulated in terms of services. These services are then made available over some communication network, so that they can be combined with other services in order to achieve dynamically composable computer systems.

As new technologies make it possible to move interaction with computers beyond the desktop and into the larger real world where we live and act it becomes possible to bring the power of computation to new situations and activities and thus in new ways support the tasks at hand. This increasing inhabitation of technological devices in our everyday environments will thus not only alter the way we understand and interact with computers, but it will also require fundamental changes in the design of the software applications. So far the computing devices are still very much designed as standalone personal devices. In this sense they are not that well integrated with the physical world, nor do they easily adapt to situational and environmental changes. One solution to this is to try and provide the computer systems with clues regarding the *context* in which the interaction is taking place. By incorporating various forms of sensors into the computer systems it becomes possible to acquire information about for example properties of the physical environment where an activity is taking place. Providing information about the users' context to applications could be useful in several ways. The behaviour of an application can be adapted to make the interaction more efficient or to increase the ease of use. You can also imagine entirely new types of applications that are designed specifically to make use of some certain context information (Dey 2001). There are already a number of existing commercial systems that uses context information as input in different ways. The absolutely most common class of these systems are those that use information about the users´ geographical location as input, such as e.g. GPS based car navigation systems.

The combination of the service oriented computing and context aware computing domains have the potential to create computer systems that can dynamically rearrange and recompose themselves with respect to the changing environments. The context information can thus be seen as a way to bridge the physical world and the world of computational services. This thesis strives to explore some of the properties in the intersection between these areas, and more specifically how to provide support in the creation of applications within this domain.

Creating computer applications whose behaviour is affected by information from sensors is a complex task that includes a number of sub problems. The sensors might be decoupled from the device that runs the application, or you could have several applications sharing information from the same sensor. The application designer then has to consider issues like how to transport the data, how to represent it and how to infer some useful information from it. To simplify the process of creating these applications, much of the sub problems can be taken care of by a *middleware system,* providing different kinds of tools for the application developer, and also providing means to make applications more efficient, by e.g. shifting some of the computation load from clients to infrastructure. This work will address a number of design challenges that has to be faced when designing this kind of context information middleware. These design challenges have been addressed through the design and implementation of several versions of context information middleware systems.

In order to realize open service oriented systems in ubiquitous computing environments, one key issue that has to be addressed is to provide means for the services in the system to discover other services in an efficient and meaningful way. One approach to this problem that this work addresses is an attempt to develop

mechanisms that uses context information in the service discovery process. In this way services could e.g. find other services in a shared location.

Much of the research within the context aware computing domain deal with trying to make different kinds of assumptions based on some more or less limited sensor input. This sensemaking process generally includes mapping the sensor data against some model of concepts of the real world. In many cases the assumptions you want to make concerns aspects of human behaviour, for example to be able to adapt application behaviour to a specific user activity. There have however been several studies showing that there is a great complexity in the mechanisms people use to organize their everyday activities, e.g. (Salvador et al. 2003), making any modelling of human activities a very cumbersome task except for in very constrained settings. When trying to create generic support systems for context awareness a key issue then becomes to design formalization and interpretation processes of sensor information that balances the need for generality with the requirements from specific applications. This work addresses some issues related to how such interpretation processes can be designed to in spite of these problems still support a wide range of application scenarios.

Another issue that is examined in this thesis concerns the role of the user in context aware systems. Having systems that collect and uses data related to the context of a user raises issues concerning privacy and user control. How can you make sure that, on one hand, the user has control over what data is being shared and with whom? And on the other hand, how can the user have control over how the data can be expressed and interpreted? This work addresses different ways of ensuring user empowerment through an increased participation in both the setup and maintenance of these systems, as well as being involved in the processes of interpreting and formalizing context information.

1.1 Problems and objectives

This work examines some approaches to support the design of context aware systems for ubiquitous or pervasive computing environments. There is also a focus on supporting so called service oriented systems. More specifically the work addresses how these problems can be addressed through the design of *context information middleware* systems. This work points out three key problems that will be examined in further detail:

- **Context interpretation in context information middleware:** When encoding context information in a middleware system, you have to choose a representation format that will eventually restrict the range of possible application areas. How can you design mechanisms in a context information middleware that will maximize the range of applications that can be supported?

- **Achieving dynamic service discovery through context information middleware:** Existing technologies for service discovery in ubiquitous computing environments generally have too imprecise selection mechanisms. How can you design mechanisms where context information is used to improve the service discovery and selection process?
- **Engaging the user through design of context information middleware:** How can you empower of the users of context aware systems by incorporating them in both the setup and maintenance of these systems, as well as involving them in the processes of interpreting and formalizing context information?

1.2 Methodology

The work in this thesis is to a large part affected by the current discourse of human computer interaction (HCI), even though the actual contributions rely more on traditional software engineering methods. More specifically the contributions of the thesis fits well within the different research areas that can be described as ubiquitous or pervasive computing as well as the closely related area of context aware computing.

The research methods used in this thesis covers analytical methods such as studying and evaluating existing research and forming new hypotheses. Engineering methods such as creating prototypes as proof of concept has also been used. Finally a small amount of empirical evaluation through user studies has been used.

1.2.1 Problem formulation process

During the course of this work a strong focus has always been to try and provide useful tools that can be used to make it easier to create applications for ubiquitous computing environments. Early on one question emerged as an interesting problem to address, that of *how to discover a set of software services that might be relevant in a certain situation.* Therefore the problem of how to achieve context dependent service discovery is one of the key questions that is addressed in this work. Besides from being a well acknowledged research problem in the scientific community, the need for this functionality also emerged locally during the development of a series of software tools to support local collaboration. This problem has since then been addressed in several design solutions including implementations both on middleware and application level.

The second key problem in this work, i.e. how the choice of specific middleware data formalizations clashes with the aim that middleware systems should support a wide range of possible applications, have emerged gradually during the design work with various context information middleware systems. This problem first became obvious when an existing middleware system had to be adapted to fit a new application domain. When designing later versions of context information middleware systems this problem was more thoroughly addressed in the system design. Different design approaches to solve this problem were implemented in the different systems and could thus be compared and evaluated.

The final key problem concerns how user can be involved in context information middleware, was mainly formulated based on critique against the domain of context aware computing by e.g. (Dourish 2004; Greenberg 2001; Bellotti and Edwards 2001). This problem was explicitly targeted in the later work, but experiences from the earlier work and design choices could still be used as valuable input in the discussions.

1.2.2 Design process

The contributions in this thesis are mainly extracted from the work on three context information middleware systems, The Context Shadow System, the ACAS architecture and the Spots System. The research process can to a large extent be described in terms of how these system designs evolved in parallel with the key questions of this work, and how they were evaluated. The first system, *Context Shadow,* was conceived while working on various forms of support for co-located collaboration. The work relied heavily on service oriented and agent based systems and a need emerged to provide the agents in the system with information related to the physical context so that it for example could present information on a shared display. This problem was later generalised as to how to achieve context dependent service discovery. A prototype of the Context Shadow system was implemented and later incorporated in two larger systems that included user interface components and thereby means to perform user studies.

The work with the context shadow system raised a number of questions that seemed to be interesting to investigate further. Issues of how to represent context information in context information middleware seemed increasingly important. In the *ACAS architecture*, developed as a joint venture in a larger project, issues of context information representation and transport was addressed. Starting out from the experiences made with the first system, this design kept some properties of the first system while also exploring alternative approaches. The system design is manifested in a rather extensive system architecture specification, describing various aspects of the system design. The system has only been partially implemented, so that certain components have been evaluated independently.

While working on the ACAS architecture, a number of new insights and ideas started to appear, mainly concerning the role of the user in context aware systems, that to some extent was incompatible with the ongoing work on the ACAS architecture. This resulted in a new system design, the *Spots system,* which was developed around ideas about how users can be involved in the sensor interpretation process as well as in the maintenance and setup tasks in context aware systems. A prototype system was developed and has been refined in a number of versions. A large number of smaller applications have been developed based on the Spots system.

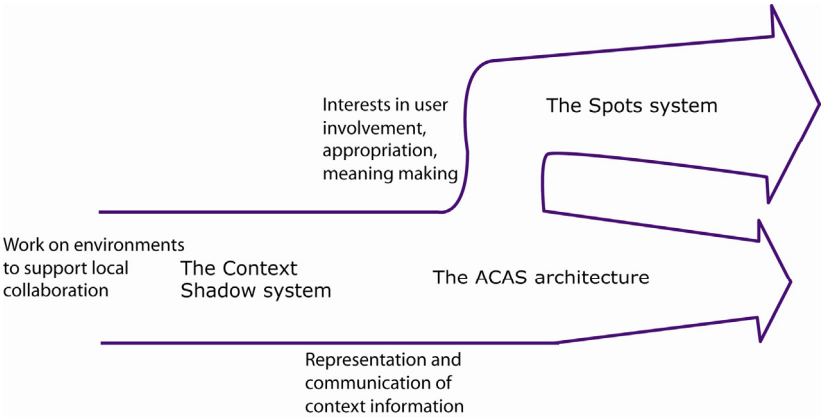


Figure 1. Overview of the system development process

1.2.3 Theoretical perspectives

Throughout the work with this thesis a shift in theoretical perspective has taken place. The early works presented in the thesis are based on a fairly *rationalistic* perspective especially with respect to how human activities should be understood. This position manifested itself for example in the usage of various decision making techniques, where human activities and preferences were modeled internally in the system.

Gradually, mainly through readings on *situated cognition* (Suchman 1987; Nardi 1996) and *phenomenology* (Merleau-Ponty 2002; Dourish 2001), new understandings were formed around aspects of human activity and perception. By gaining an understanding of the complexity and dynamicity of the contextual factors governing our everyday activities, a certain skepticism evolved against general models and conceptualizations describing human activities. This shift in perspective became clearly manifested in the work with the Spots system, in which the tasks of assigning meaning to sensor information to a large extent is handed over to the end users.

Due to this shift in perspective, some of the early work has had to be revisited and understood in new ways. In spite of these revisions certain inconsistencies with respect to how different components is described and evaluated can probably still be detected. In general the shift, which is still in progress, has not been an unproblematic process. Finding ways to combine theories from sociology and philosophy with conventional engineering methodologies is sometimes a somewhat complex process.

1.2.4 Evaluation

A large part of the work in this thesis concerns design considerations with respect to middleware systems. There are some specific characteristics of middleware designs that make them hard to evaluate using existing evaluation techniques. This problem has earlier been addressed by Edwards et al. (2002). One point they make is that middleware is mainly meant to act as enablers of functionality for applications built on top of the middleware.

> "Unlike middleware, *user-visible applications* - applications with which the user interacts directly have long traditions of user-centred design and evaluation. Techniques such as participatory design, ethnography, and

> others all play a part in deciding what features go into an application, how well those features address the needs of users, and ultimately the worth of the application itself. There is, however, no comparable set of user-cantered techniques for determining what features should go into a middleware system (the design of the middleware), nor for determining the success or failure of those features (the evaluation of the middleware)." (Edwards et al. 2002)

What they claim then is that the only proper way to evaluate how well the middleware performs is to build a number of applications on top of it. This approach however has some drawbacks. The most obvious drawback is that it is hard to distinguish whether you are evaluating properties of the test application or the underlying middleware. Edwards et al. suggests a different workarounds to this problem. For example you have to decide whether to implement rich applications for real settings or lightweight proof of concept applications used solely as tools for evaluation. If you choose to create applications that are to be used by real users for real tasks over a longer period of time these applications must then adhere to design aspects of usability and usefulness in order to provide useful data for evaluation. On the other hand you can choose to create numerous lightweight proof-of-concept applications that are specifically deigned to illustrate some specific features of the middleware. The process of creating these simple applications might reveal strengths and weaknesses of the middleware. Since these simple applications most likely will have shortcomings with respect to usability and lack several features, they are not so suitable for user studies in real usage settings. Smaller and more controlled user studies might however provide useful data with respect to the perceived usefulness of certain functionalities in the middleware.

In this work evaluations of the proposed solutions have been performed on two levels. On one hand they have been evaluated as complete and separate design solutions providing certain functionalities and on the other hand as instruments to examine some more specific design problems.

The complete systems have been evaluated mainly through implementations of light weight applications, in order to prove that the middleware can actually perform the functionalities it is supposed to. Two user studies have also been performed, where users have been using applications developed on top of the middleware in real work situations. As described above it was however hard to get results going beyond the properties of the test application in order to be able to say anything about the underlying middleware. The most valuable insights and results have been acquired by usage of the systems internally by researchers in the lab. This method with a continuous process of subjective evaluations in parallel with the evolution of the system has earlier been described in terms of a *Living Laboratory* (Schmidt 2002).

The other level of evaluations focuses on specific properties of the implemented systems rather than on the entire system as a whole. The different implementations should thus be seen more as instruments whereby these more specific questions were addressed and evaluated. The properties that were examined are to different extent and in different ways addressed in all of the different implementations. In this way different approaches to address the same problem have been compared and evaluated.

The methods used to evaluate the different proposed design solutions include identifying a number of criteria that the sought solution should encompass. Each proposed solution have then been analysed analytically and empirically against these criteria.

1.3 Thesis contributions

The contents of this work can be divided into a number of contributions of different kinds. The first contribution is a framework called Ubiquitous Service Environments in which service oriented computer systems can be described in terms of *environments* including both software and hardware components as well as aspects of the physical context affecting and constraining the system.

The second category of contributions consists of designs and implementations of three different *Context Information Middleware Systems,* which are tools to support the creation of context aware applications:

- Context Shadow: A fully implemented support system mainly targeting the problem of how to achieve context dependent service discovery
- The ACAS architecture: A partially implemented architecture for context information management. This system has been jointly developed as a part of a larger project.
- The Spots system: A fully implemented user managed support system for sharing location dependent information within communities.

The third category of contributions consists of specific design solutions that address the three key problems for context information middleware introduced above. These contributions can be described in terms of mechanisms:

- to enable abstraction and interpretation of sensor data in the middleware layer without constraining the range of possible applications.
- to support context dependent service discovery.
- to engage the user in context information middleware systems.

1.4 Division of labor

The Context Shadow system was entirely designed and implemented by me. The applications with which the spots system was used and evaluated was mainly jointly constructed by me, Fredrik Kilander and Patrik Werle.

The ACAS architecture was formulated within the ACAS project. The overall architecture was jointly formulated by me, Fredrik Kilander and Li Wei. Within this project my focus has been on representation and interpretation of sensor information, location detection using Mica Motes and service description and discovery. These are also the components from this project that is covered in some detail in this thesis.

Finally the Spots system was initially designed and implemented entirely by me. A reimplementation was then performed as part of a Masters Thesis work, by Tommy Westman. The system have since then been further developed by me. Some joint efforts have been made to combine the Spots system with the works of Li Wei, in order to achieve the location sensitive messenger application. In a similar fashion a joint effort was made together with Johan Mattsson to achieve the Interactive room configuration application.

1.5 Thesis overview

Chapter one in this thesis gives a brief introduction as well as presents the research questions, methodology and contributions. In *chapter two* the context in which the work should be understood is presented, in terms of the relation to relevant research domains. This chapter also introduces the Ubiquitous Service Environments framework, which is a way to describe the kind of computing environments that the rest of the work is targeted for.

Chapter three consists of an investigation of the problems regarding provision and acquisition of context information, both in general and with respect to three specific questions: 1) How to support generality in context interpretation. 2) How to support context aware service discovery and 3) How to engage users in context information middleware systems.

The *fourth chapter* describes the three separate context information middleware designs with implementations and example applications. *Chapter five* discusses the problem of how to achieve generality in context information middleware, and how this has been achieved in the different systems. In *chapter six* the problem of context aware service discovery is examined, starting off with a comparison of different service discovery techniques, and then a discussion regarding how context information could be used in this process, and more specifically within the USE-framework. Different approaches to this problem is presented and evaluated. *Chapter seven* discusses the problem of how users to a larger degree can be engaged in the setup and maintenance of context aware systems.

Table 1. An overview of the three examined key questions and to what extent they have been examined in the different system designs

	CIM design I: The Context Shadow system	**CIM design II: The ACAS architecture**	**CIM design III: The Spots system**
Generality in context interpretation	Addressed to some extent	Addressed	Addressed
Context aware service discovery	Addressed	Addressed	Addressed to some extent
Engaging users in context information middleware	Touched upon	Touched upon	Addressed

Chapter 2
Background

2.1 Ubiquitous computing

Over the latest years a strong new trend has been seen, moving the interaction with computers away from the desktop and the desktop PC. New technologies such as micro controllers and wireless technologies have opened doors for entirely new types of computing devices. Small laptops and notebooks make it possible for us to use computers while away from office whereas cell phones and other wireless technologies make it possible to be connected to the Internet from anywhere. Highly specialized computing devices such as digital cameras, audio players, digital books etc. have also started to appear, blurring the distinction between computers and other electronic appliances. The promising technical advancements have inspired several new research fields that challenge our existing view of computers and how they are used by envisioning entirely new ways of understanding and interacting with computers.

One early vision of this kind was that of "ubiquitous computing", presented 1991 by Mark Weiser (Weiser 1991). According to this vision, huge numbers of computers will occupy our offices and everyday environments, in a way that information and computing power is always available in our periphery, and can easily be put into focus when needed. His vision has over the latest few years been revitalized and adopted by a large number of researchers from different domains. From the interest in this area combined with recent technical advancements a number of novel research areas have appeared.

Mobile and wearable computing: This research area focuses on support for performing work when being on the move, including the design of input and output interaction techniques, support for wireless connectivity etc.

Disappearing computers: As computer technology becomes smaller and cheaper, it becomes possible to equip objects in our everyday environments with computing power, which could either enforce the original function of the object or provide entirely new functionalities.

Tangible interfaces: People have developed sophisticated skills for sensing and manipulating our physical environments. However, most of these skills are not employed by traditional graphical user interfaces. As computing moves away from the desktop and into the world around us, objects in the real world have the potential to become the interface into our digital world. In the tangible interface research area physical artefacts are created that could both influence and/or reflect the digital world.

Interactive environments: Most of today's computing environments are designed to support the interaction between one person and one computer. Different attempts have been made to create alternative environments, carefully designed to support groups of people working on a multitude of devices simultaneously. This research area deals with questions like how to support uniform interaction for several people using several devices as well as questions regarding information flow etc.

2.2 Service oriented computing

In a Service-Oriented Architecture, loosely coupled pieces of application functionality are published, consumed, and combined with other applications over a network. Each such component can be modelled as a service performing some task over the network.

The most obvious development towards a service oriented view is the upcoming notion of Web services on the Internet, a technology that is mainly targeted to support different aspects of e-commerce. There are already a growing number of services available on the web that does many interesting things: keep track of your contacts, edit and manipulate photographs, etc. The web in the present form however underlines the presentation of data in a human readable format (with all the necessary graphical and aesthetical issues). Web services, on the other hand, aims at presenting the information in a machine readable format, thus enabling retrieval, access, composition and, in general, automatic interaction of systems.

The service oriented view has also been adopted within the post desktop computing domain, where e.g. the integration of several devices in an interactive environment can be simplified by decomposing the functionalities of the devices into a number of services. In this domain, research issues could e.g. be to provide means for a mobile user to utilize computing resources like shared displays etc. in new and unknown environments. Suns Jini technology (Waldo 1999) and the UPnP technology from Microsoft (Microsoft Corporation 2000) are examples of commonly used support for creating service oriented architectures for the post desktop domain. Some interesting issues within the service oriented computing domain concern:

- Service discovery: How do the services present themselves so that they can be found by other services?
- Service interoperability: How do the different services communicate? What protocols should be used?
- Service composition: How do you compose two separate services to a more complex service that could be presented to a user?

2.3 Ubiquitous Service Environments

The way that people are using computer is a process in constant change. With the development of the personal computer as a multi function tool supporting a constantly widening range of tasks, working on a task with a computer very often includes composing and switching between *a set of applications*. The internet revolution added the possibility to incorporate additional *remote* resources and information to this working set. Finally the introduction of an increasing number of mobile or pervasive computing devices enables situations where the set of supporting technologies are spread out over several devices. Consider for example a scenario where a person hears a new song on the radio on his mobile phone, and then decides to buy the song from an internet mp3 store by using a browser on the PC. When the song is downloaded it is transferred to the person's iPod, which is then hooked up to the home media center to play up the song.

To be able to understand, analyze and design this kind of situations we believe that the existing notions of applications computers and devices are inappropriate. In this chapter we will therefore introduce the concept of *Ubiquitous Service Environments* or *USEs*. The USE notion is a framework by which you can describe a computer system in terms of a specific assembly of services and applications that are used at one instance in time to support a specific task. A key property of the USE concept is that it includes contextual information such as the physical boundaries of the system and who the users are etc.

2.3.1 Moving towards a service oriented perspective

So far the manufacturers of pervasive computing devices (PDAs, projectors, mobile phones, etc.) have chosen to design their devices very much as atomic entities to be used mainly by one single user, an approach very similar to the design of standard PC's. With this approach software systems can be created similarly to the way systems are being created for normal PC-based technology focusing mainly on applications designed to be used in a standalone fashion. But since the devices are becoming increasingly equipped with various communication technologies like GPRS, WiFi and Bluetooth, the application design is slowly moving towards a more service oriented perspective, where the different applications to a higher extent are communicating with other entities. An example of this is the evolution of calendar applications on PDAs and mobile phones, where the synchronization process with the calendar on the

user's PC has shifted from being a manual process involving cables to an automated and invisible process using wireless connections. Taking the service oriented perspective a step further, each device can become a node in a larger system; an approach that would make it possible to create much more flexible and usable systems, since each part could be used in numerous ways. Sharing devices in this way also opens up for the creation of new tools for collaboration, where the tools can be designed to support groups working together using numerous devices. To fully embrace the service oriented approach will require that both the hardware and software in some ways has to be designed to be open for communication with other software entities. On the hardware side some sort of network connection might be sufficient, a property already present in most novel computing gadgets. If the devices should be contactable from other system parts they might also have to be "turned on" to a higher extent than today, putting new requirements on power consumption, battery life etc.

While the requirements on the hardware design to achieve this vision are rather modest, the changes in the software design have to be much more extensive. Going from standalone applications towards a more service oriented perspective will impose a number of new problems that has to be dealt with, such as:

Interoperability: how could the different components interoperate given that they might be written in different programming languages and maybe does not have exact knowledge regarding how to use each other?

Service discovery: How do the different components find each other? How can you weed out services that are not relevant to the current task?

Security: Making software components that are open for usage/contact by other services over the network also opens up for malicious services that might damage the system. The openness might also raise integrity issues.

Usability: How do you create usable systems, when an application might be spread over different devices, and consist of components from different vendors?

Stability: A distributed software system, where components can appear and disappear at any time, requires design efforts to remain stable.

Business model: Who pays whom for what?

Maintenance: How do you maintain or upgrade applications consisting of parts from different vendors.

2.3.2 Describing the new computing environments

In order to examine some of these problems, as well as the benefits with these kinds of systems, we have come up with a framework that we call Ubiquitous Service Environments or USEs. This approach provides means to model and design the new computing environments that no longer consist of standalone computers running standalone applications, but rather consists of a plethora of independent services spread out over numerous devices.

A crucial idea within the USE approach is that the design of a computer system also includes the physical environment in which the system is deployed, for example to

determine borders to delimit the system. Acknowledging that the physical environment is important also creates an incentive to try to incorporate knowledge of the physical world into the software system. An instance of a USE should be understood as a snapshot of an ongoing activity describing the different entities that are relevant for the system. Each such USE model thus contains a number of elements as outlined in Table 2.

Table 2: The different elements constituting a USE

Activity	The task performed by one or several persons that determines the borders of the USE.
Services	Software entities performing different kinds of computational tasks.
Perceived tools	A service or a composition of services providing distinguishable affordances (for example through a GUI or a TUI) for persons to support an activity.
Devices	Physically standalone containers of hardware resources, user interface elements and services.
Persons	One or more individuals involved in the activity.
Physical spaces	The physical environment containing an activity. Includes devices, furniture, walls etc.
Logical spaces	The subset of services and information being used in an activity.
Information	Digital material such as documents, images etc. that is manipulated and transformed by persons or services.

One key idea with the USE approach is that a specific USE is supposed to support a certain activity. This activity can be an individual task but also activities where several people are working together. We believe that a model that includes several people as well as several devices will make it easier to develop tools that support collaborative work. The activity centered view is to a large part inspired from Activity theory (Nardi 1996) where human actions can be described in terms of activities, and where the activity consists of subjects using artefacts to transform some object. Activity theory has been used successfully to analyse the usage of computer systems as a tool in the design process. These existing analyses have often focussed on the artefact as being some software application that should be designed to solve some problem. In the USE philosophy the artefact may consist of an entire environment, including both software as well as physical "real world" components, where the latter is often being neglected in software design.

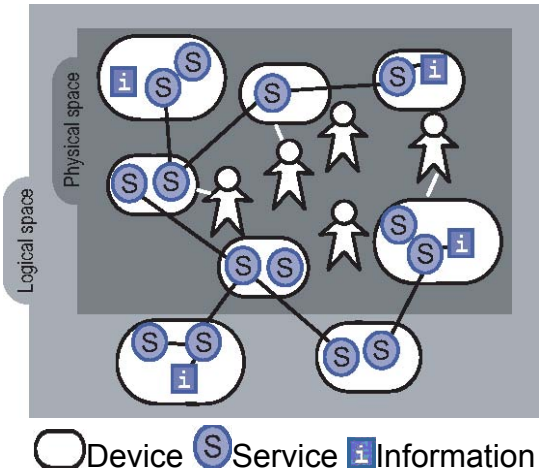


Figure 2. A snapshot of a USE supporting a specific activity

A USE differentiates between the physical space and the logical space containing the activity. The physical space concerns the physical environment containing the supported activity. It includes properties and restrictions of non-computational entities such as walls and furniture as well as properties and restrictions of the computing devices within the space. The logical space concerns the software system and the relation between the different service components. An important function of the notions of physical and logical spaces is to enable exclusion of entities that are not relevant to the activity in focus, thereby describing the boundaries of the USE. In a pervasive computing scenario containing a broad usage of open service environments, each computing device could contain a great number of services. Each specific activity would use only a subset of these services, whereas one service may be shared between different activities. So when talking about something like today's applications in such an environment it is rather a specific subset of services and devices.

Instead of talking about users of the system the notion of persons is used. We believe that the notion of users is misleading since the main task of the persons in question is not mainly to use the system but to actively participate in the current activity. Another entity of this model is the information entity. Information resides on devices and can be transformed and utilized by persons and services. These pieces of information might be files and documents that is being transformed, consumed or passed around within the scope of the activity

The computing devices in a USE acts as bridges between the physical and logical spaces since an instance of a service will always reside on a physical device. The devices might have specific characteristics and functionalities that affect the kinds of services they can provide, like for example user interface properties regarding audio or display functions. The devices can be both stationary and part of the local environment or they can be mobile devices owned by some person and only temporarily inhabiting the space.

The basic building blocks in a USE are however the actual services. This notion embraces a number of different types of software components with rather different properties. The common broad definition of a service in a USE is here defined as:

"A service is a software component that performs a task for a user directly or another service in the USE".

Even though any software application can be characterized as a service using the USE framework, the framework is mainly targeting systems that in some sense encompass a service oriented architecture. For such services, specifically designed to interoperate with other services, a central property is that of accessibility. A service should be accessible by as many other services as possible over the network, or it should be easily accessible for a user. The interconnection of services is a key feature in order to enable automatic composition of services to support specific tasks.

Since the web of interconnected services constituting the USE might be partially invisible to the users, tool abstraction is introduced, consisting of the actual distinguishable affordances for users to support the targeted activity. Thus a tool is typically a service or a composition of services that provide some kind of user interface like a GUI or a physical interface like a remote control.

2.4 Context aware computing

One aspect of the post desktop computing area described above is that it tries to bring computation closer to the real world, our daily lives and every day environments, mainly by immersing the technology in the environment in different ways. Another way of bridging the gap between the digital and physical world is to provide information about the real world to software applications using different kinds of sensor information, an approach that can be described as "context aware computing"

Within the context aware computing research community the notion of context is commonly understood as mainly physical (and to a smaller extent, social) aspects of the situation in which an interaction with a computational device is embedded. One goal of context-aware computing is to acquire and utilize information about the context of a computer usage situation so that the behaviour of the device can be adjusted to the particular people, place, time, events, etc. For example, a cell phone could always vibrate and never beep in a concert, if the system can know the location of the cell phone and the concert schedule. However, this is more than simply a question of gathering more and more contextual information about complex situations. More information is not necessarily more helpful. Context information is useful only when it can be usefully interpreted, and it must be treated with sensitivity. Research in this area is performed on different levels (Moran and Dourish 2001):

- Collecting data from sensors
- Distributing data to applications
- Defining context models
- Refining data
- Creating context aware applications

Some key topics targeted in this thesis concerns how you can support the process of acquiring context information from sensors as well as how you can provide this information to applications. In order to address these problems one should first analyze the concept of context and how it could be understood and used in relation to computer usage and applications.

2.4.1 Using context in applications

Context aware applications can be created that makes use of context information in many different ways, thus filling a number of different functions:

Applications can be created that display context information to the user or use context to propose selections of actions to the user. Typical examples could be applications that display maps with information about nearby sites or entities of interest (Abowd et al. 1997), or applications that provide information about the whereabouts of other people (Salber, Dey, Abowd 1999; Schmidt et al. 2000).

Another function that a context aware application can provide is to discover computational services based on the available context information. These services could either be presented to the user directly, or they might be used by other software components without even noticing the user. A typical application of this kind might be an application that finds the printer that is closest to a person's current location (Schilit et al. 1994). Applications could also monitor context information so that when a specific "situation" is identified, some action of the application might be triggered. This could be location aware reminder application (Beigl 2000) or a recording whiteboard detecting the presence of a meeting and automatically starts recording the meeting notes (Brotherton, Abowd 1998).

Finally, applications could attach context information to captured data for later retrieval, as for example in the memory augmentation applications Forget me Not (Lamming, Flynn 1994) and the Remembrance Agent (Rhodes 1997).

2.4.2 Towards a situated perspective

In parallel with the technical advancements in different areas a shift has happened in our understanding of humans and how we interact with computers. In the childhood of the human-computer interaction (HCI) research area, cognitive psychology was seen as the key to understanding the interplay between man and machine, with a strong focus on how the design of the application inflict different kinds of cognitive loads on the user. During the latest years the HCI community has expanded vastly and has also started to understand the importance of different kind of contextual factors outside the person-machine interaction. Research areas like computer supported cooperative work (CSCW) and the very recent ubiquitous computing area has started to borrow theories and methodologies from sociology to increase the understanding of the context in which that interaction emerges. In this way it becomes possible to examine how different contextual factors affect the interaction with computers, such as social, cultural and organisational factors, as well as the current physical environment etc.

An example of this is that ethnomethodology and ethnographic studies have become popular methods within HCI for requirements analysis and system evaluations. In ethnographic studies the researchers immerse themselves into a certain culture, thereby trying to understand how the members of that culture experience different situations. The researchers makes no assumptions in advance regarding what contextual factors that might affect a person in a specific situation but rather tries to understand a situation as a whole and how the persons in that situation interprets that situation.

According to *ethnomethodology* (Dourish 2001), people's actions are determined by a shared set of understandings, which has emerged within their community or culture. The everyday actions within this community are then perceived as rational with respect to those understandings. Ethnomethodology more specifically tries to identify the different rules that has emerged in a specific setting, rules that the people in the setting adhere to in order to manage their everyday situations and problems.

The notion of *situated action* introduced by Lucy Suchman (Suchman 1987) is based on the theories of ethnomethodology and deliver a strong criticism against the artificial intelligence community who tried to make presupposes about peoples actions in different situations, assuming that peoples actions are determined by rational plans. Suchman instead argues that people's actions are formed by the current situation, highly affected by the constraints and opportunities present here and now.

Activity theory (Nardi 1996) is another theory that stresses contextual factors as crucial when trying to understand and describe situations or activities. The theory can be used to model the world in terms of subjects transforming objects using different kinds of artefacts, and have been used extensively as a tool for analyzing situations of computer usage. All the theories above show that very much care should be put into examining the presumed context of usage when designing a computer system, and tailor the system for that specific situation. The common way to do this in traditional PC based systems is to make some kind of study of the context of use before or while designing the system.

In the new types of computer environments described earlier, context factors will play a far more important role for the usability of the systems. In mobile systems, hand held computers, telephones etc. the physical context is constantly changing. The user might move between different locations with different properties. There might be groups of people doing collaborative work using different kind of artefacts and the technical resources available might be changing dynamically. Given these circumstances, a static description of contextual factors will not be enough in order to create systems that can handle these shifts of context in an optimal way.

2.4.3 Understanding the notion of context

The notion of context has been used in several different research domains, with quite different meanings. Within language research for example, the notion of context plays an important role when trying to understand the nature of human communication and might cover a wide range of issues such as the common history of the participants engaged in communication, the social setting, body postures, the choice of specific words etc. Different sociological domains such as ethnography or activity theory also

use a very broad definition of context including all possible external factors that might affect the behaviour of people in different social situations.

Within the newly established field of *context aware computing* (Dey 2001) the common goal is to try and provide a computer system with information that is *relevant to the activity* that the application is trying to support, information that is not provided directly through the interaction with some user but rather collected using some sort of sensors. In this domain the term context mostly refers to only a small amount of measurable properties of the physical environment.

It is important to understand the difference between how the notion of context is used within the context aware computing domain compared to how it is used within sociology and language research (Duranti and Goodwin 1992). In the latter case the context notion is primarily used in order to explain the behaviour of people, thus including all the external factors that affects what a persons actions and all the factors that provides meaning to those actions. Given the complex nature of the human mind and the social world that we are embedded in, creating a model for how contextual factors affect our behaviours is a nontrivial if not an impossible task. Such a model would most likely be incomplete, thus failing to fully explain (or predict) the behaviours of people.

Attempts have been made, especially within the research field of user modelling, to create models of persons (including some contextual factors) in order to understand what the person is trying to do and provide support for that task. These attempts often failed because of the complexity of how our actions are determined by contextual factors, as shown by for example Lucy Suchman (1987), Saul Greenberg (2001) and Salvador et al (2003).

In the case of context aware computing, the "context" that is being referred to should rather be understood as *some contextual factors relevant to the use of a computer application*. In general these contextual factors are some measurable properties that have been identified as being relevant to the application task. These properties are fed as input to the application that can act on them in different ways. A typical example of such an application could be a tourist guide that shows nearby sites of interest on a map based on information about the location of the user.

The simple and in many ways incomplete description of a situation used in this kind of applications has got very little to do with the rich set of contextual factors that is needed to explain the behaviour of a human being. In fact one could argue that one should try to avoid using the term context aware computing since it might be misleading. It might e.g. suggest that an application can have some kind of model containing all contextual aspects that affects the usage of the application. The statement that an application is *aware* of the context, also suggests that the application has some humanlike mental properties. In fact even for a human being it seems like a strong statement to say that the person is *aware* of the contextual factors affecting the person's behaviour. A better term might be context *dependent* or *sentinent* computing, but since the previous term is so widely used it will be continuously used in this thesis.

2.4.4 Sensing and making sense

The title of this thesis "*Sensing and Making Sense*" implies that the information you receive from sensors are not immediately meaningful or useful. This clear demarcation between sensing and sensemaking stems to a large extent from a phenomenological perspective of perception and human action. As was pointed out by one of the early phenomenological philosophers, Merleau-Ponty (2002) human perception is not a process of passive registrations of properties in the world. Instead it should be understood as an activity where we actively examine the world around us by directing our perception to entities in the world. We choose to direct our eyes towards certain objects and when we examine an object with our hands we do not only touch is but turn it around in our hands and stroke it with our fingers.

A central idea within phenomenology is the rejection of the dualist perspective separating mind from body. Instead there is a basic assumption that our cognition is situated in the world. Any sensed phenomena of entities in the world will only become meaningful in relation to how we perceive it in relation to the rest of the world, and our experiences of earlier phenomena. According to Merleau-Ponty technology can be seen as extensions of our bodies, through which we can perceive and understand the world. In one of his examples he describes a blind person using a stick to "see" the surrounding environments. The sensory phenomena acquired from the usage of the stick will however only become meaningful in relation to the persons previous interactions with the world. A reading from a sensor could be compared with the phenomena provided by the stick. These readings only become meaningful for a person through the way they manifest themselves to that person. A temperature reading from a thermometer used to determine the temperature of water before taking a bath becomes meaningful because we have been able to interact with the device over time and thus been able to create mappings between the numbers on the thermometer with our bodily experiences from touching the water (Figure 3). In the case of a context aware application, the sensor readings will manifest themselves through the behaviors of the application. The *meaning* of the sensor readings will thus only reveal themselves through the interactions with the application. The sensor readings should thus not be seen as true representations of aspects of the world, but rather as a piece of a phenomenon that will later manifest itself to a person and then be rendered meaningful in relation to that persons subjective experiences. This leads to the conclusion that any representation of sensor information will only have an *instrumental* function in order to achieve the sought application behavior.

A problem with this view is the subjective perspective. If a sensor reading is meaningful for one person, how can we assume that it will be meaningful for someone else? This problem is addressed by another phenomenologist Alfred Schutz (Schutz 1975), under the notion of the *intersubjectivity problem*. Schutz proposes a solution to this problem based on the fact that we share a common life world. We can thus assume that other persons have got similar experiences as ourselves, and thus will make sense of certain phenomena in a similar way as we do. The intersubjective sensemaking is also an activity that takes place over time. By interacting with each other and sharing a common environment we create common experiences that makes it easier for us to communicate and agree on the meaning of symbols and language that is meant to describe properties of the world. Following from this line of reasoning is that any

representation of sensor data should be seen as a social agreement that have been evolved over time and in relation to some meaningful activity. The existence of different temperature representations (Celsius, Kelvin and Farenheit) have for example to be understood in relation to the activities and social settings in which they have been developed and maintained.

To summarize this it can be stated that the phenomenological perspective on context awareness implies the following:

- Sensor information becomes meaningful only when put to use in an application
- Internal representations of sensor information in a computer system only have instrumental functions
- The symbols used to describe the sensor information should be seen as a common social agreement within a community sharing similar experiences.

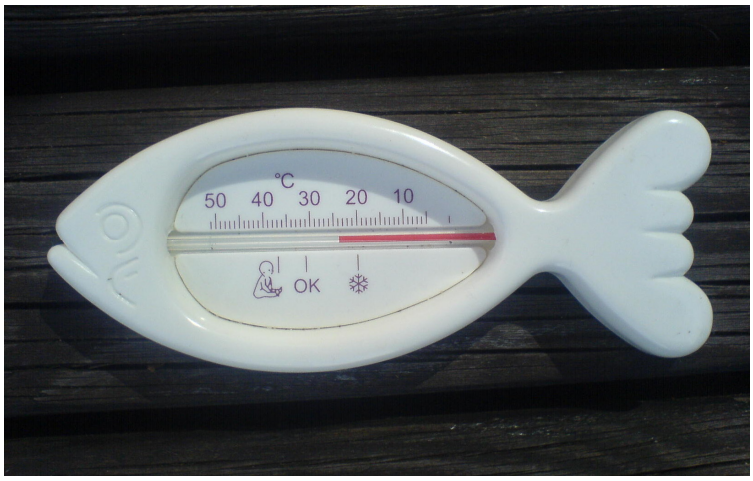


Figure 3. A bath thermometer with different symbolics to represent temperature

2.4.5 Defining the term context information

Since the notions of context and context information can be understood in many different ways, a definition of how context information should be understood within the scope of this thesis is a necessity. A first distinction is to try and avoid talking about context and instead focus on the term *context information*. By doing this it hopefully becomes clearer that we are focussing on pieces of information that describe certain aspects of the context of for example an activity or person, rather than the more holistic view of the term used in other domains.

A piece of context information should be understood as being a *description* of *something* and is equivalent with either some form of sensor reading directly or a statement that is derived from one or several sensor readings. Such a piece of information is constituted of two basic elements: On one hand a symbolic representation of the

actual 'value' or reading from the sensor and on the other hand a representation of the entity that is being described.

Following definition of how the notion of context information should be understood in this work is loosely based on the definition of context used by Dey in (Dey 2001), but has been modified to better suit the framework of this work. The definition reads:

"Context information is any descriptions of the situations of entities that can be used by a software service in support of an activity"

Where a "situation of entities" should be understood as:

"Representations as well as properties of entities and relations to other entities"

Where the entities typically are physical entities such as people, places or computers but also embrace system components like software services or organizational entities like projects or groups.

Chapter 3
Investigating the problem space - Identifying three key problems for context information middleware design

This chapter will explain how middleware can be useful to support the creation of context aware applications. Three key problems related to the design of context information middleware are introduced, including a brief overview of how these problems have been addressed. An overview of three different middleware designs is also presented, where the different designs in different ways are used to illuminate and evaluate aspects of the key problems. In later chapters the system designs and their specific characteristics will be presented in further detail. Approaches to deal with the specific problems introduced here will also be analyzed more thoroughly in later chapters.

3.1 Why do we need middleware to support context aware applications?

The need for middleware to support context aware computing has been acknowledged by several different researchers (Salber et. al 1999; Hong and Landay 2001; Hendricksson et. al 2005). There are several functions that such a middleware can fill, dealing with issues from transportation of sensor data, providing high level representations and APIs, querying functionalities and data refinement procedures.

3.1.1 Collecting context information

In order to build context aware applications, contextual information must in some way be collected from the situation in question. Some context information can be gathered using entirely computational components. Such computational sensors are for example used in various instant messaging applications, where information about network connection, recent button-presses and mouse movements is used to indicate to other users whether you are available for communication or not. For most context aware applications the computational sensors are however not sufficient. Instead, information about the real world is collected through different kinds of physical sensors.

The collection of context information is a two step process. First one has to consider which contextual factors are necessary to collect in order to achieve the sought application behaviour. The next step is to identify an appropriate method to collect those contextual factors, including the choice and deployment of sensors. Often several different methods could be used in order to examine a specific contextual factor.

As an example of how this can be done, one could examine how the most commonly used contextual factor, namely the location of people, can be fetched. A suitable method, at least for indoor localization, could be described in terms of identification, where typically the identity of a person is detected at a specific location, or the other way around, that the identity of a location is detected by a wearable device. Such an identification method could then be implemented using numerous different sensors, such as IR-beacons and receivers, card readers, iButton readers, fingerprint recognition sensors, short distance radio (RFID) sensors and different kinds of image processing sensors like face or object recognition sensors. This identification approach differs a bit from other more direct methods of context information assembly such as the ones used for collecting information about the physical properties of a specific location. Such information is mainly gathered directly using different kinds of environment sensors, like thermometers, photo resistors, pressure sensors etc.

3.1.2 Supporting deployment of sensors

Experiences from work with sensors in a number of different settings (Jonsson & Mattsson 2002) have gained useful knowledge both regarding what context information that is useful in different settings, as well as experiences regarding the implementation of sensor based systems.

Many context aware applications use their own specific implementations of sensors and a lot of effort is generally put into low-level implementations. The degree of specialization of these implementations makes it hard to reuse parts of the system for other purposes, as well as it makes it hard to modify the system according to future changes in the application.

We believe that the implementation process could be supported in several ways. Simplifications regarding low-level installation of sensors would decrease development time a great deal. General components could be used that only requires smaller modifications to support the sensors being used. Some kind of general tools could also

be used for the process of subscribing for or fetching sensor data. Ideally, it should be possible to fetch the sensor data using several different protocols to ease the integration with different applications.

Our own experiences as well as the work of others (Salber et. al 2001, Yoshimi 2000) make it possible for us to draw some conclusions regarding the usage of different kinds of sensor information in applications.

- Some context information, like the location of people, is very common in different applications, thus sharing sensors among services might be a good idea.
- Certain context information could be fetched using different types of sensors, such as the identity of people and objects, the level of activity in a room etc, making it possible to use higher-level abstractions to hide the details of the underlying sensor implementation.
- Different applications may want the sensor data from one sensor in different formats, or on different levels of abstraction.
- Many sensors are similar in terms of implementation. Examples of this are e.g. simple AD converter based sensors like light and pressure sensors, or binary sensors like motion sensors and step sensors. This would suggest that general templates could be defined that only requires minor changes to support different kinds of sensors.

3.1.3 Hide heterogeneity and provide useful abstractions

The middleware could also have the role of hiding a lot of complexities existing in a pervasive sensor environment. Hardware components might range from resource-poor sensors, actuators and mobile client devices to high-performance servers. These devices might be accessed via a variety of networking interfaces and programming languages. A middleware system could then provide a unified interface to this complex environment, allowing end user applications to be ignorant of the details of all components. A middleware system could instead provide generic interfaces through which you can access sensor information. The sensor data can also be abstracted to various degrees, providing higher level representations or interpretations of the sensor data, making it easier for the applications to make decisions with respect to e.g. a user's activities.

3.1.4 General schematics of a context aware middleware

Many of the proposed architectures for context aware systems as the ones described above contain approximately the same type of components providing certain functionalities. These functional components can be organized in a generic layer structure as shown in Figure 4. Similar layer architectures have earlier been presented by e.g. Hendricksen et al. (2005).

On the lowest layer in these architectures are the actual sensors including both various kinds of hardware sensors as well as software components like for example a sensor that indicates whether the keyboard or mouse of a PC have been used recently.

In a pre-processing layer the sensor readings can be refined e.g. by reducing noise by calculating mean values and adapted to appropriate representation formats. The components in this layer are also responsible for disseminating the sensor data to other components. This includes packaging the data and communicating it using an appropriate protocol.

The repository layer acts as an intermediate storage point of sensor data where data from several sensors can be assembled. The data in the repositories are accessible by either applications directly or by various forms of refinement components.

In the reasoning layer data from several sensors can be combined and analyzed in order to for example infer higher level situations or to transform the data into other representation formats. The resulting data can either be provided to applications directly or posted back to the repository.

Finally in the application layer there are various APIs or protocols by which the applications can get access to the sensor data. This could be in the form of query APIs, event listeners or various subscription mechanisms.

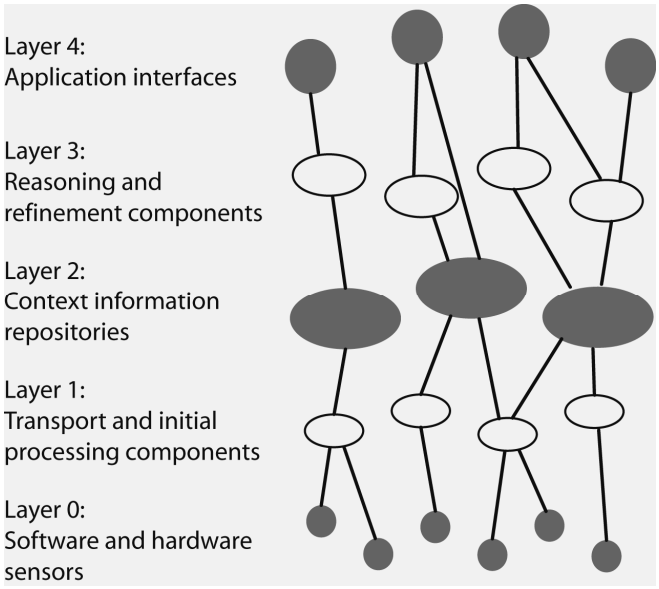


Figure 4. A generic description of the components of a context aware system

3.1.5 An overview of existing systems

Most of the early prototypes that would display a context aware behaviour were created in an ad-hoc fashion, mainly in order to investigate the problem space (Long et al. 1996). This means that the application designers had to consider a lot of different issues, including the details of implementing and reading sensor data, distributing sensor data, transformation of sensor data as well as the details of the application adaptation behaviour. From a software engineering perspective, this makes developing context aware applications very cumbersome.

Several researchers have realized that the process of collecting and distributing context information to applications could be simplified. Several support systems have thus been created that in order to simplify parts of the design process (Conner et al. 2001; Holmquist et al. 2001).

The problems that these support systems are targeting can roughly be divided into two categories; support for sensor deployment and support for context data management. Within the first category the existing support systems often consists of some hardware sensor platform with communication abilities and a number of preinstalled sensors and maybe some possibilities for further extensions.

The solutions for the other category of problems, regarding management of context information, often consists of different kinds of software components e.g. in the form of toolkits (Salber et al. 1999) or infrastructures (Hong and Landay 2001). These systems can e.g. abstract or compose sensor data into higher level context information or via some context model identify the occurrence of specific situations. Other systems focus on how to simplify the integration of contextual components in applications by using different kinds of computational abstractions such as widgets.

One of the earlier and most influential systems available is the Context Toolkit, developed by Anind Dey (Dey 2000). The context toolkit system focuses on supporting the application design process by encapsulating context information resources in familiar programming abstractions such as widgets, making the context information more easily accessible for system designers that are not experts of sensor technology.

While the Context Toolkit system focused more on how to create standalone applications, another approach is to create *infrastructures* that can be used to provide context dependent functionalities to applications. The basic idea with an infrastructure approach is to provide a shared resource for applications in order to hide complexity and heterogeneity, and to relieve the applications from some of the computation overhead. The Context Fabric (Hong and Landay 2001) system is an example of an infrastructure based context information middleware system. In the Context Fabric system context information is provided in XML documents over http. The information is presented in a web-like manner with links creating relations between entities.

3.1.6 Identifying key design problems for context information middleware

Based on the experiences from work on several versions of context information middleware systems a number of key design issues have been identified as being crucial when designing context aware middleware in general and especially in the context of ubiquitous service environments. Following issues will thus be discussed and analyzed to some detail.

One issue concerns the interpretation of context information in the middleware systems. In order for sensor data to become meaningful for applications it must generally be matched against some kind of context model. Matching the data against a specific model might however restrict the possible usages of the system.

The next problem concerns how you work with services in context information middleware. More and more of today's computer systems can be modelled in terms of service environments, where atomic computational entities become accessible and controllable over the internet. By using sensor information from the service environments you can achieve a lot of interesting behaviours, such as dynamic service composition, where services can find and interact with each other automatically. Crucial problems to enable this kind of functionality include how the services should present themselves to each other and how the discovery process can be enhanced.

The final problem concerns how context aware systems can benefit from involving users in the system setup and maintenance to a higher degree. By doing this you can not only address performance oriented problems concerning for example reasoning and representation but the users will also be empowered in various ways.

3.2 Achieving generic interpretation in context information middleware systems

How can context information interpretation processes be designed in the middleware that make it possible to perform useful interpretations on context information while still supporting a wide range of applications? The typical functionality of a context aware system is that some application is changing its behaviour based on input acquired from some kind of sensor. This process can however be implemented in different ways, going from very simple causal "if-then" relations to very complex interpretation and decision-making processes. Since introducing complex reasoning typically creates a lot of overhead in terms of computing power, and also introduces a number of pitfalls (which will be discussed later) compared with a more straightforward approach, it is first interesting to see where and why this kind of reasoning is needed.

3.2.1 Handling data inconsistencies and contradictions

Data coming from sensors might be erroneous in many ways. It might e.g. be noisy, due to the inherent properties of the sensors. This kind of errors could be dealt with by methods such as computing mean or median values. There might also be data from different sensors that in some sense are inconsistent or contradictory. This scenario is somewhat more complex, since you have to decide whether to pick one piece of data and discard the other, or whether to try and combine the two pieces of data.

Evaluating the quality of sensor data could be done in several ways. You could either have some kind of reliability score for each sensor or type of sensor, describing how trustworthy it is, or you could try and compute how accurate the sensor has performed in the past.

Combining data from different sensors could be simple e.g. if the sensors happen to be of the same type and provides some kind of numerical data. It might however also be very difficult if the sensors provide data in different formats. If you are to compare different data of this kind, the only option you have is to try and transform the data into a common format to make it comparable/combinable. Sometimes this is a

straightforward operation, such as if you have temperature readings in Celsius and Fahrenheit. Sometimes it is however not possible to transform directly from one reading to the other. In these cases what you need is a model that in some sense describes how different concepts and entities in the world are related to one another.

3.2.2 Abstraction of sensor data

Often in context aware applications, you are not interested in the sensor data itself, but rather in different kinds of abstractions or inferences you can make based on the sensor data. A typical abstraction could be to go from sensor data from an accelerometer to descriptions of different kinds of motion.

Often you also want to draw different kinds of *conclusions* based on the available sensor data, where the sensor data then can be seen as evidence supporting the conclusion. To be able to do this kind of inferences you have to make a number of assumptions regarding data you are considering. These assumptions about real world entities and how they relate to each other are normally formalized and encoded using some kind of *context model*. Context models can be implemented in a number of different ways, and vary greatly in complexity. A crucial part of the interpretation process thus concerns what information that should be incorporated in the models as well as how you choose to represent it.

A further level of complexity is added if you want a system that tries to take actions based on the available context information combined with some user preferences. For these cases you need some way of representing the user's preferences, typically this is done using some form of user model. Based on the context and user models rules can be created, describing what actions should take place given clues about the user's current situation.

3.2.3 How high to aim? Choosing an appropriate level of abstraction

When interpreting data coming from a set of sensors, you can strive for different levels of abstraction in the end result. Level of abstraction in this case is determined by the amount and quality of the assumptions that are made concerning certain aspects of the world. An example of a low abstraction level inference could be e.g. that the electric pulse generated by a magnetic field sensor in a pedometer means that a human being has walked one step. Here you have to make assumptions such as that the device is worn properly by a person etc. If you want to infer more complex activities such as the occurrence of a meeting based on information from calendar information and some location sensors, you have to make assumptions such as: Meetings occur when people are co-located, and when the participants have a scheduled event in their calendars at that time. This kind of inferences concerning aspects on human activities mostly requires assumptions with high uncertainties and often requires a lot of information to be encoded in the system. Given the complexity of the high level abstractions and the uncertainty of the assumptions made, in general the higher level of abstraction you strive for, the more probable it is that you get an erroneous result.

When designing context aware systems you have to make a design choice with respect to how complex inferences you want to be able to make. You can either choose to use

complex models, making it possible to infer high level situations from the sensor data, or you can chose to have simpler models, allowing less interesting but potentially more correct inferences. There is of course not a clear mapping between the model complexity and the risk of errors in the inferences, since a simple model can be seen as more incomplete, e.g. lacking information about special cases etc. Complexity in this sense should thus be understood the amount of semantics encoded in the system in combination with the level of abstraction that the model is aiming at. A complex context model according to this definition will also have further drawbacks. Inference engines working on complex models will for example require considerably more computation than a simple model (Strang, and Linhoff-Popien 2003; De Brujin 2003). Complex models require more maintenance. Somebody will have to encode these models and also keep them updated. In a distributed computing scenario, this might be a cumbersome task since the models might be scattered over a large number of devices.

3.2.4 Aspects of context formalization

There are several possible angels to examine the problem of how to formalize context information, as have been shown by e.g. (Strang et al. 2003). Several approaches have been proposed to encode context information to support sharing of information and different kinds of inference tasks.

Tag based models

Tag based models are the simplest form to represent metadata or context information. In tag based models an entity is described entirely by entirely separate text strings. Lately several web based services have appeared that uses this approach to add metadata to web pages (e.g. del.icio.us) or images (e.g. Flickr). For these settings this approach has been referred to as *folksonomies* (Vander Wal 1995; Christiaens 2006). A crucial feature in the folksonomy approach is however that it is the end users that provide the tags.

Key value based models

The model of key-value pairs is a very simple encoding format for contextual information used by e.g. (Schilit et al. 1994). The key-value modelling approach is frequently used in distributed service frameworks. In such frameworks, the services itself are usually described with a list of simple attributes in a key-value manner, and the employed service discovery procedure (e.g. SLP or Jini) uses an exact matching algorithm on these attributes. In particular, key-value pairs are easy to manage, but lack capabilities for sophisticated structuring for enabling efficient context retrieval algorithms.

Markup scheme models

Common to all markup scheme modelling approaches is a hierarchical data structure consisting of markup tags with attributes and content. The typical encoding format is extensions of the Extensible Markup Language, XML. An example of this approach is the Comprehensive Structured Context Profiles (CSCP) by Held et al. (2002). CSCP does not define any fixed hierarchy. It supports the full flexibility of RDF/S to express

natural structures of profile information as required for contextual information. Attribute names are interpreted context sensitively according to their position in the profile structure. Other examples of markup based models are e.g. Stick-e-notes situation model (Brown et al. 1997) and the Context Fabric system (Hong and Landay 2001).

Ontology based models

Ontologies are a more dynamic approach to specify concepts and interrelations onto a data structure utilizable by computers. Using ontologies provides a uniform way to specify the model's core concepts as well as an arbitrary amount of sub concepts and facts, altogether enabling contextual knowledge sharing and reuse. This contextual knowledge is typically evaluated using ontology reasoners. The CONON context modelling approach by Gu et al. (2004) uses an upper ontology which captures general features of basic contextual entities and a collection of domain specific ontologies and their features in each sub domain. The CANON ontologies are encoded in OWL-DL which has a semantic equivalence to well researched description logics. This allows for consistency checking and contextual reasoning using inference engines developed for description languages. An emerging context modelling approach based on ontologies is the CoBrA system (Chen et al. 2004). This system provides a set of ontological concepts to characterize entities such as persons, places or several other kinds of objects within their contexts. The CoBrA system uses a broker-centric agent architecture to provide runtime support for context-aware systems.

3.2.5 The generality problem

As stated above, the generality problem addressed here concerns how context information interpretation processes can be designed in the middleware that makes it possible to perform useful interpretations on context information while still supporting a wide range of applications. This becomes problematic due to the clash between on one hand the requirement that a middleware system should be application independent in order to be useful, and on the other hand the need for and internal representation format of context information that in some sense is application dependent.

The generality problem has earlier been discussed in Artificial Intelligence by for example McCarthy (1986). The original problem was related to epistemological concerns on whether it is possible to describe general knowledge or facts that are valid in all possible situations. As McCarthy states (McCarthy 1993, McCarthy and Buvac 1997) a fact can only be said to be valid within a specific context. McCarthy's proposed solution to the generality problem was to try and formalize aspects of the context in which the facts are stated to be true and then include them into the system. As will be shown here, this is not a solution to the problem, since any formalisation of the context will also suffer from the generality problem. In other words: The concepts and facts you use to describe the context of something are also context dependent.

The generality problem as addressed here is however not focussing on how to create generic descriptions. Instead the focus here lies on how you can achieve generic

mechanisms for interpretation. The aim is to create a versatile system, and whereas generic knowledge representations would be helpful it is neither a goal nor a requirement.

In context aware systems the facts and concepts used to describe the context of something is formalized in *context models*. There are two major aspects of the formalisation of context models that will affect the versatility of the system. The first one is similar to the AI – generality problem: Is the context model *valid* for all application domains in which it will be used? The second aspect concerns whether the context model is *sufficient* for all the domains. Context models can be seen as descriptions of entities in the world. As any description context models must be seen as being incomplete, as otherwise the description would become the entity itself. It is also highly questionable whether there are certain properties of the described real world entities that in some sense are more essential than others. What one has to bear in mind is that any description of an entity is only meaningful in relation to a specific purpose. It is thus important to remember that the models only have an *instrumental* function, and have to be designed with a particular usage in mind.

A problem then arises when you want to store, combine or abstract context information in a generic way without a specific application area in mind, which to a large extent is the case with a context information middleware.

3.2.6 The addressed problem

Some specific aspects of the generality problem described above will be addressed in this work. The question that is specifically addressed is: *How can you design mechanisms that allows for transport, storage and interpretation of context information and at the same time minimizes the restrictions on the versatility of this information?*

A question that has to be posed is what kind of answers that are possible to achieve with respect to the question stated above. How do you evaluate whether a design solution "minimizes the restrictions on the versatility"? The most obvious way to get any answers to this question would be to implement a broad range of applications on top of each of the proposed design solutions. But except that this would be a very time consuming approach it is not obvious what it would actually show. What is for example a *broad range* in this case?

Instead each implementation is evaluated on one hand using smaller proof of concept implementations, verifying that the mechanisms actually provide the functionality they are designed for. On the other hand, to evaluate whether the design proposals addresses the generality problem, a number of assumptions are made with respect to how different properties of CIM systems are assumed to affect generality. Then an analysis is made of how each of these system properties manifest themselves in the implemented systems and which role they play with respect to the generality problem.

3.2.7 Crucial properties

When addressing the generality problem, aspects of how to model the context information is one of several mechanisms affecting the versatility of the system, where others include e.g. how to divide the interpretation between different system layers, how easy the system is to manually adapt to new areas etc. A number of design

parameters have been identified which are expected to have an effect on the versatility of a middleware system. But first the notion of versatility has to be defined more clearly: For the following examination of this problem we will say that a CIM-system is considered to be more versatile if it:

a. initially supports a wide range of application domains.

b. is easy to *adapt* to new application domains.

By including adaptability as a part of the problem allows us to examine the CIM systems in a broader context including aspects of usage and maintenance of the systems. Following stipulations are also made with respect to how certain design parameters will affect versatility:

Layer division: *Versatility increases if a larger part of the interpretation is handled in the application layer instead of in the middleware layer.* It is not obvious how large part of the interpretation that should take place in the middleware layer, and how much that should reside in the application layer.

Representations: (A CIM system can be designed with or without representations of instances of real world entities) *A system that does not contain representations of instances of entities is easier to modify to fit new application domains.* The generality aspect is thus also affected by whether you choose a situational or representational encoding of the context information. Whereas a formalization that contains representations of real world entities might become more coherent, a situational or event based encoding allows for parallel interpretations inconsistent and incomplete models.

Level of detail: (The amount of information included in the context model, in terms of the number of concepts included and the number of claims made on these objects, including how they relate to each other etc.) *A system that contains a more detailed context model might be able to cover more application domains but is also harder to modify once it is not applicable.* Including a lot of concepts or 'knowledge' in a context model makes it possible to make higher level deductions on the activities the sensor information is describing. On the other hand, it might be harder to find a rich model that is applicable for a broad range of applications.

Openness: *A system with an open taxonomy context model is easier to adapt to new application domains.* The taxonomy of the context model used can be either open or closed. In a closed taxonomy, the vocabulary is determined in advance, whereas in an open taxonomy, like in so called folksonomies, additions and changes can be made continuously.

3.2.8 Proposed interpretation mechanism designs

Within the space expanded by the different design parameters outlined above, there are certain points that have been considered especially interesting and that have been examined through various extents of implementations.

A Simplistic representational model

[Thin model, basic interpretations made in the middleware, representational, not adaptable, semi open]

In this design a simplistic context model is encoded in the system structure. This model consists of three basic entities: Persons, places and groups. These entities can have different kinds of relations to each other. Additional context information can be added that describes each entity, but the responsibility for the format of this context information is pushed up into the application layer.

Supporting application specific interpretation

[Varying model, application specific interpretation in middleware, situational, adaptable, semi open]

In this design approach the amount of semantics encoded in the system in terms of generic context models has been further minimized compared with the previous approach. In this design an underlying assumption is that since any sensor data can be represented in an almost infinite number of ways, one should not try to find generic models for context information. The only factors that should determine how context information should be interpreted are thus the specific needs of specific applications. The sensor data you subscribe for from the sensors is thus encapsulated into small packets called "context elements", representing the sensor data in a *close-to-sensor* format. Context Refiner modules can then be used that transform or combine the context element into new elements. The rules that are used to transform the context elements are provided by the applications.

Collaborative tagging of locations

[Thin model, basic interpretation in the middleware, representational, adaptable, open semantics]

This design focuses on how location can be described using a very simple base-model, based on an open a key-value tagging system. The users of the system are free to add semantics describing each location in order to make them understandable by other people or tailored to be machine readable by specific applications. A key requirement for this kind of open ontology systems to work is that they are used within the borders of *groups sharing some contextual references*. An assumption that is made is that naming practices will evolve within these communities over time.

3.3 Using context information middleware to achieve dynamic service discovery

An interesting application domain for context information middleware is to try and use it as a resource to manage larger sets of software services. Encapsulating computation in terms of services is an increasingly popular software engineering approach, for example in the domain of ubiquitous computing. Since the notion of service environments is a fairly broad concept and is still fairly new and immature, it is necessary to specify what service environments we are talking about and what kind of behaviours we are looking for.

3.3.1 Ubiquitous service environments and service discovery

In chapter two the notion of ubiquitous service environments or USEs was introduced. The USE framework is to be seen as a tool to help analyze and understand computer dense environments, where each computing device is broken down and rephrased in terms of a number of *services*. By using this terminology it becomes easier to understand how the separate devices can be combined in various ways to form new equivalents to applications in the form of distributed environments.

In order to realise these dynamically recomposing service environments a crucial component is for the services spread out on the various devices to be able to present themselves and to find each other in practical and meaningful ways. This problem has been addressed by various techniques under the notion of *service discovery*. The USE domain however has some specific characteristics that make it interesting to investigate new means to achieve this goal.

An important factor in a USE is the containment of interaction in shared physical spaces. Physical *proximity* thus becomes an important factor that has to be taken into account.

The services in a USE might be implemented on different standards making it important that any service discovery mechanisms are *service technology agnostic*.

The services constituting a USE is constantly changing depending on the needs from the current activity as well as due to mobile devices coming and going. Service compositions should therefore happen *dynamically*.

3.3.2 Examples of implemented ubiquitous service environments

From the USE philosophy a number of instances of USEs have been successfully implemented. So far these implementations have been almost solely based on Java with the Jini extension. An especially useful feature in Jini is the discovery mechanism whereby clients can locate and employ services without prior knowledge of the service such as host names and port numbers.

The first implemented USE, named fuseONE (Werle et. al 2001) is a meeting support system that targets the problem of document management in project meetings. The system contains a type of active services called Active Documents that can use information concerning who are in the same room to identify the occurrence of a meeting. When the meeting starts the document presents itself on a public display in the room. The system also supports sending documents between computers in the same room. A context sensitive desktop shows the possible receivers of documents present in the room.

Another implemented USE was a system for non-intrusive messaging (Jonsson et al.. 2002). In meeting situations, incoming messages often have a disturbing effect on the participants. In an attempt to tackle this problem a prototype was created where the presentation of the messages could be varied in different ways, by using different modalities, personal and public displays, peripheral and ambiguous renderings etc.

Private messages where e.g. sent to private displays embedded at the position where a person where seated. The displays were built using network connected handhelds embedded in the table, with iButton-readers (Dallas Semiconductors 2002) connected to them. The iButton readers were used to personalize the display services. All the components in this USE were standalone Jini services. An active service was used to decide where to send the messages and in what format.

Other smaller implementations are the Picador service that allows for sending pictures to a shared display, where the images can be arranged and modified in different ways, and also the key-sender service that uses the JXTA protocol (Gong 2001) to send keystrokes from one device to another in a peer-to-peer manner.

So far most of the implementations have been based on the capabilities provided by Jini. Jini is however not a prerequisite for building USEs. Even though it has some nice features such as service discovery support, it also has many drawbacks. It is limited to Java, excluding a large part of the software developing industry. The service discovery function finds all services in the local area network, which is sometimes too fine-grained and sometimes not precise enough.

3.3.3 Service composition

In traditional PC-based systems the concept of applications is used to encompass a preconfigured set of services to the user of the device. In distributed ubiquitous computing environments, the notion of what is an application is much more unclear.

As was mentioned before, a key issue to make USEs a reality is the means to assemble sets of services in a dynamic way. This assembly is to a large extent an issue of combining individual services with each other in a dynamic way.

The composition process can roughly be divided into three sub-problems:

1. Service description: How do you describe the service in a way that makes it possible for a user or another service to decide whether to try to connect to it or not?
2. Service discovery: How do the individual services publish information about themselves and how do they find other services or their descriptions?
3. Service binding: How does the interaction between the services take place once they are found?

Normally these three steps are intimately intertwined in the technologies performing the composition. Technologies for connecting computational entities over a network already exist within the distributed computing domain. In this work we examine how you can build upon these existing technologies to achieve the dynamic behaviour outlined earlier. When trying to achieve dynamic composition of services using these existing technologies, it is important to make clear what service composition actually means with respect to each different technologies. You can divide the existing ways to combine computational services into a number of categories:

Loose coupling: This class of services has a generic interface that is publicly available for other services. There typically also exists a description of the interface that can be examined or queried by other services. Web service technology is the most well known

technology of this kind, with its service description language WSDL. To compose two services of this kind is simply an issue of making the WSDL document available to a service that can interpret it correctly.

Shared interface composition: There are several technologies for distributed computing that provides means to access computational resources remotely via some kind of shared API. The APIs can be more or less generic. Examples of such technologies are Java RMI, CORBA and XML-RPC. Combining two services of this kind is mainly an issue of providing a pointer to one service to another (or a piece of code or stub as in RMI)

Protocol composition: Whereas the technologies above provide distributed access in form of method or procedure calls, another class of services communicates using protocols, where the sequence of commands is crucial. Examples of such protocols are e.g. the SIP protocol or the very simple but very useful http protocol.

3.3.4 What does *dynamic* composition mean?

Dynamic composition simply means that services will connect with each other automatically, given that some conditions are fulfilled. In this chapter we are specifically examining context dependent service composition, where context information in different ways affects the service composition process.

Achieving dynamic composition requires some degree of autonomy in the services that are involved in the process. The degree of autonomy might however vary quite a bit. In most of the examples presented in this work the services are fairly simple and non-proactive. For this kind of services the composition task consists mainly of initiating the communication between the services. In other service composition tasks where one or both of the services have more agent-like reasoning capabilities, the composition task can become more complex, including more extensive examination of services and context information before attempting to connect with and use the other services. What should be noted is that the services in the composition process in most cases are not equal peers in the process. Typically there is one *consumer* service that in a proactive way is trying to find and connect to the more passive *resource* services.

The resulting behaviour of a successful composition does not necessarily mean that any explicit actions are being triggered that a user might notice. The effects might be much more subtle, such as the number of options in a graphical user interface might change etc. In this work we will mainly address questions related to service description and discovery and not so much issues of binding and interoperability.

3.3.5 Existing methods for service discovery

The service oriented computing perspective on the design of computer systems is gaining importance in several different domains. This has had the result that a number of different technologies for service discovery have started to appear, that aims to target the specific needs of the different domains.

The popular concept of web services provides means to find global Internet services through the UDDI and Disco technologies. Office environments can utilize technologies like Jini (Waldo 1999), or LDAP (Howes and Smith 1995) to discover

e.g. a printer service, whereas home network environments might use the UPnP service discovery. Salutation (Pascoe 1999) and SLP (Guttman 1999) are service discovery techniques that are more targeted towards the telecommunication domain.

Lightweight Directory Access Protocol (LDAP)

LDAP (Lightweight Directory Access Protocol) is a software protocol for enabling anyone to locate organizations, individuals, and other resources such as files and devices in a network, whether on the public Internet or on a corporate intranet. An LDAP directory is organized in a simple tree hierarchy consisting of the following levels:

- The root directory (the starting place or the source of the tree)
- countries
- organizations
- organizational units (divisions, departments, and so forth)
- individuals (which includes people, files, and shared resources such as printers)

An LDAP directory can be distributed among many servers. Each server can have a replicated version of the total directory that is synchronized periodically. An LDAP server is called a Directory System Agent (DSA). An LDAP server that receives a request from a user takes responsibility for the request, passing it to other DSAs as necessary, but ensuring a single coordinated response for the user.

UDDI

The Universal Description, Discovery, and Integration (UDDI) specification describes an online electronic registry that serves as electronic Yellow Pages, providing an information structure where various business entities register themselves and the services they offer through their WSDL definitions.

The Universal Description, Discovery, and Integration (UDDI) specification defines a 4-tier hierarchical XML schema that provides a model for publishing, validating, and invoking information about Web Services. UDDI uses standards-based technologies, such as common Internet protocols (TCP/IP and HTTP), XML, and SOAP (a specification for using XML in simple message-based exchanges). UDDI is a standard Web Service description format and Web Service discovery protocol; a UDDI registry can contain metadata for any type of service, described by Web Service Description Language (WSDL).

There are two types of UDDI registries: public UDDI registries that serve as aggregation points for a variety of businesses to publish their services, and private UDDI registries that serve a similar role within organizations.

Service Location Protocol (SLP)

The Service Location Protocol (SLP) is a product of the Service Location Protocol Working Group (SVRLOC) of the Internet Engineering Task Force (IETF). It is a protocol for automatic resource discovery networks based on the Internet Protocol.

SLP is a language independent protocol. Thus the protocol specification can be implemented in any language. It bases its discovery mechanism on service attributes, which are essentially different ways of describing a service. It can cater to both hardware and software forms of services. The SLP infrastructure consists of three types of agents:

1. User Agents
2. Service Agents
3. Directory Agents

The User Agents acquire service handles for end user applications that request the services. The Service Agents are responsible for advertising service handles to the Directory Agents thus making services available to the User Agents. The Directory Agent maintains a list of the advertised services in a network. SLP offers the following services:

- Obtaining service handles for User Agents
- Maintaining the directory of advertised services
- Discovering available service attributes
- Discovering available Directory Agents
- Discovering the available types of Service Agents

A service is described by configuration values of the attributes possible for that service. For instance, a service that allows users to download audio or video content can be described as a service that is a pay-per-use real-time service or a free-of-charge service. The SLP also supports a simple service registration leasing mechanism that handles the cases where service hardware is broken but the services continue to be advertised.

Jini: A service discovery architecture based on Java

Jini is a distributed service-oriented architecture developed by Sun Microsystems. Jini services can represent hardware devices, software programs, or a combination of the two. A collection of Jini services forms a Jini federation.

The overall goal of Jini is to turn the network into a flexible, easily administered tool on which human and computational clients can find services in a flexible and robust fashion. Jini is designed to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.

One of the key components of Jini is the Jini Lookup Service (JLS), which maintains dynamic information about the available services in a Jini federation. Every service must discover one or more Jini Lookup Service before it can enter a federation. The location of the JLS could be known before hand, or they may be discovered using multicast.

When a Jini service wants to join a Jini federation, it first discovers one or many JLSs from the local or remote networks. The service then uploads its service proxy (i.e. a set of Java classes) to the JLS. The service clients can use this proxy to contact the original service and invoke methods on the service. Since service clients only interact with the Java-based service proxies, this allows various types of services, both hardware and software services, to be accessed in a uniform fashion.

A user searching for a service in the network first multicasts a query to find the JLS in the network. If a JLS exists, the corresponding remote object is downloaded into the user's machine. The user then uses this object to find its required service. In Jini, service discovery is done by interface matching or Java attribute matching. If the JLS contains a valid service implementing the interface specified by the user, then a proxy for that service is downloaded to the user's machine. The proxy is used henceforth to call different functions offered by the service.

Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP), pushed primarily by Microsoft, is an evolving architecture designed to extend the original Microsoft Plug and Play peripheral model to a highly dynamic world of many network devices supplied by many vendors. UPnP works primarily at lower layer network protocols suites (i.e. TCP/IP), implementing standards at this level. UPnP attempts to ensure that all device manufacturers can quickly adhere to the proposed standard without major hassles. By providing a set of defined network protocols UPnP allows devices to build their own APIs that implement these protocols - in whatever language or platform they choose.

UPnP uses the Simple Service Discovery Protocol (SSDP) to discover services on Internet Protocol based networks. SSDP can be operated with or without a lookup or directory service in the network. SSDP operates on the top of the existing open standard protocols, using the Hypertext Transfer Protocol over both unicast User Datagram Protocol and multicast User Datagram Protocol. The registration process sends and receives data in hypertext format, but has some special semantics.

When a service wants to join the network, it first sends out an advertise (or announcement) message, notifying the world about its presence. In the case of multicast advertising, the service sends out the advertisement on a reserved multicast address. If a lookup or directory service is present, it can record such advertisements. Meanwhile, other services in the network may directly see these advertisements as well. The "advertise" message contains a Universal Resource Locator (URL) that identifies the advertising service and a URL to a file that provides a description of the advertising service.

When a service client wants to discover a service, it can either contact the service directly through the URL that is provided in the service advertisement, or it can send out a multicast query request. In the case of discovering a service through the multicast query request, the client request may be responded by the service directly or by a lookup or directory service. The service description does not play a role in the service discovery process.

Salutation

Salutation is a service discovery and session management protocol developed by leading information technology companies. Salutation is an open standard independent of operating systems, communication protocols, and hardware platforms. Salutation was created to solve the problems of service discovery and utilization among a broad set of appliances and equipment in an environment of widespread connectivity and mobility. The architecture provides applications with different services that are scattered all through out the network. It also contains functions to find out capabilities of remote services. Salutation provides features for an application to establish interoperable sessions with any remote service.

The Salutation architecture defines an entity called the Salutation Manager (SLM) that functions as a service broker for services in the network. Different functions of a service are represented by functional units. Functional Units represent essential features of a service (e.g. fax, print, scan etc). Furthermore, the attributes of each Functional Unit are captured in the Functional Unit Description Record. Salutation defines the syntax and semantics of the Functional Unit Description Record (e.g. name, value). SLM can be discovered by services in a number of ways such as:

- Using a static table that stores the transport address of the remote SLM.
- Sending a broadcast discovery query using the protocol defined by the Salutation architecture.
- Inquiring the transport address of a remote SLM through a central directory server. This protocol is undefined by the Salutation architecture, however, the current specification suggests the use of SLP.
- The service specifies the transport address of a remote SLM directly.

The service discovery process can be performed across multiple SLMs. A SLM can discover other remote SLMs and determine the services that are registered there. Service Discovery is performed by comparing a required service type(s), as specified by the local SLM, with the service type(s) available on a remote SLM. Remote Procedure Calls are used to transmit the required Service type(s) from the local SLM to the remote SLM and to transmit the response from the remote SLM to the local SLM. The SLM determines the characteristics of all services registered at a remote SLM by manipulating the specification of required service type(s). It can also determine the characteristics of a specific service registered at a remote SLM or the presence of a specific service on a remote SLM by matching a specific set of characteristics.

JXTA: A peer to peer solution

The JXTA protocol (Gong 2001) does not mandate exactly how discovery is done, since its protocols only define the message format for communication between peers. It can be completely decentralized, completely centralized, or a hybrid of the two. In JXTA Version 1.0, the following discovery mechanisms are supported:

- LAN-based discovery. This is done via a local broadcast over the network.
- Discovery through invitation. If a peer receives an invitation (either in-band or out-of-band), the peer information contained in the invitation can be used to discover a (perhaps remote) peer.
- Cascaded discovery. If a peer discovers a second peer, the first peer can, with the permission of the second peer, view the horizon of the second peer, discovering new peers, groups, and services.
- Discovery via rendezvous points. A rendezvous point is a special peer that keeps information about the peers it knows about. A peer that can communicate via a rendezvous peer, perhaps via a pipe, can learn of the existence of other peers. Rendezvous points are especially helpful to an isolated peer by quickly seeding it with lots of information. It is conceivable that some web sites or its equivalent will be devoted to providing information of well-known rendezvous points.

3.3.6 A comparison of existing service discovery techniques

You can roughly divide the service discovery process into two steps. First the existence of the actual services has to be discovered on the network, and correspondingly the services have to advertise themselves in some way. The next step is a selection process where the "most appropriate" service(s) are distinguished from the rest of the previously discovered services. In some of the protocols above the two steps are separated, whereas in others the two steps are intertwined from the perspective of the services performing the discovery.

The different protocols use different strategies to achieve the discovery and selection functions. For the discovery part the strategies can be categorized as follows:

- Local peer to peer: The services broadcast information about themselves directly to other services on the network. (UPnP, JXTA)
- Dynamic local registries: The services announce their presence to a local registry. In order to discover a service the registry must first be discovered, automatically or by explicitly pointing it out. (Jini, SLP, Salutation, UPnP, JXTA)
- Structured static directories: Information about services is manually incorporated into a tree like structure, which might correspond to e.g. an organisational structure. The discovery consists of searches on the tree components or on the service descriptions (LDAP)
- Global registry: All services are manually registered in a shared global registry, an approach that essentially removes the need for a separate discovery step. (UDDI)

The peer to peer approach is nice in the sense that it requires no separate infrastructure components, making it potentially more robust than approaches with single points of failure. On the other hand it puts a lot of computational overhead on the services, and especially when scaling up the number of available services.

By introducing an infrastructure entity that knows about the different available services and can handle queries and make selections on that set of services, most of the computational effort can be moved to the infrastructure. In the case with a dynamic local registry, the services on the network are mainly discovered using multicast. The centralized registry could then also work as a bridge between networks. The centralized approach also makes it possible to make the selection part of the discovery more complex, in the sense that the descriptions of the services can be richer, allowing for more complex queries by services using the infrastructure. In most cases however, the different protocols above only use different kinds of simple template matching in order to spare the services from computational overhead. The matching is mainly performed with regard to the following aspects:

- Compatibility: A powerful and commonly used selection strategy is to only search for service that you know that you will be able to use. This can be achieved by some interface or object matching (Jini, JXTA) or via a description of how the service is contacted and used (UDDI)
- Service attributes: More general descriptions of the services can also be used as a factor to distinguish one service from another. Such descriptions could include e.g. a service type and different kinds of attributes describing the functionality of the service. (Salutation, SLP, LDAP)
- Contextual factors: Sometimes descriptions of the services themselves are not enough to make a proper selection. Thus external information about organizational or physical context could be used. (LDAP, UDDI)

A final distinction between the different discovery mechanisms concerns the balance between discovery and selection. If the number of discovered services is small, then the selection process will be easier and the service descriptions can be less extensive. When using global directories, like in the case with UDDI, the service descriptions has to be very detailed in order to enable distinctions between the numerous services.

Thus by utilizing a smart discovery function, or by clustering the services in a clever way, the service selection part could be reduced, and the service discovery part as a whole could be simplified.

3.3.7 The addressed problem

As have been shown above, there are already several technologies to support service discovery. These approaches however suffer from a number of problems. One problem is that the delimiting factors determining which services you are able to find is to imprecise. A typical example of this is technologies like Jini or UPnP that uses properties of the local network as delimiting factors. Network boundaries might however map very badly to the needs of the querying application. This might result in resulting sets of services that are either to big and cumbersome or to small, lacking services that are potentially interesting.

Other problems exist in technologies with global registries like the web services. For these technologies, the large available sets of services make the selection process cumbersome, thus putting high demands on the querying services. The question that is addressed here thus becomes:

How can you design technologies for service discovery in ubiquitous service environments that in a *realizable* and *efficient* way will provide *relevant* sets of services given the needs of the querying application?

One way to achieve a smarter discovery mechanism is to utilize different kinds of context information in the discovery process. This feature is to some extent utilized by some of the protocols above. Both LDAP and UDDI place the services in an organisational context making it possible to make selections that are not solely based on the intrinsic properties of the services. In Jini it is possible to encode information about the user and location of the service into a service proxy object. These methods use the context information as an aid in the selection part of the discovery process. A disadvantage with this approach is that by adding context information to the service descriptions, the querying services must have some knowledge about these organisational or other contextual entities in order to utilize them.

An alternative approach is to use the contextual factors to delimit the discovery part of the process. The crucial problem with this approach is how to decide what services that are relevant and meaningful in a specific context and how to exclude services that are not interesting. This immediately raises the question of what the properties could be that makes a service relevant to another service. One such property, which has been used quite frequently in context aware applications, is proximity (Starner et al.. 1997; José et al. 1999). It is likely that two compatible services that are close to each other could "interact" in a meaningful way (Gustafsson and Jonsson 1999; Pham et al. 2000).

The absolutely most common example of proximity based service discovery, implemented by several of the discovery protocols described above, can be described in terms of network proximity. Protocols such as Jini and UPnP can chose to limit the service discovery to cover only services that reside on the local area network. An approach that might be quite sufficient when only looking for local services and when the number of available services on the network is quite small.

Sometimes, using the network structure as delimiter might be a too blunt tool. A more precise factor that has been explored to some extent is physical proximity, enabling service discovery that only discovers services in the physical vicinity. One example of a discovery mechanism that uses physical proximity is the radio based Bluetooth service browser, Bluetooth includes its own service discovery protocol that locates services offered by devices within the radio range of a user's Bluetooth device, where the range is normally limited to around 10m.

Discovery based on network proximity or physical proximity is typically useful when a person with some mobile computing equipment enters a new and unknown environment, and has to get access to some public resources in that environment. One can however imagine cases when other factors than proximity might be more suitable delimiters for the service discovery. One such factor that can be considered to be important is personal association. Many of the services in use have an owner. This might be the software on your personal computing artefacts such as laptops or mobile phones, or it might be some agent based service residing on a server, maybe collecting information for you on the Internet. It seems reasonable to assume that these services that share owner might benefit from being able to discover each other. Other useful

properties that one could use in order to create meaningful collections of services could concern social or organizational relations, such as project membership etc. An interesting approach for service discovery would thus be to extend the network and physical proximity based protocols used today to also handle collections of services clustered by other contextual factors.

A crucial issue when creating open service environments like USEs is what knowledge the services have about other services and especially knowledge regarding how to use them. If one assumes full knowledge, each service component knows the communication protocol (RMI, Corba, SOAP, etc.) as well as the exact API of the services it wants to use, and also knows exactly what the service does, e.g. it can differentiate between two services with the same API. In the other end of the spectrum the services know nothing of each other, and has to figure out both how to use the other services as well as what the services actually does. The possibly most fruitful way might reside somewhere in between these two approaches, allowing the services to be *loosely coupled*. One way to achieve a loose coupling between services is to assume that the services not only share communication protocol but also shares a set of simple standard interfaces each describing a function in a rather general way, such as that it can consume a certain MIME type. The interfaces could also be slightly more specific such as e.g. a file viewer interface that can receive any file and try do display it to a user in an arbitrary way, or a messaging interface, that can receive a piece of text and render it to the user in some way, visually or by audio.

These general interfaces can then be combined with dynamically generated metadata describing the functionality of the service and the context in which the service resides. This metadata could concern type and attribute descriptions of the service functionality or contextual factors such as in which room the service currently resides, and whom it belongs to. This approach makes it possible to perform service discovery on several different levels. Simple compatibility matches for simple services, which then have to contain no functions for reasoning about metadata, as well as more complex service discovery mechanisms that use the provided metadata to make a more thorough analysis of the available services. Since the metadata could describe not only static properties of the services but also external dynamic context information, this raises a need for an external system that can collect and provide this kind of context information to applications.

3.3.8 Crucial properties

We can identify a number of important properties that a system that implements context aware service discovery should encompass in order to become practically realizable. One crucial key to success is whether the service discovery technology ca interplay with existing service oriented technologies. Therefore, criteria for evaluating new approaches for context aware service discovery have to include aspects related to *realizability*. So except that the proposed designs will have to provide the sought service discovery behavior it will also have to adhere to the following requirements:

- *Service adjustment requirements:* In an open service environment, different services might be implemented on different service architectures. For a service discovery protocol to be successful it will have to coexist and integrate well

with existing service architectures, so that only smaller modifications is needed to incorporate the service discovery functionality on existing services.

- *Service capability requirements*: Selecting an appropriate service from a large set is a potentially complex task. A service discovery mechanism might support this selection process to different degrees. Minimizing the need for services to be able to perform this selection makes it possible to achieve a dynamic composition behavior also using simpler services. There is however a tradeoff between simplicity at one hand and precision and control on the other.
- *Deployment overhead requirements*: Enabling context aware service discovery in most cases requires services to be enhanced with some metadata in order to become discoverable. The overhead caused by this deployment process might highly affect the success of the service discovery mechanism.

3.3.9 Approaches to achieve context dependent service discovery in ubiquitous service environments

Context aware service discovery using meaningful service collections

In the Context shadow system each service announces its existence to a context server that represents some entity in the world, such as a location or a group of people. By using sensor information these entities are dynamically interlinked, creating a searchable web of service information.

The service discovery in Context Shadow is based on 1) a query interface to acquire sets on services based on context information, and 2) a simple key value match to distinguish specific services. Several applications were created that demonstrates the behaviour of the system.

Dynamic bootstrapping with unknown service infrastructures

The way services announced themselves in the previous system, context shadow was a bit unpractical, since each service had to act as a beacon against one or several context servers. In the ACAS architecture on the other hand, service information is compiled in larger sets, each representing a *service infrastructure*, where a service infrastructure could be something like an organization or a department etc. which might have its own system management group that could be responsible for maintaining these service collections.

A new meta service description format has been developed, that acts as an umbrella description of services implemented using different technologies, also providing means to add context description elements to the service description.

Pointers to the service descriptions are provided through the context information infrastructure in the form of context elements. The service description is also made available through a service announcer service, through which you make queries against the contextually enhanced service descriptions.

Collaborative service deployment

Both the previous approaches to achieve dynamic service composition rely on that the environments are enhanced both with software mechanisms to support the service discovery, as well as with sensors, to enable the context enhanced discovery. With the Spots system, the idea is instead that the instrumentation of the environments becomes a collaborative activity. By defining places using detectable patterns already present in the environment, such as radio strength patterns from WiFi access points, the deployment and maintenance activities becomes decoupled from the physical and organizational environment. Several prototypes have been implemented that shows how users themselves can instrument the environments they inhabit. By adding machine interpretable tags at specific locations, software services can be triggered to behave in a certain manner at that location, and thus in a way instrument that place for that person. Information regarding what services that is available in each defined place can also be added in a collaborative fashion. Places defined by a user can then be shared with other users via a web of community servers. Dynamic service composition is achieved by adding simple service descriptions in the form of key-value labels, at a defined place. Another service can then subscribe for specific labels that are present at a user's current location.

A proof of concept application has been built that allows a user to automatically get access to shared displays and other computers in the vicinity, when entering an environment.

A smaller user study has also been executed in which students used the Spots system as an enabling technology to create their own pervasive technology applications. This study showed that the system was able to support a range of different application scenarios containing dynamic service composition in different forms.

3.4 Revisiting the user's role in context information middleware

In many previous attempts to design support systems for context awareness the users have been given a fairly passive role, an approach that in many cases will become problematic. In software system design there is normally a sharp line drawn between the developers of the system on one hand and the users of the system on the other. The role of the user is typically understood in terms of someone that is using the functionalities of a system in accordance with the intensions of the designer. The designer on the other hand is expected to anticipate the needs of the user and provide functionalities accordingly. In cases where users are involved it is mostly during the design process, for example by various forms of participatory design methodologies.

In this chapter a number of problems will be identified that can be mitigated by extending the role and responsibilities of the user with respect to the system.

3.4.1 User appropriation

The notion of appropriation can be described as the process in which users of an artefact put that artefact into use in ways that are meaningful for them, given their experiences and existing practices. Appropriation is often mentioned in explainations regarding how people learn to use new tools and artefacts. In such learning situations certain behaviours can initially be mimicked without the learner having a full understanding of what he is doing. It is only after an appropriation process that the usage becomes entirely meaningful and for example can be generalised to other situations than the actual learning situation. The artefacts that can be appropriated in these settings range from intellectual artefacts such as new languages or mathematical principles to more tangible artefacts such as computing devices or software applications (Säljö 2000).

Often when appropriation is mentioned within the field of HCI what is referred to is a special case where users put technology into use in ways that goes beyond the original design intention (Höök 2006). This kind of appropriation is sometimes perceived as being problematic and as a failure of the designer to foresee the actual needs of the user. In many cases this process is however described in terms of success stories where technologies designed for small user groups and specific situations are being adopted by a much wider user groups and in a much broader set of situations. An often mentioned example of this is the SMS technology which was initially designed as a tool for busy businessmen to exchange information, but that is now broadly used primarily by teenagers in various social settings.

A question that is interesting to pose is if it is possible to support appropriation through system design. It have been claimed that appropriation can be supported by giving the design a certain degree of "openness" (Dourish 2004; Höök 2006). Exactly how to achieve this openness is still open for discussion, but a starting point could be to examine existing information technologies that to various extents have reached success by being adapted to fit several usage areas:

- **SMS:** used for everything from advertising to dating to collective political action
- **email:** can also be used for note-taking, to-do's, scheduling etc.
- **http protocol:** simple connection protocol that has been used for a wide range of applications
- **Wikis:** open web communities that allows for a high degree of adaptation.
- **Blogs:** simple content management,
- **Spreadsheets:** simple tables adapted to many uses.

What is interesting to see when examining these systems is that appropriation can be performed not only by end users (as e.g. in the SMS case), but also by *software developers* (as e.g. with the http protocol). This form of appropriation when a programmer chooses to tweak a technology to fit for example a new application scenario can be referred to as *hacker style adaptation*. In chapter 7 we will come back to this notion as we

examine how the distinction between user and designer of the system are blurred in adaptable and end user programming systems.

An overall goal with respect to design for appropriation is however to create designs that allows for adaptation to users' actual needs and work practices, which supposedly will lead to increased user participation and system usage.

3.4.2 Appropriation and context aware systems

As have been pointed out by for example Höök (Höök 2006) the usages of applications in the new ubiquitous computing environments become tightly interweaved with our everyday social life. Mobile artifacts and applications are for example carried around and used in several different social contexts such as various work situations or settings with family or friends. This broad range of usage situations makes it hard for application designers to foresee all possible use cases, increasing the need for openness that allows for a certain degree of appropriation.

These issues should always be addressed when designing applications for ubiquitous computing environments but becomes especially problematic in context aware applications due to the fact that many of these systems contain internal representations and assumptions about the situations of users as well as about various preferences. Problems that might appear in such situations are that the designer of the system has interpreted the sensor data in a different way than the end user would do. Dourish states that the meaning making in context aware applications should be performed by the end users and not in advance by the designer of the system (Dourish 2004):

> "By turning our attention from "context" (as a set of descriptive features of settings) to "practice" (forms of engagement with those settings), we assigned a central role to the meanings that people find in the world and the meanings of their actions there in terms of the consequences and interpretations of those actions for themselves and for others. The important point, however, is that we now see those meanings as essentially open-ended; we recognize that part of what people are doing when they adopt and adapt technologies, incorporating them into their own work, is creating and communicating new meanings though those technologies as their working practices evolve. The broad principle that these examples illustrate is that users, not designers, determine the meaning of the technologies that they use, through the ways in which they incorporate them into practice. Accordingly, the focus of the design is not simply "how can people get their work done," but "how can people create their own meanings and uses for the system in use"; and in turn, this suggests an open approach in which users are active participants in the emergence of ways of working." (Dourish 2004, pp. 27)

These issues are relevant to all designs of context aware applications but are particularly important when designing context information middleware. An important feature of a middleware is that it should be versatile and generic in nature. Issues of how the system will behave in new and unforeseen contexts of use thus become crucial.

Another issue that one has to take into consideration when designing context information middleware is that they will be running during longer periods of time, and thus must be able to handle both changes in usage patterns and additions of new applications. Consider for example the following scenario describing the changing nature of a work process at an imagined company:

At one point in time it is decided that weekly briefing meetings should take place every Monday morning. In the first weeks these meetings take place as initially intended. Various topics is presented and discussed by the participants while some breakfast is being served. After a few weeks the character of the meetings is gradually starting to change, becoming more and more informal in nature until finally the main focus of the activity is the breakfast and the social chitchat.

Now imagine how this situation is perceived from a context aware system. Most measurable factors of the initial more formal meeting situation and the later informal one, stays the same. The same people are still gathering at the same time and place. They are speaking approximately as much and are still eating and drinking. The rules or preferences that were set up with the first situation in mind might however not be applicable any more since the way that the situation is perceived by the participants has changed so radically.

This problem can also be addressed by revisiting the role of the user. A comparison can be made with how operating systems for personal computers are used. An operating system is to a large extent managed by end users. The system can be tweaked and tuned by the user in various ways in order to keep it updated with the user's actual practices. An example of this is the structure of folders used to organize documents. If practices change, the structure can be changed. Applications can be added and removed according to the needs of the users. One can imagine that the rules and models governing the behavior in a context aware system could be available to users in a similar fashion.

3.4.3 Adaptability and adaptivity

In the best of worlds, context aware computer systems should be hidden away and behave in accordance with the users' wishes. In any more realistic scenario the middleware will from time to time have to be adjusted in different ways. The adaptive system must thus also be adaptable.

Adaptability of a middleware system can take place on different levels. At one level end users can affect the system in order to get it to behave in accordance with the user's preferences. This is typically done by changing some parameters in a graphical user interface. Means to support adaptability can also be provided to system developers, to for example to extend the system to new application domains etc. For this group of users the tools provided to adapt the system could be for example open programming APIs and protocols or sharing of code through open source etc.

There is however a zone in between these two approaches for adaptation, where end users can take on some of the roles of system developers. One example of how this can be achieved is through the notion of end user programming. Simple examples of end user programming in other domains are for example activities of creating filters

for filing email or simple programming by demonstration tasks like macros in spreadsheet applications (Lieberman 2001).

One example of a context aware system that uses end-user programming to empower the users is the aCAPpella system (Dey et al. 2004), which encompasses a paradigm called programming by demonstration. In aCAPpella the user demonstrates desired system behaviours by carrying out actions manually. By using machine learning techniques the system creates recognizers that can detect the situations in which the actions should occur. Once the system has been trained, it is able to carry out actions automatically without prompting the user.

3.4.4 Ensuring control over information

Another area where a too passive user role can cause problems concerns the ownership and management of context information. The context information in a context aware system can be extracted from a number of different sources. Mobile devices can provide for example location information from GSM phones, WiFi equipped mobile devices, GPS devices etc. Other kind of context information can be provided from the environments the users are dwelling in, for example from card readers in office environments for access control, cameras, etc.

Since the context information might have so many different sources, and potential applications using the context information also might be situated in various different places, it is not obvious where to place different system components in a context information middleware system. Could you rely on that the data providers provides the functionality that is required? Or can you put the majority of the functionality on the device running the context aware application?

To examine this question it might be interesting to look at some existing approaches. Even though there are not yet that many context aware applications available as commercial products, there are a few exceptions. One piece of context information that is possible to acquire today and that some companies actually provide is location information based on GSM phone localization. Today's telecom networks (cellular base stations) have a pervasive coverage, and are serving a vast amount of users. This technology has the potential to provide mobile phone users with location detection capability based on information regarding nearby base stations. Some telecom service providers already offer commercial services of this kind, like the Find Friends provided by AT&T in U.S., and Friend Finder by TeliaSonera in Sweden. These services allow a user's friends to gain an awareness of her current location by simply sending and receiving Short Message Service (SMS) messages or by logging in to a website. To protect the user's location privacy, each person has to be authorized by the phone owner beforehand in order to get access to that person's location information. The user can also choose to terminate a buddy's location access at any time. The Find Friends service will further notify the user each time her location is queried by others. These services provide a simple means for a user to share her current location with other people. There is however a number of problems related to how these systems are designed.

In addition to the cost and the burdensome actions of sending and reading multiple SMS messages, the user is very limited with respect to how location information is

represented and shared. Firstly, the location information is derived by a third party service, which may not allow users to define places in the person's own terms, such as "home", "office" etc. Secondly, it is not possible to define access policies towards different requestors: e.g., my location can only be seen by colleagues during my working hours (9:00–17:00, Monday to Friday), while family access is granted all the time; and neither do these services support presenting user location in different granularity towards different requesters, e.g., presenting location in city scale to strangers but in street scale to close friends and family members. Thirdly, these services are generally restricted to a specific operator's network, something that heavily restricts the ways in which the location information could be used. The dissemination of location information is for example restricted to customers of the companies for the described services. Being dependent on a third party service provider also means that the service could be changed at any time or even be removed, like the AT&T Find Friend service. Finally, the potential threat to the user's location privacy still remains, similarly to other infrastructure-based location sensing solutions.

The telecom companies that sits on this context information of course wants to make a profit and will therefore not give it away for free. There is however other plausible business models where the telecom operators only sells the actual location information and lets other parties provide services based on it. Another option for this scenario is to acquire the context information from the mobile phone directly, as have been done for example by (Wei and Jonsson 2006; Hightower et. al 2006) and thus entirely avoid the telecom operators.

This application scenario highlights some important aspects with respect to the ownership and control of context information. General problems with having external parties managing context information include:

Privacy issues: You have to trust the party not to share information with other parties.

Access limitations: There might be restrictions regarding with whom information can be shared.

Application limitations: There might be restrictions on which applications that have access to the data

No control over format: The context information extracted from the provider might be formalized in a way that won't fit the user's needs.

Based on this discussion we can conclude that it is interesting to examine user managed approaches to context information management.

3.4.5 The addressed problem areas

1. How can context information middleware be designed to increase the users' control over how context information concerning them is disseminated and represented?
2. How can context information middleware be designed to increase the users possibilities to assign their own meanings to context information.

3. How can you provide means for adaptation of context information middleware to handle changes in usage patterns and additions of new applications?

3.4.6 Design approaches

User controlled context information management

Choosing a user centric system design in which the user is the owner and to some extent also the administrator of his/her part of the system is one way to handle problems related to ownership and maintenance. By designing a system where the user has increased control over the context information, privacy intrusion issues can be avoided or mitigated, for example by being able to control both with whom to share the information but also to control the format of the context information. The overall goal with the increased empowerment is thus to create designs that allows for adaptation to users' actual needs and work practices, which we suppose will lead to increased user participation and system usage.

Context information might be assembled from sources like your mobile phone, the usage of your computer or by specialized sensor equipment. Some of this information like the position of your mobile phone is currently owned by a company and without control from the user. By trying to collect this information in a system that is owned and maintained by the user, the user gets more control over which information to share and with whom. By having the user "own" part of the system, deciding where to deploy it and being able to shut it down at any time is a key feature to ensure user control. All context information related to a person could then be assembled in this "personal context system", where decisions are made regarding with whom to share the information etc.

Both in the Context Shadow system and in the ACAS system, each user has her own context repository, containing the context information related to that person. In the Context Shadow system, the context server consists only of a small java process that easily can run as a background process on a personal computer. The notion of a personal context system in the ACAS architecture some contains further elements. Apart from a context repository that acts as a container of context information, there is also a communication server that handles subscriptions of incoming context information, as well as the outgoing communication to application subscribing for the information.

User community driven context interpretation

A danger when designing context aware systems is to include too much semantics into the underlying structures and context models. If it is something that humans are really good at and machines are equally bad at it is to extract meaning from unstructured contextual information. How can we create context aware systems that don't rely on fixed predefined models to determine how sensor information should be interpreted and what actions that should be triggered? One solution to this is to allow users to do some of the interpretation of sensor information as well as some of the modelling work, assigning meaning to the available sensor information

Instead of having a context model describing how sensor data should be interpreted, this process can be delegated to the users of the system. In the Spots system places are defined by users, who also define how to name and describe these places. A friend finder application was created in which users could present their current location to others using their own naming for the places they visited.

Supporting adaptability and end user programming

Another form of user empowerment is to engage the user in contributing to the system's functionality. By providing means to modify or add new functionalities to the system in a simple way, the user can adapt the system so that it better supports her specific needs and requirements. In this way the user can become motivated to contribute to the system deployment and maintenance, issues that are normally problematic in context aware systems.

By making the context information middleware more visible, users might get a better understanding of the system functionality. One important aspect of this is to provide user interfaces through which the users can adjust features of the system.

Chapter 4
Three middleware designs supporting context aware computing

In the previous chapter a range of motivations for the need of middleware support for context aware computing was presented. In this chapter three different designs of context information middleware (CIM) systems will be presented in further detail, together with some example applications.

4.1 CIM design I: The Context Shadow system - Supporting context dependent service discovery with interlinked context servers

The Context Shadow system is a context information middleware system which is mainly targeted at providing context dependent service discovery functionalities. The system provides a simple query interface for applications to get information about for example local services.

In the system, software services are tied to entities such as locations, users or groups. Sensor information is used to create dynamic links between for example representations of users and locations. In this way a searchable web is created which makes it possible for applications to make queries regarding the physical and computational context of the different entities in the system.

The general aim of the Context Shadow system is to provide a "shadow of the real world" to software services. By using a simple query API it is possible for services to

acquire important context information such as information concerning local artefacts, services and people. More specifically, the system provides:

- Support for context aware service discovery.
- Organization of services and context information in meaningful collections.
- Context information for applications derived from sensors and other services
- Support for refinement of context information.

The Context Shadow system is based on a blackboard architecture where certain stable entities are represented as context servers. These servers act as repositories for context information related to that entity. Typical entities that can be provided with a context server are persons, locations and groups/projects. A context server contains two types of data; context information concerning the entity that the server represents and links to other context servers.

Sensors and applications that provide context information are provided with a simple communication interface which allows them to post their information to one or several specified context servers. The context servers are implemented using TSpaces from IBM (Wyckoff 1998). TSpaces can be described as a network communication buffer with database capabilities implemented as a tuplespace. TSpaces provides a simple and robust network interface as well as advanced querying capabilities.

4.1.1 Cross referencing

A key feature of the system is the possibility to establish links or relations between different context servers. A typical example of this is the detection of a person entering a room. If either the person detects the room or the room detects the person this results in that a cross reference is established between the local and personal context server. This can be done since the location sensors provide references to a location context server, or in the case of person detection, the person sensor receives references to personal context servers.

The context servers and the links between them create a searchable web where the topology changes dynamically. Information about the users' current context does not only consist of pieces of information in the context servers, but is also embedded in the topology of the surrounding web of context servers.

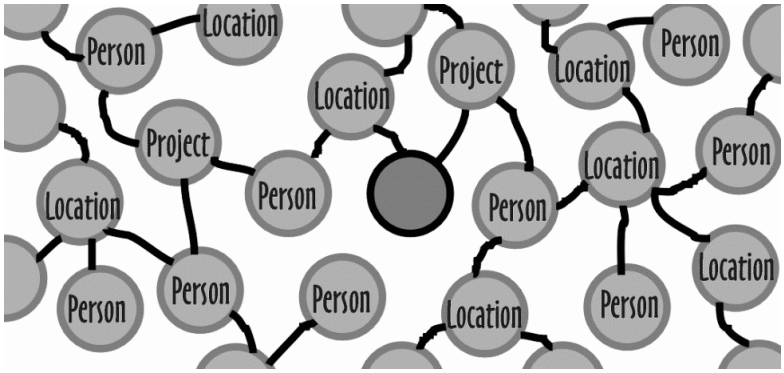


Figure 5: The linked context servers create a searchable space, where the topology of the space is part of the context information.

More static references can also be established. Examples of this can for example be references describing the relation between locations. There is a general problem regarding how to represent location in context aware applications. In Context Shadow there is no structured way of describing locations in terms of hierarchies and distances. Instead you define "places" in an arbitrary way, and then create relations between these places. In this way it is possible to create hierarchies when needed, but there is no requirement for developers to provide a complete or coherent location model. Another example of references of a more static nature could be references between persons and projects. By connecting people to each other via a project entity, it is possible to create CSCW tools that e.g. could have knowledge about meeting history, shared documents etc.

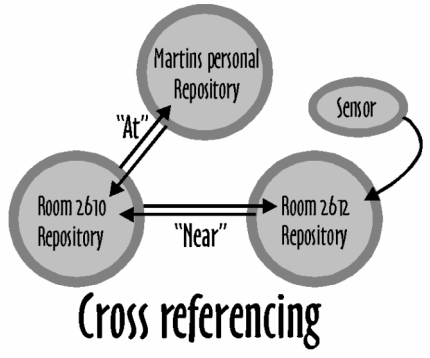


Figure 6.The context servers are linked with references. By following the links it is possible to acquire context information from other context servers than your starting point.

4.1.2 Example applications

Three implemented prototypes will be described that illustrates the functionality of the

Context Shadow: A messenger service that uses Context Shadow to find public viewer services, a tool for local teamwork which uses information about the location and information about people in the room to provide support for collaboration, and finally an active document service that support document management by using context information.

Messenger service

With this prototype we wanted to examine how public resources can be used to send a message to a person. In the prototype an active messenger service actively tries to find resources near the receiver of the message that can receive the message and display it to the person. The prototype was used in the following scenario:

1. An active messenger service with a message to a person has migrated to that person's laptop. This computer however lacks output resources that the service can use to display the message.

2a. The person enters a room with a public wall display. The messenger service discovers this output resource and chooses to display its message on this resource.

2b. The person enters a room where another person sits with his personal computer. This computer has a public speaker resource. The messenger service discovers the speaker resource and chooses to play an audio version of the message through the speakers.

The person's entrance in a room is registered by a camera attached to that person's laptop, where the camera reads and recognizes a barcode-like symbol in the ceiling of the room. The identified symbol is then translated to a reference to the Context Server representing that room, whereby the sensor service on the laptop establishes a cross reference between the context servers representing the room and the person.

The active messenger service regularly queries Context Shadow for appropriate display services. First a query is sent that asks for services at the person's current location. This query propagates to other Context Servers using the dynamically established references. In this case it follows the newly established reference to the Context Server for the room. If no appropriate services are available there, the query propagates to Context Servers owned by other persons present in the room, thereby also covering services owned by those persons. The existing audio and image presentation services constantly announce their presence to the Context Servers that they are connected to by beaconing descriptions of themselves. The implemented services were simple wrappers around a standalone HTML browser and a sound player application that would receive URLs that they would open using the standalone applications. So when the messaging service finds one of the service descriptions it interprets it and then uses that service to present its message.

Tools for local collaboration

There tend to be more and more computer artefacts present at meetings, both personal mobile devices such as laptops and PDA's as well as stationary devices such as projectors. These artefacts however often create more problems than is of aid. One irritating obstacle is problems with information exchange; a paper you can just reach across the table, but sharing a document on a computer is often more problematic.

What we did to solve these problems was to create a set of services to enable a seamless flow of information between computers in a room. The prototype, called fuseONE

(Werle et al. 2001), is an example of a ubiquitous service environment, consisting of a large number of standalone components. All the services are implemented in Java and uses Jini technology for service lookup and remote method invocation. One set of components consists of very simple but useful services that can receive and display documents of different kinds on the computer where the services reside. What these services actually do is that they accept arbitrary remote files and instruct the current operating system to launch the application with which the file's type is associated and then open the file. These services are numerous and reside both on personal and public devices.

Another component is a context sensitive desktop, which makes the services described above and other services accessible to users via a GUI. This service queries Context Shadow about which other services that are relevant to its user's context, and then filters out the irrelevant ones from the desktop. More specifically the context sensitive desktop queries its owner's Context Server for (in following order) personal services, local services and services owned by other persons in the room. Whenever a service is found, a service description object is returned to the browser application. The application has already found all Jini services in the network by using the lookup function that comes with Jini, and uses the service description from Context Shadow to filter out services that are not available in the actual room.

In this prototype a person identification sensor was used to create the references between location and personal context servers. The actual sensor is a Dallas Semiconductors iButton (Dallas Semiconductors 2002), a small button-like computer memory that the persons actively have to press into a receptor in the room to announce their presence. The button actually contains the URL reference to its owner's Context Server. What happens when the button is placed within the receptor is that the reference is being put into the Context Server for the room, and a cross-reference is automatically posted into the person's personal Context Server.

Active Documents

Another type of service that was developed is the Active Document service. The idea of Active Documents is to take off from the agent-programming paradigm, and turn documents into autonomous mobile agents and by that give them some useful qualities. A document should, for example, be aware of its content and the intention with it, and be aware of the context it is operating in, e.g. its receivers (who, why, preferences about formats, physical surroundings, etc.). The documents are active in the sense that they are autonomous (act independently), reactive (react on changes in the environment), and proactive (have their own goals and plans). One of the main ideas is that the documents actively should participate in the work and thereby support the work process. In this prototype the Active Document service can identify when a certain project has gathered for a meeting, and then actively display information that it has stored from earlier meetings. The service uses information from Context Shadow about project membership, the users' locations, what people are in the room and what services that are available in the room.

In more detail, the Active Document uses Context Shadow for monitoring people that are members of the same project as the document. Based on the information given by Context Shadow, the Active Document tries to find locations where at least two

project members are present for the moment. If it finds such a location, the Active Document assumes that there is a project meeting taking place! The Active Document now tries to migrate to that location by asking Context Shadow for the nearest execution environment available. When it has migrated to a suitable host where it can execute, it tells Context Shadow that it has entered the room. As the document appear for the Context Shadow as a person, which for example means that it has its own context server, the document announces its presence within the room by telling the Context Shadow that it has entered the room (even if that not always is true in a physical sense) just as what happens when a person put his iButton into a receptor inside the room. This mean for example that Context Shadow will include the Active Document when the context sensitive desktop described above asks for relevant services to show. Now the Active Document asks Context Shadow for an appropriate public display resource inside the room. If such a service is available, the document utilizes that public service to display itself on. If someone clicks on the icon representing the Active Document service on the desktop, the document is notified about who has clicked. The Active Document then asks Context Shadow for a suitable resource to display itself on for that user. Of course, that resource does not have to be available on the same device as the one that the user clicked on.

4.1.3 Related work

The problem of creating an infrastructure of meaningful assemblies of services was earlier taken on in the Cooltown project by HP labs (Caswell and Debaty 2000), where places are provided with a web page. Services that exist in these places can then be accessed via that web page. The system provides several interesting ways to automatically assemble the services at a location and then make them accessible through the web page. The notion of tying services to a location also exists in Context Shadow, with the difference that in Cooltown the ambition is to make services easily accessible directly by users, while Context Shadow has the ambition to provide services mainly to other services. Another support system, which targets the problem of how to support the design of context aware applications, is the Context Toolkit system from Georgia Tech (Dey 2001). This is a middleware system with which you can incorporate sensor data in your applications. The system is built with a widget approach making it possible for applications to incorporate context data about the same way as you incorporate a GUI component. This system does not have an explicit infrastructure approach but rather supports the creation of standalone applications. The Context Toolkit provides much of the same functionality as the Context Shadow, such as context queries and refinement of context data. The major difference from the Context Shadow system is that Context Shadow includes other applications as being part of the context, thus providing support for collaboration between services. The TEA project presents a system architecture as well as a method to support the design of context aware systems (Schmidt and Laerhoven 2001). The system architecture consists of several layers where the bottom layer consists of cues that represent an abstraction of the sensor-data. The cues are then combined into contexts, which can be seen as a high level description of the current situation. These context descriptions can then be fetched by the applications from a tuplespace. The provided method gives step-by-step support for the choice and assembly of sensors as well as for the application development. The architecture from TEA is similar to the Context Toolkit

approach in the sense that they both support the development of standalone applications and that none of them uses shared context servers to represent real world entities. The Interactive Workspace project at Stanford University (Fox et al. 2000) uses a system they call an event heap to enable services in a room to communicate on an event level. The event heap could be compared with the context servers representing locations in the Context Shadow system, with the difference that the context servers are not used for communication between services to the same extent as the event heap. The system also has no support for combining several eventheaps into an infrastructure, nor will the event heap communicate with services that have other properties than being in the room.

4.2 CIM design II: The ACAS architecture - An event based approach with dynamic subscriptions

The ACAS architecture contains functions for subscribing for and transporting context information to applications. The basic idea is that you have separate local infrastructures (companies, buildings, persons etc.) providing both specific services that are relevant to the local activities, as well as sensors providing information about aspects of the local environment. In the ACAS architecture, each local infrastructure is represented by an *infrastructure server* (IS), which through a bootstrap mechanism, dynamically can connect to a *personal context system* (PCS) or to other infrastructures to enable exchange of context information. Key issues for this architecture are e.g. how to setup flows of information from the local infrastructures in an efficient way and how to encode context information in a way that is suitable for a large number of applications.

4.2.1 Gathering context information from public environments

To simplify the management of context acquisition from public environments, we define a Public Service Infrastructure (PSI) as being an instance of a service environment containing both services and sensors. A PSI may be confined to a room, a vehicle, an organization or any other natural or abstract boundary. Although PSIs are not necessarily identical to geo-location spaces it is likely to be conceptually convenient to create mappings between the two. Within the PSI, context data is produced by context generators which are attached to hardware sensors that measure physical properties of the PSI, or software sensors that measure computing properties of the PSI:

The representative of a PSI is called the Communication and Coordination Service (CCS), which manifests the PSI to the Internet as an addressable entity and communicates with Personal Servers for exchanging context information. The CCS subscribes for context data from the context generators in the PSI. When context indicates that a user with a Personal Context System is present within the PSI, the CCS adds the user's PCS as a temporary resource of context and services.

The hardware Sensor Context Generator consists of a set of services; each presents certain context information about the PSI (or a user inside the PSI). Such context information is derived (sensed or inferred) from various physical sensor input, e.g., a Personal Location Service could detect user's presence through a device such as an Active Badge (Want et al.).

The software Sensor Context Generator collects and maintains a fresh status description (including the service metadata) about the end User Services (US) associated with the PSI while the USs refer to the services which cater for user's immediate requirements, such as a media player on a wall-mounted display.

4.2.2 Location based interaction bootstrapping

There are two complimentary ways to initiate an association between a user and a PSI

1. The user presents herself to the PSI through some means of direct or indirect action: a login interface, a broadcast identification or others. The PSI then contacts the PCS.
2. The user detects the PSI and sends a location code or broadcast identification to the PS. The PS then contacts the PSI.

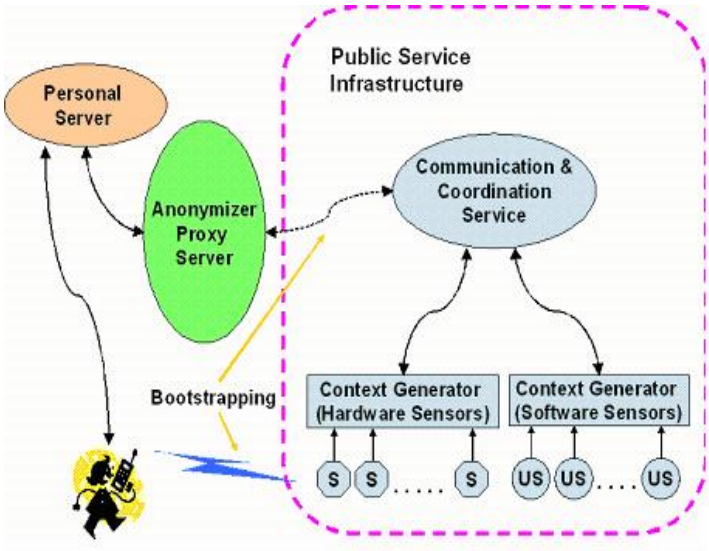


Figure 7. Schematics of the bootstrapping mechanisms in the ACAS infrastructure

Location information has been exploited as a very useful piece of context information as there are many location based systems based on different positioning technologies. These systems are also referred as location based services. In the ACAS architecture the location sensing becomes especially important to achieve the bootstrapping with the Public Service Infrastructures. Two positioning technologies have been evaluated for this purpose: A wireless sensor network based on the Mica Mote sensor platforms, as well as a combination of mobile and stationary Bluetooth enabled devices.

Using Mica Motes for context monitoring and bootstrapping

Wireless sensor networks is a technology that is becoming increasingly popular. The first Motes appeared in the Smart Dust project at Berkeley, and were referred to as

COTS Motes, since they were easily manufactured using standard components. The Smart Dust project was followed by the NEST project in which different kinds of hardware designs has been presented: Rene, Mica and Mica2 Motes. This project has also produced the operating system running on the motes: TinyOS, as well as different kind of additional software and hardware blueprints.

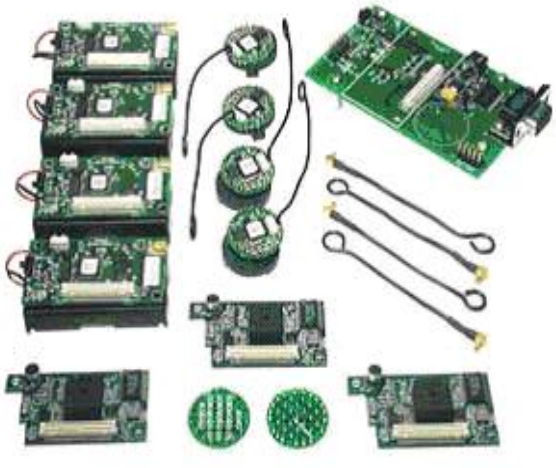


Figure 8. The Mica2 Motes used for the ACAS call forwarding demo

The Mica2 Motes used within this project is manufactured by the company Crossbow, but the hardware design is free and publicly available. In short the Motes consist of a processor and radio module, running the operating system as well handling all the I/O, including the radio communication with other motes. Radio range indoor has been measured to approximately 25 meters with the standard antenna, with thin office walls blocking the way. This platform is easily extended with different kinds of sensor platforms, which can easily be created based on current needs. In this project a prefabricated sensor platform has been used, that contains a number of different sensors: Light, sound (noise, tone detection, raw), temperature, acceleration (2 axis), magnetometer (2 axis). To program the motes, and also to get the readings to a computer, a programming board is used.

The most obvious usage of wireless sensor networks is to use them to monitor certain measurable contextual aspects in a specific area. In this setting the motes are deployed in fixed positions, spread out in a way that they can communicate with one or several other motes. In a ubiquitous computing scenario, the motes could e.g. be used to capture information about the activities in different rooms. The motes could be placed on the inside of every office door, sensing the acceleration when somebody opens or closes a door, the noise in the room, detecting if lights are on or off etc.

For this project the Motes were used both for acquiring context information about the physical environment from the sensors as well to enable the bootstrapping mechanisms described above. The Mica motes and the available software are in no way designed to support positioning of individual motes. In this work several different approaches for positioning have been tried out, with the goal of finding an approach to reliably detect in which room a user (carrying a specific mote) is situated. The fact that the motes should be able to support both the task of positioning users and

simultaneously report sensor information, put some severe restrictions on the system design.

A seemingly simple approach was to adjust the transmission strength of the mobile motes, so that their communication would only be detected by the nearest stationary mote. One could then make the assumption that the mobile mote is occupying the same space as the stationary one. It is possible on the Mica motes, to control the transmission strength from the operating system. In another approach, information about which other motes each mote can ”hear” at any given moment were collected. This information is analyzed over a short period of time in order to detect the most reliable connections, and thus assuming motes to be nearby. The neighbour information is collected from the motes and stored in the database. The data is then statistically analyzed in order to find out which motes that show the least packet loss and is thus assumed to be within shortest range. The Neighbour check approach combined with a proper adjustment of the transmission strength was approximated to provide good enough resolution for the positioning. Another advantage with this approach is that the sensor network simultaneously can be used to collect other environment variables, like temperature or noise. Other attempts have been made to enable positioning using Mica Motes, but these are specialized solutions that will not allow for simultaneous collection of sensor data.

Interaction bootstrapping with Bluetooth

Except from Mica motes, positioning was also achieved using Bluetooth. A detection service was implemented to run on ordinary desktop PCs (connected with a 3COM Bluetooth USB dongle) with BlueZ (a Linux Bluetooth Stack) installed on Redhat Linux 9.0. A nearby user carrying a Bluetooth mobile phone or PDA will be discovered by this service. This discovery process is reasonably fast (less than 2 seconds for 5 devices simultaneously) as we only inquiry the Bluetooth device address (BD ADDR). The detection service will then try to obtain a vCard from the discovered devices using Bluetooth OBEX - a standard protocol for information exchange over Bluetooth supported by most Bluetooth mobile phones and PDAs; and then afterwards extracts a temporary token from the ”home address” field of the vCard file. This token is formatted in terms of a Personal Context System address. With this token, further context data are able to arrive at user’s PCS system from the infrastructure.

This process can be generally regarded as an automatic association bootstrapping between PCS and PSI systems as the PSI may only need to send its reference (like a URL in CoolTown (Caswell and Debaty 2000)) to the PS while PS can choose interesting context data to subscribe to. Another system vCard will then be received by the user (through OBEX as well) from the local infrastructure telling its use of the information from her vCard the first time she enters the environment associated, which is like an SMS send to new users of cellular telephone systems when they roam to a new operator’s network.

4.2.3 SIP and SIP Event Notification

The communication in the ACAS architecture relies heavily on the Session Initiation Protocol (SIP) (Campbell and Rosenberg 2002), which is a general purpose IP-based communication protocol supporting interactive session establishment cross the Internet. It defines a complete process mechanism for establishing distant communication session, which is independent of the underlying transport protocol without dependency on the type of session to be established.

In general, SIP is a text encoded protocol based on elements from the popular Internet protocols like Hyper Text Transport Protocol (HTTP) (Fielding et al. 1999), and the Simple Mail Transport Protocol (SMTP) (Klensin 2001). SIP works on a client-server transaction model similar to HTTP. A SIP client generates a SIP request to the SIP network, and some SIP server responds to it with a SIP response, in which way a communication session is established. SIP also uses Uniform Resource Identifiers for addressing, the form of which resembles the normal email address. Since every SIP message includes enough routing and session status information, each single message can be delivered to its destination without problem. These communication features make SIP an outstanding candidate protocol for asynchronous information transfer.

In addition, SIP has a specific Event Notification Specification (Roach 2002) to address the issue of asynchronous notification of events through a SUBSCRIBE/NOTIFY mechanism. An application (called Subscriber in the specification) has to subscribe to the event (changes) it is interested in, and will then receive notification when the corresponding changes occur. This mechanism is very suitable for information delivery like context information in a distributed fashion. Below we elaborate some details of SIP event specification, which will give deep understand of how it will be used for context information delivery:

- Subscription Duration: Any SIP event subscription has a duration, which would be either explicitly specified present in the SUBSCRIBE request or be implied by the default event type/package used. The determined duration is presented in the ”Expires” header in its OK response message, and the duration should be no longer than the requested period. The Subscriber has to refresh the subscription by resubscribing before the duration expires; otherwise no further notification message will be sent which means the subscription is ended.

- Unsubscribe: The subscriber uses the SUBSCRIBE message with a zero value in the ”Expires” header to unsubscribe the established subscription. The notifier can also cancel the subscription to events by sending a NOTIFY message with a ”SubscriptionState” value of ”terminated”. A similar message also denotes the successful approval of unsubscribe.

- Applicable Scope: The specification indicates that it should not be used as an infrastructure for any major classes of event subscription and notification. For example, the highrate or highvolume event notification generated by GPS coordinates (changing at a rate of 100 times per second) does not match its

initial design principle since it can easily overload the network traffic, particularly for mobile networks.

- Event Package: The event package to send as a notification is not restrictively specified. Rather it is suggested to be defined as relevant separate SIP extensions in conformance to the guidelines specification [36], such as the SIP Presence Event Package extension draft [34]. There is also a templatepackage mechanism defined, which can be applied to any event package.

4.2.4 SIP Presence Frameworks

SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) is an IETF working group focusing on introducing the Instant Messaging and Presence (IMP) service extensions to the SIP network. SIMPLE is committed to leverage some of the work of another IETF working group (Instant Messaging and Presence Protocol (IMPP)) to make sure the interoperability with the legacy services and applications conforming to IMPP. Basically, the work within SIMPLE will be compliant with the requirements specified in the IMPP Specifications: "Instant Messaging / Presence Protocol Requirements" (RFC 2779) and "A Model for Presence and Instant Messaging" (RFC 2778), as well as another work in progress draft "Common Presence and Instant Messaging" (CPIM), developed within the IMPP working group. The proposed SIP presence framework consists of a number of key components:

- Presence User Agent (PUA): A presence user agent manipulates presence information for a presentity, which is a presence entity providing presence information on behalf of subjects such as a user. One presentity may have multiple PUAs like the scenario where a user has many devices, such as mobile phone and laptop. Each of the devices can generate presence information and the PUA pushes them to the presentity's presence agent.

- Presence Agent (PA): A presence agent is a SIP user agent capable of accepting subscriptions, storing subscription state, and generating notifications when there are changes in the presence information held by the corresponding presentity. Thus the PA is the entity providing presence service in the SIP presence framework.

- Watcher: The Watcher is a presence service client consuming the presence information distributed by the presence service. Typically, a Watcher will subscribe to some presentity by sending a request to the corresponding PA representing the presentity, and then wait for the notification after the subscription is granted. There are two kinds of Watchers, called Fetchers and Subscribers. A Fetcher simply requests the current value of some presentity's presence information while a Subscriber requests notification of future changes. A special Fetcher is called Poller, which fetches information on a regular basis.

- Presence Event Package: Both pres URI and SIP URI are supported as address protocol schemes. Presence information is constructed in a series of units called PRESENCE TUPLES conforming to Presence Information Data Format (PIDF) in "Common Profile for Presence" (CPP) framework. Some

criteria are also defined there like the default expiration time for subscription within a presence package is 3600 seconds, and the notification frequency should be lower than once every five seconds.

- Publish and Presence Aggregation: SIMPLE also proposes a new mechanism PUBLISH as a complementary method to SIP for presence information composition simplifying collecting presence information from multiple PUAs into a complete presence view of a presentity fed to a PA. The Event Notification Extension for Collections further proposes a mechanism enabling the PA to aggregate a group of presentities into one single presence entity for subscription. This releases Watchers from the overhead to subscribe to each of them in the group individually. So the aggregation mechanism is supported on both the inbound and the outbound side of the PA as a publish and notification collection respectively.

4.2.5 Context model: Pushing the model to the application layer

In this design approach the amount of semantics encoded in the system in terms of generic context models has been further minimized compared with the Context Shadow system. In the ACAS system the general assumption is that since any sensor data can be represented in an almost infinite number of ways, one should not try to find generic models for context information. The only factors that should determine how context information should be modelled are the specific needs of specific applications. Therefore the rules that are used to transform or refine the sensor data are provided inserted into the middleware in a plug-in fashion. This procedure is described in more detail in chapter five.

4.3 CIM design III: The Spots system - A user managed positioning system

The Spots system targets the problems of setting up and maintaining context aware systems. In the Spots system the users themselves define the locations they need, and may add any meta-data to that spot to be shared with other members of a community.

Why have we not seen more commercially successful context aware systems? We can identify some issues that might become problematic when you want to develop these kinds of systems on a larger scale: Obvious obstacles are related to *costs*, especially concerning hardware installations and system maintenance. Another obstacle concerns the often sought pervasiveness of many ubiquitous computing systems, where the system boundaries often exceed the boundaries of organizations, making questions of *ownership* and *maintenance* problematic. *Privacy* issues have to be dealt with when information about peoples' whereabouts is handled by the system. A final obstacle concerns the balance between on one hand the underlying context models that are supposed to give meaning to sensor data, versus on the other hand, the strive for versatility, making it possible to reuse sensor data for different application purposes. A difficult balance since each proposed context model, with its specific description of

the world, enforces constraints on what conclusions you can draw from the sensor data. In this chapter we present a system design where the obstacles sketched above are tackled using two design approaches:

1. Parasitic computing: By making creative use of existing non sensor equipment, costs related to sensor installation can be avoided. This parasitic approach also makes it possible for the system to grow in accordance with the actual usage of the system, thus minimizing the risks associated with massive hardware installations. The parasitic approach could also include use of "soft sensors" that extracts context information from existing software, such as calendars or instant messaging clients. Similar ideas to the notion of parasitic computing have earlier been presented in for example the Placelab project (LaMarca et al. 2005), where existing WiFi radio patterns were used to calculate the users' geographical position.

2. Empowering the user: By empowering the user we mean that the users should be given increased control over the system, in several aspects: Choosing a user centric system design in which the user is the owner and to some extent also the administrator of his/her part of the system is one way to handle problems related to ownership and maintenance. By designing a system where the user has increased control over the context information, privacy intrusion issues can be avoided or mitigated, for example by being able to control both with whom to share the information but control the format of the context information. The overall goal with the increased empowerment is thus to create designs that allows for adaptation to users' actual needs and work practices, which we suppose will lead to increased user participation and system usage.

4.3.1 System implementation

An implementation of a system named "Spots" that follows the design approaches sketched above is presented in this chapter. The basic functionality of the Spots system is that it makes it possible to detect when a person is present at some predefined place, where users themselves define the places they need. This is done using a minimum of expensive and cumbersome sensor equipment. Instead the system uses existing detectable properties of the surrounding environment, such as available WiFi access points or stationary Bluetooth devices. Information regarding detection of places can then be provided to various applications which may use the information in different ways. A key part of the system is a framework for describing places in terms of Spots, where the basic model itself contains no semantics regarding locations and places. The semantics is instead provided by the users in terms of labels.

The usage of the system is separated into two modes: a *design mode* and a *sense mode*. In design mode, a user uses a mobile computing device to define various places in terms of "Spots", in accordance to his or her needs in order to achieve a certain behaviour of the system. Any number of descriptions of a place can be added to the spot in terns of labels. In sense mode, the system continuously tries to determine whether the user is situated in some spot, and if so, makes information regarding the spot available to applications.

Roughly you can say that the Spots system is an enabling technology for two different types of context aware applications:

1. Applications that provide information about a person's whereabouts to other people. In these applications the ability to use your own descriptions of places makes it possible to control how the information your whereabouts are being perceived by others. A Friend-finder application has been implemented and is presented in this paper.
2. Applications that will adapt their behaviour based on aspects of the user's current location. Adding application-specific labels to Spots is a crude but effective way to achieve context dependent adaptation. An implementation of an application allowing users to control and share information with computers in the same spot demonstrates this function.

Finally, for the above functionality to work there has to be some sensors making definition and detection of Spots possible. A spot detector sensor was implemented that uses WiFi signal strength profiles acquired from ordinary WiFi network cards to distinguish different places, achieving a precision that for example makes it possible to distinguish between two adjacent office cubicles.

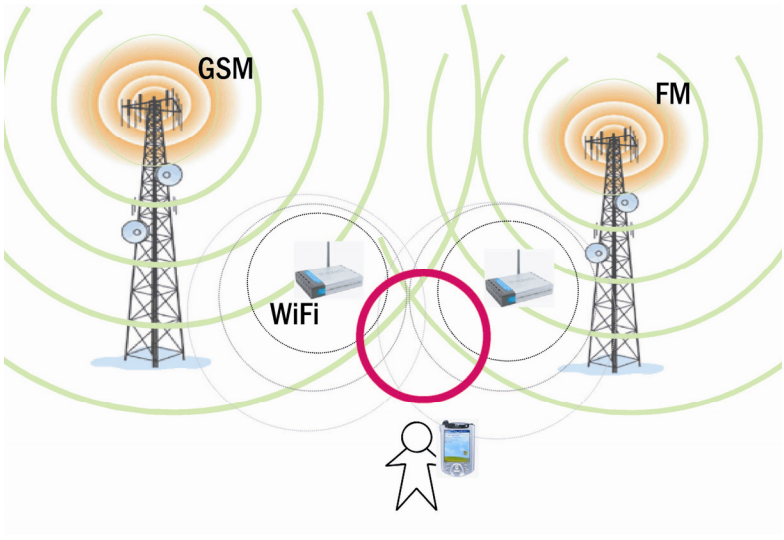


Figure 9. A spot is defined by the specific (radio) patterns which the client can sense in that place

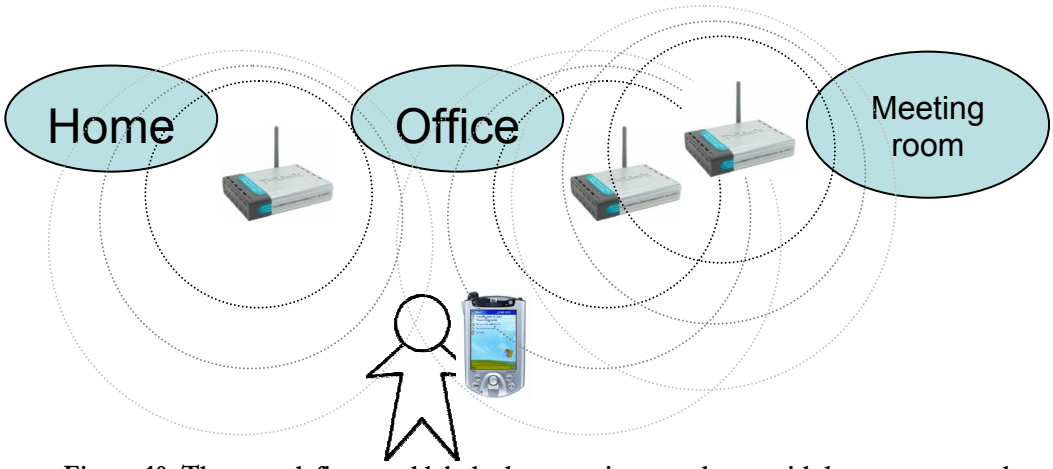


Figure 10. The user defines and labels the spots in accordance with her current needs

Defining places as Spots

To have an unambiguous description of the locations or places that exists in the Spot system, it was decided that describing them as places or locations was not suitable. It should be clear that there is a specific type of location/place that exists which is understood by the system. The idea was that the Spots system can identify Spots and a spot is something that is not just any place, a spot is a specific location/place with associated information that can be found by the sensor application in the Spots system, compared with for example a GPS-based system which can identify any location (at least outdoors). To make a location/place identifiable you need to define a spot at that location. What it actually means to be detected in a spot should be left open. The Spots in the Spots system are an entity representing a location/place that has;

- a unique identifying value.
- a Spot profile
- at least one Key-Value label

Spots can also act as containers of other spots. Every children of that spot will inherit all labels of the parent spot. The containment property is defined using the label entity. The above mentioned entities relate to each other as shown in the following very simple conceptual model.

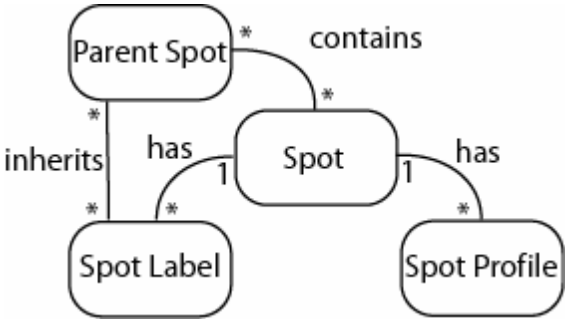


Figure 11. Conceptual location model design

Spot Labels

Spot Label is the user definable context entity that exists in the Spot System. It has the additional responsibility of being able to present itself semantically as a key-value pair to another plug-in application and can thus be defined as follows: *A Spot label in the context model of the Spots system is a specific key-value pair that is associated with a specific spot.* Except from the key-value pair each label also contains information about the *owner* of the spot and information regarding whether the label should be shared with other users or not.

A user is allowed to add any number of labels to a spot, and is also free to define new types of key-value pairs when needed. The system however contains three predefined key-value pairs:

- Name-String Value
- Contains-SpotID
- ContainedBy-SpotID

The containment labels are used to define relations between spots, which in practice mean that a child spot will inherit all labels from the parent spot. Naming of Spots using the Name-key is provided for the users so that they can name the spots they define in a way that is suitable for their current needs with respect to e.g. what application is supposed to use the information or privacy concerns.

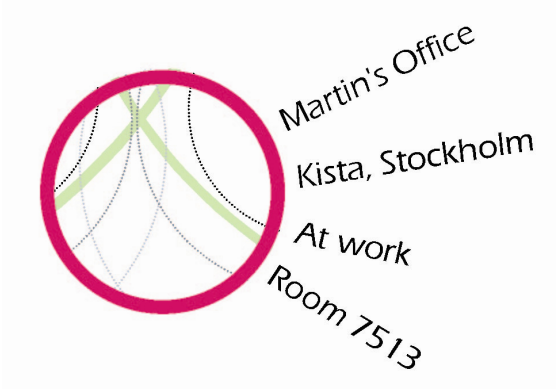


Figure 12. Each spot can be given any number of labels

Spot Profile

The Spot Profile entity encapsulates the detectable properties of a location that is used to separate locations from each other. The responsibility of the Spot Profile entity also includes being able to compare Spots and calculating similarity between Spots. The Spot profile is thus a container of a set of semantic location coordinates that belong to a specific spot. The spot profiles are *sensor dependent,* meaning that a profile created using e.g. a WiFi sensor can not be used to identify the location of e.g. a Bluetooth sensor.

4.3.2 Spots system implementation

This implementation of the Spots system was compiled to test the Spots system idea in a first set of user studies. The implementation uses existing wireless networks (WiFi/802.11) as the means for defining and detecting Spots. The system consists of the following components:

A Spot sensor and designer client, which is a client application with two major functions:

- A graphical user interface allowing users to define Spots and add labels to them
- A sensor mode in which the application runs in the background detecting Spots and passing this information on to other entities.

Client applications were implemented for laptops and iPaq PDAs equipped with WiFi/802.11 wireless LAN capabilities.

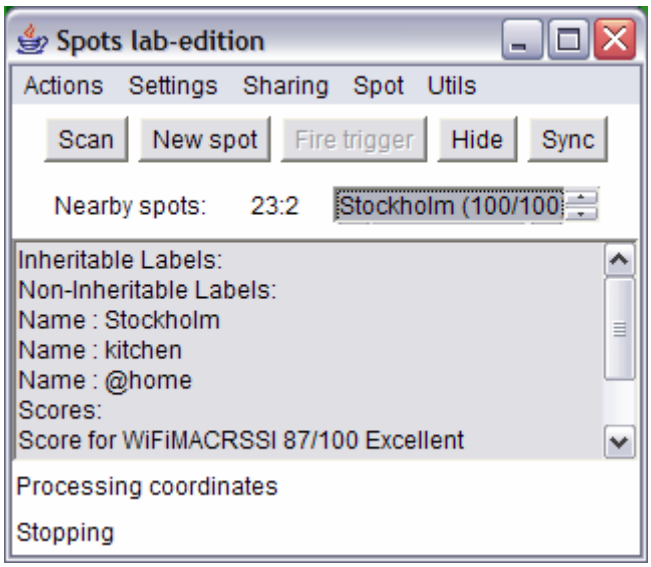


Figure 13. The user interface of the Spots application

A *Spot repository* with the purpose of storing spot information and labels is running on a central server. An interface towards external applications as well as towards the sensor applications is provided through a set of *web services*. A set of *web applications* is also

running on the server, taking care of management of accounts, and some web applications presented below.

Information about a user's current location is published both locally on the user's device and remotely via a web service interface. Locally, information is published using a server-socket. A plug-in host application is listening to this socket and triggers registered listeners when a certain key in a key-value pair appears in the users current spot. spot definitions are shared between users since every newly defined spot is uploaded to the repository using the web service interface as soon as a connection to some network can be found. The sensor application is also regularly checking the repository for updates.

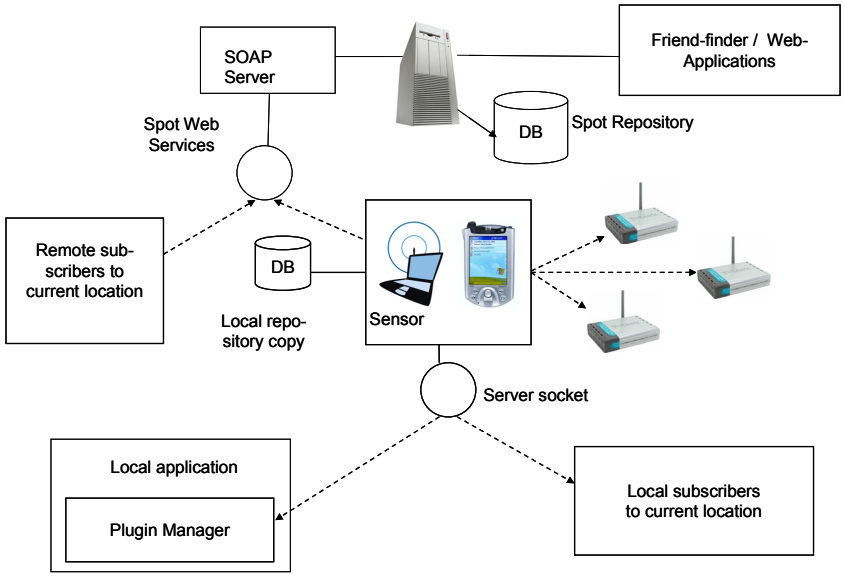


Figure 14. The initial Spots system architecture

4.3.3 WiFi positioning

In search of the most suitable positioning technology to be used in the first implementation of the Spots system, several candidates were examined. There are constraints to most positioning technologies such as with GPS positioning, which has limited to no coverage indoors. Another interesting candidate was WiFi positioning. Positioning based on using properties of WiFi/802.11 access point transmissions is a technology under development that has shown a great promise. Several efforts such as (Bahl and Padmanabhan 2000, Haeberlen et al. 2004, Tao et al. 2003) have built systems showing impressive achieved precision down to a few meters. These efforts based on geometric models require quite extensive pre-calibration work to work well though. Other systems such as (Krumm and Hinckley 2004, Cheng et al. 2005) have built systems making a tradeoff sacrificing some level of precision in favour of minimizing pre-configuration efforts still achieving granularity usable for many tasks. Especially a system like the Spots system, which relies only on location detection rather than relative positioning, can benefit from such a tradeoff. The PlaceLab initiative (LaMarca et al. 2005) concerned with developing a low cost solution for positioning based on detecting wireless beacons including WiFi AP by using standard

commodity hardware have created an API supplying classes that can be used for accessing information from surrounding wireless transmitters, this API was used to build the Spot sensor in this implementation of the Spots system. The low-cost aspect using standard commodity hardware, the pervasiveness of access-points, and applicability for use on a variety of high-end devices including PDAs, laptops and also including a few mobile/smart-phones made this candidate the preferred choice of positioning technology for the Spots system implementation.

The classification algorithm used was similar to simple 'fingerprinting' algorithms that have been used by for instance (Cheng et al. 2005) where the test setting is not mapped in advance and no 'training sessions' have been performed on the system. The algorithm simply calculated as matching score based on the differences in the received signal strength indicators for each access-point listed in the profile of the Spots and in the sensor readings and thus did not address problems associated with how 802.11/WiFI network cards from different vendors calculate this value in different ways as have been mentioned in for instance (Haeberlen et al. 2004, Tao et al. 2003, Cheng et al. 2005). For matching in between readings and profiles created by the same type of WiFi-NIC, this algorithm was very successful in room-level granularity presence detection indoors, for office size rooms or larger (we do not want to give a more exact estimate since we have only tried it out in a few parts of a single building), as long as there were at least two (preferably more) discriminating access-points to determine the location from, as long as the user remained stationary in the room for at least 30 sec to a minute, since the system did not handle motion very well. Since the Spots system is only about detecting presence at Spots, not tracking users movements, effects of motion was not further addressed, neither was any specific distance-measure to specific access points. But for classifications where different WiFi-NICs were used for creating and sensing the spot, the classification was not surprisingly highly unreliable. Studies have addressed this problem, and it is necessary to solve this problem in a future implementation since it makes this system slightly less usable than desirable considering users are expected to use a wide variety of devices. In the prototype built for the user studies it was deemed that it would be acceptable to have users define multiple Spots at each location, one for each type of NIC. There were no controlled experiments of creating Spots outdoors, informal testing throughout the development of this system showed that it is possible to create Spots outdoors using this positioning technology, but we give no estimates on how well it works.

4.3.4 Example applications

Based on the Spots system, a number of location aware example applications were created, in order to display how the Spots system can be used in different ways.

The Friend-finder application

The Friend-finder application is a simple web-application displaying a list of the latest known whereabouts of registered users with a timestamp of when that location had been reported. The application will display only labels created by the spotted user and the ones created by the user querying for the information.

tetsttt	N/A	N/A	N/A	
tommy	402	402	2005-05-29 19:31:39.066464	Infokiosk Eniro
tommy2	N/A	N/A	N/A	

Figure 15. The website part of the FriendFinder application

Presence sharing using instant messaging

Many instant messaging applications like for example Skype and Messenger have a status information function that allows users to manually enter textual descriptions about for example their current whereabouts. By connecting the Spots system to these applications it becomes possible to automatically update this information, based on the labels from the last visited spot.

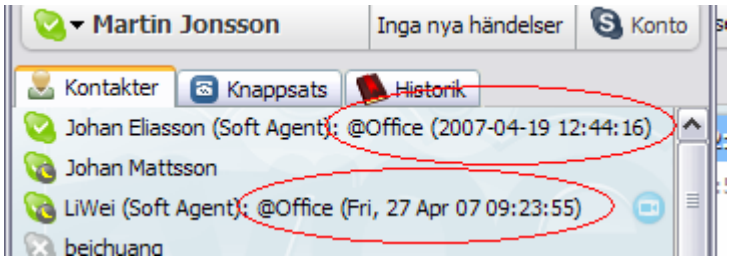


Figure 16. A Skype client with persons showing status information acquired from the Spots system

A location sensitive messenger

Using this application it is possible for a person to send a message to another person (or to himself), that will be delivered when the user is detected at some specific location. The messages are composed and sent from a web page. An add-on to the sensor client application will receive this message immediately and wait until the right spot is detected until it will display the message.

Location aware TeamSpace

This application was created by location awareness to an existing application. The TeamSpace (Shih et al. 2004) software is a toolkit for building applications in interactive work environments with decoupled artefacts and services. It contains functionalities such as controlling mouse and keyboards of other computers as well as sending and displaying documents to other computers. By connecting it to the Spots system the application would automatically connect to recourses in the local environment when the user entered a new spot (Figure 17).

Figure 17. Mobile devices get automatically connected to shared devices when entering the room

Interactive room configuration with a physical cursor

In this application the Spots system was used to identify the different devices present in an interactive room designed to support local collaboration. In this system each spot was represented by a RFID tag. A wireless RFID reader was connected to the Spots application whereby labels assigned to each device can be acquired. On top of the spots application a number of configuration features were then designed. One such feature was for example that by rapidly sweeping over the tags of two computers these devices becomes temporarily coupled. This means among other things that you can use the mouse and keyboard of one computer to control the other computer.

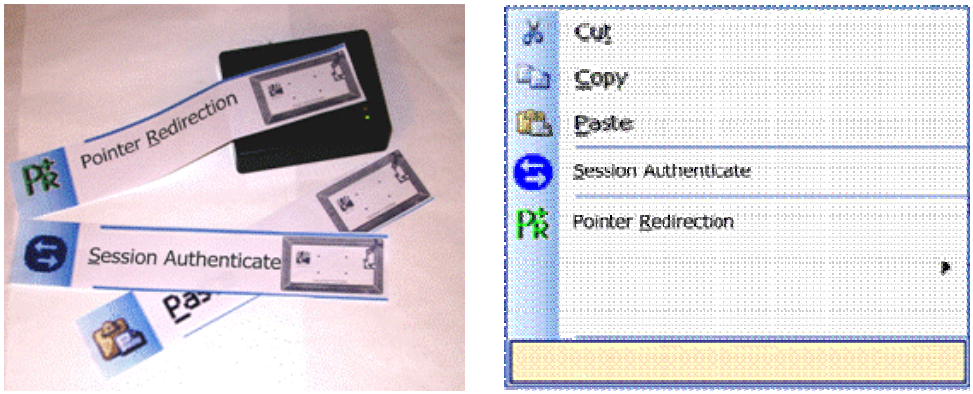


Figure 18. The physical cursor platform

Adapting existing web applications

The Spots system was also used to exploit two already existing web applications; the 'Infokiosk' application, a guide application to a university campus area, and the 'Eniro' application a yellow pages similar application that can provide maps over the city area and links to nearby stores etc.

By using Spot labels containing search criteria applicable to these applications, links could be created in the Friend-finder application that would initiate a search and present the results from these web-applications

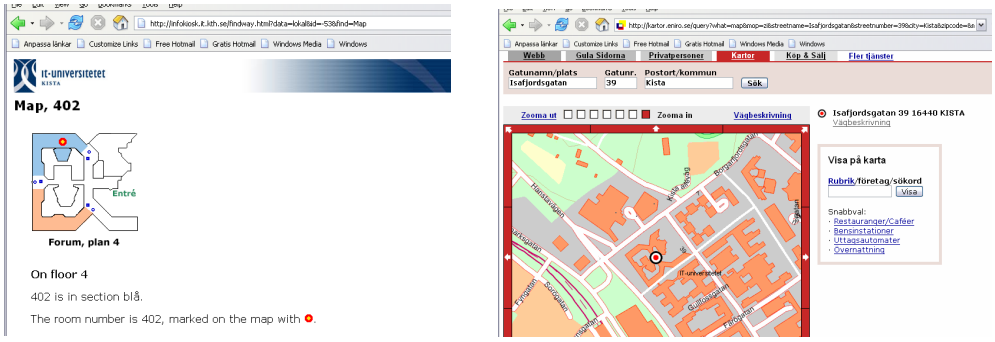


Figure 19. The Infokiosk and Eniro applications

4.4 Summary of system properties

In this chapter the selected details of three context information middleware systems have been presented. As will be presented in the forthcoming chapters, there are some common design approaches in the three systems presented above, but also crucial differences.

4.4.1 Context Shadow

The Context Shadow System is mainly targeting issues of context aware service discovery. In the system software services are tied to entities such as locations, users or groups. Sensor information is used to create dynamic links between e.g. users and locations. In this way a searchable web is created which makes it possible for applications to make queries regarding the physical and computational context of the different entities in the system.

The aim of the Context Shadow system is to offer a "shadow of the real world" for software services. By using a simple query API it is possible for services to acquire important context information such as information concerning local artefacts, services and people. More specifically the system provides:

- Support for context aware service discovery.
- Organization of services and context information in meaningful collections.
- Context information for applications derived from sensors and other services
- Support for refinement of context information.

4.4.2 The ACAS architecture

The ACAS architecture is mainly targeting issues of representation and communication of context information. The architecture provides means to initiate subscriptions of sensor information from unknown service environments in a dynamic fashion. Sensor information is handled as events and is encoded in a simple context

description format. The architecture is person centric in the sense that each user participating in the system has her own personal context information system handling that person's context information and the linking to different applications. The system does not try to maintain any higher level context representations. Instead context refinement is seen as a series of transformations of data in order to achieve the required input for an application. The rules used to transform the data are injected into the system together with the deployment of any new application.

4.4.3 The Spots system

The Spots system focuses on active participation of users in issues of system maintenance and interpretation of context information. While the two previous systems rely heavily on that the surrounding environments are enhanced with both sensor and software components, the Spots system is an attempt to achieve location awareness by utilizing patterns in already existing properties of the environments. A place can for example be defined by the specific radio strength profile created by the nearby WiFi access points. By utilizing existing properties of the environments it is possible to create location aware systems in a more ad hoc fashion.

In the Spots system the users themselves defines places of interest to them. Users can then add information describing properties of those places, either to share their whereabouts with others or to be used by applications that can use it to in some way adapt their behaviour. Information about the defined places can be shared with other users through a web of community servers.

4.4.4 System comparison

The different system designs are different in many ways but there are also similarities. Among the similarities are for example the concept of a personal repository for context information, avoidance of complex context models and a service centric design approach. In the forthcoming chapters some specific design issues will be discussed in further detail. These issues include how to model context information in the middleware, how to support service composition and how the role of the user can be revisited in different ways.

Chapter 5
Designing for generality in context interpretation processes

In chapter 3.2 a design question was introduced concerning how context information interpretation processes in the middleware can be designed to make it possible to perform useful interpretations on context information while still supporting a wide range of applications. This becomes problematic due to the clash between on one hand the requirement that a middleware system should be application independent in order to be useful, and on the other hand the need to choose specific internal representation formats of context information that always to some extent limits the range of possible interpretations. The problem was more specifically phrased as:

How can you design mechanisms that allow for transport, storage and interpretation of context information and at the same time minimizes the restrictions on the versatility of this information?

A number of variables were also identified as having an effect on the versatility of the middleware system. The proposed design solutions are then evaluated against these variables.

Layer division: *Versatility increases if a larger part of the interpretation is handled in the application layer instead of in the middleware layer.* It is not obvious how large part of the interpretation that should take place in the middleware layer, and how much that should reside in the application layer.

Representations: (A CIM system can be designed with or without representations of instances of real world entities) *A system that does not contain representations of instances of entities is easier to modify to fit new application domains.* The generality aspect is thus also affected by whether you choose a situational or representational encoding of the context information. Whereas a formalization that contains representations of real

world entities might become more coherent, a situational or event based encoding allows for parallel interpretations inconsistent and incomplete models.

Level of detail: (The amount of information included in the context model, in terms of the number of concepts included and the number of claims made on these objects, including how they relate to each other etc.) *A system that contains a more detailed context model might be able to cover more application domains but is also harder to modify once it is not applicable.* Including a lot of concepts or 'knowledge' in a context model makes it possible to make higher level deductions on the activities the sensor information is describing. On the other hand, it might be harder to find a rich model that is applicable for a broad range of applications.

Openness: *A system with an open taxonomy context model is easier to adapt to new application domains.* The taxonomy of the context model used can be either open or closed. In a closed taxonomy, the vocabulary is determined in advance, whereas in an open taxonomy, like in so called folksonomies, additions and changes can be made continuously.

5.1 Design approach 1: A representational approach with interlinked context repositories

In the Context Shadow software services are tied to entities such as locations, users or groups. Sensor information is used to create dynamic links between e.g. users and locations. In this way a searchable web is created which makes it possible for applications to make queries regarding the physical and computational context of the different entities in the system. The Context Shadow system is based on a blackboard architecture where certain stable entities are represented as context servers. These servers act as repositories for context information related to that entity. Typical entities that can be provided with a context server are persons, locations and groups/projects. A context server contains two types of data; context information concerning the entity that the server represents and links to other context servers.

Sensors and applications that provide context information are provided with a simple communication interface which allows them to post their information to one or several specified context servers.

5.1.1 Cross referencing

A key feature of the system is the possibility to establish dynamic links or relations between different representations in the system, i.e. the context servers. A typical example of this is the detection of a person entering a room. If either the person detects the room or the room detects the person this results in that a cross reference is established between the local and personal context server. This can be done since the location sensors provide references to a location context server, or in the case of person detection, the person sensor receives references to personal context servers.

The context servers and the links between them create a searchable web where the topology changes dynamically. Information about the users' current context does not

only consist of pieces of information in the context servers, but is also embedded in the topology of the surrounding web of context servers.

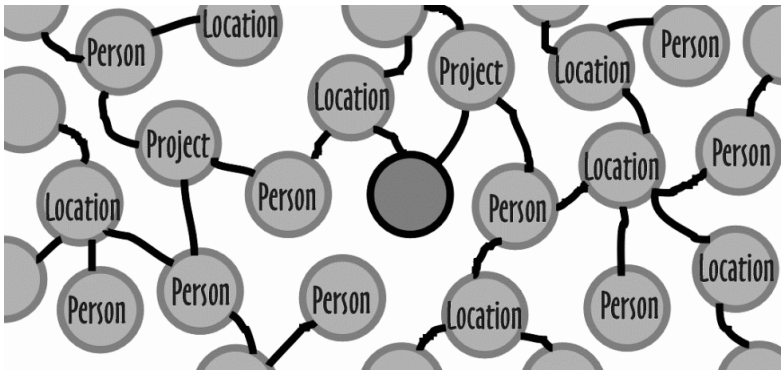


Figure 20: The linked context servers create a searchable space, where the topology of the space is part of the context information.

More static references can also be established. Examples of this can for example be references describing the relation between locations. There is a general problem regarding how to represent location in context aware applications. In Context Shadow there is no structured way of describing locations in terms of hierarchies and distances. Instead you define "places" in an arbitrary way, and then create relations between these places. In this way it is possible to create hierarchies when needed, but there is no requirement for developers to provide a complete or coherent location model. Another example of references of a more static nature could be references between persons and projects. By connecting people to each other via a project entity, it is possible to create CSCW tools that e.g. could have knowledge about meeting history, shared documents etc.

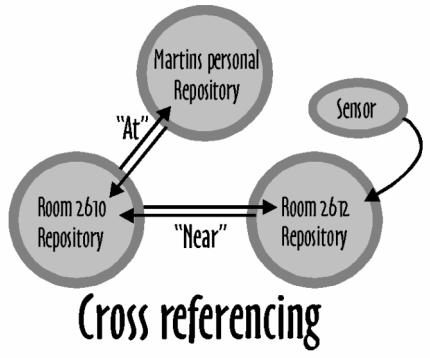


Figure 21.The context servers are linked with references. By following the links it is possible to acquire context information from other context servers than your starting point.

5.1.2 Evaluation with respect to the generality problem

In this system a simplistic context model is encoded in the system structure (low richness). This model consists of three basic entities: Persons, places and groups. These entities can have different kinds of relations to each other. The only

implemented relations were a "At"-relation between persons and places, and a "member of" relation between persons and groups. All sensors and applications using the Context Shadow system will have to conform to this model (low openness). Additional context information can be added that describes each entity, but the responsibility for the format of this context information is pushed up into the application layer.

Described using the design parameters above the interpretation mechanisms consists of a *representational* model using a *closed* context model with *low richness*, and where a restricted set of interpretations can be performed in the middleware layer. Transport mechanisms for additional sensor data makes additional interpretation possible in the application layer. The *adaptability* of the middleware is restricted to adding new instances of the predefined model entities.

Implementations of several useful applications on top of a middleware system with this specific combination of design principles shows that this approach works at least for this set of applications.

The major drawback for this specific design with respect to the generality problem is the closed context model, which makes any major changes very hard to achieve. This became very clear during the initial design of the middleware system, where at one point the underlying context model had to be changed (by adding the group entity) because of new requirements from a new application. This change required a complete rewrite of the middleware which also affected the existing applications using the middleware.

5.2 Design approach 2: Avoiding context models in the middleware layer

The ACAS architecture contains functions for subscribing and transporting context information. The basic idea is that you have separate local infrastructures (companies, buildings, persons etc.) collecting information about local sensor and service information. Each local infrastructure is represented by an infrastructure server, which through a bootstrap mechanism, dynamically can connect to other infrastructures to enable exchange of context information. The ACAS architecture has not been implemented as complete system, and can therefore not be evaluated as such. Different components have however been tested separately in order to prove that the architecture is viable.

5.2.1 Pushing the context model to the application layer

In this design approach the amount of semantics encoded in the system in terms of generic context models has been further minimized compared with the Context Shadow system described above. In the ACAS system the general assumption is that since any piece of context information can be represented in an almost infinite number of ways, one should try to avoid generic models for context information, at least models describing higher level activities. The only factors that should determine

how context information should be modelled are the specific needs of specific applications. The approach in the ACAS is as follows:

The sensor data you subscribe for from the sensors is encapsulated into small packets called "context elements", where each context element consists of a number of attributes. The context elements are produced by *writers* connected to sensors, and the consumers of the context information is referred to as *readers*. The communication is based on a subscription mechanism that can be set up in a dynamic fashion. The sensor data of course has to be formalized in accordance with some standard or model, but there is no requirement that the format has to be adapted to some generic model in the middleware. These context elements are then being gathered over the network into context repositories. Here the middleware can perform different kinds of refinements on the acquired information. The rules transforming the context elements are however mainly tied to, or even provided by, applications that are interested in specific context information. The key problem then becomes to achieve the transformation from the sensor specific encoding to the application specific requirements. With this approach there is no need to try and keep any generic representations of "the context", which we have earlier described as a rather problematic approach.

5.2.2 Defining a general context description language

The context description language used in the ACAS architecture is based on xml. The central item of the language is the context element, an object composed of the following attributes:

- id - An identifier of the context element's instance class.
- value - The value of a property of some entity.
- type - The datatype of the value, like integer, real, string or xml.
- unit - The property to which the value refers.
- entity-reference - The uri to the entity which the context element describes.
- time -The time and date when the value was captured or composed.
- source uri - The uri to the entity which captured or composed the context element.
- source content – Human readable description that explains the context element.

Here is an example of a context element formulated in xml:

```
<acas:contextelement id="123c">
<acas:value datatype="integer"
unit="temperature/kelvin">292</acas:value>
<acas:entity-reference rel="acas:dsv.su.se/k2/r7741/t"/>
<acas:time>Sat Apr 24 00:05:21 CEST 2004</acas:time>
<acas:source uri="uri:acas:dsv.su.se/k2/csf/apax"/>
</acas:contextelement>
```

The id is used to identify the context element's instance class. This class consists of the sequence of context elements that when ordered by time, represent the successive evolution of the context element's value. The practical point of the id is to allow a reader of context information to quickly locate and update previously received elements. Because the id is chosen by the writer it is the writer's responsibility to create an id that has a high probability of being unique. Otherwise a reader that acquires context elements from several unrelated writers runs the risk updating the wrong context element. Such an id could be generated for example by combining a hash of the initial context element (or its generalization), the address of the writer and a random string.

The value can basically be anything, but initially useful configurations are assumed to be scalar values, multi-dimensional values and text strings. Due to the composability of xml there is no objection against structured values, as long as they are formulated in xml. Other string based representations are also possible, but must of course be subject to the character encodings required by the xml representation.

The type is typically a URI (or indeed, a list of URIs that characterize the value. In any event it must enable the reader to understand how to decode the value. The unit is another URI and it identifies the property which the value refers to, for example temperature or location. The entity-reference is a URI that identifies the entity that has the property indicated by the unit. Together, entity-reference, unit and value form a basic tuple that is commonly used to describe properties of entities: (entity, property, value).The time is simply a timestamp that enables a reader to determine the age of the context element. The source URI is used to identify the origin of the context element. This could be a writer attached to actual sensors, or the output of a refiner process.

5.2.3 Middleware refinements using application layer rules

It is often not meaningful to send sensor data directly to an end-user application. Applications are often more interested in higher level events, such as when a person is in a specific location (Salber et al. 1999, Pascoe 1997), or if a specific activity can be detected (Werle et al. 2001). Deducing this information from sensor data might cost a certain amount of processing. Since most context aware systems uses end user applications on mobile devices, minimizing the amount of processing and network traffic is desirable. Processing the data in the infrastructure instead, will reduce the burden on the application and also reduce the amount of network traffic needed.

In the ACAS architecture, the processing engine will primarily see the processing of context information as a transformation process, thus we have chosen to work with the standardized XML transformation language XSLT. The following design proposal is a sketch of how an XSLT based reasoning engine could be implemented. The only testing and implementations made consists of experiments with the XSLT language, in order to make sure that the design is actually realizable.

XSLT

Extensible Stylesheet Language Transformations (XSLT) (Clark 1999) is a functional programming language used to specify how an input XML document is converted into another text document. An XSLT processor reads both an input XML document and

a XMLT stylesheet. In the transformation process, XSLT uses XPath, a syntax for defining parts of an XML document, to select parts of the source document that match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document. XSLT can add new elements into the output file, or remove elements. It can rearrange and sort elements, and test and make decisions about which elements to display, and a lot more.

According to some opinions like (Novatchev 2003), the XSLT language should be perceived as any other functional programming language, being equally powerful and expressive.

The suitability of XSLT for reasoning about context

In other systems that in some sense refines or reasons about context information probabilistic techniques, such as Bayesian networks have been successfully used. The probabilistic approach is especially useful when dealing with the inherent uncertainties that come with working with sensors. Typically these techniques are used for different kinds of classification tasks where the possible set of outcomes is a closed set and where sample data is available for training the system. For this kind of tasks the XSLT language is probably much too cumbersome. A large part of the reasoning with context information can, however, not use probabilistic techniques, due to different reasons. There is e.g. often not a closed set of possible outcomes, new situations might appear or new types of sensor information, or there might not be any training data available. So even if less powerful and accurate, a semantic rule-based reasoning engine might be preferable under some conditions. This section describes an infrastructure that is supposed to handle context information coming from numerous sources, and that is supposed to be used by an undefined set of applications. This will have a number of implications:

- New types of sensor information might appear dynamically.
- The rules are constantly changing, or are being added or removed, to meet new demands of new applications.

In light of this, using XSLT to express rules might have a number of advantages:

- Standard transformation engines are widely available.
- Modular approach, rules are expressed as documents and are thus easy to add or remove.
- Since XSLT performs general transformations it becomes possible to tailor the output from the reasoning to suit specific requirements from legacy applications
- Since the rules are expressed in XML, meta rules can be constructed, that reasons about other rules.

The XSLT language however has limitations:

- XSLT only transforms documents, it cannot perform other actions, and thus it will only generate events that can trigger other applications.
- Hard to deal with conflicts and interdependencies between rules.
- The syntax is complicated and templates can be hard to debug.

Context Refinement module

The context refinement module fills following functions:

- It identifies the occurrence of higher level situations from sensor data.
- Aggregation. Pieces of sensor data can be combined in order to create new types of context information.
- Format translation. Some subscriptions might require sensor data in a specific unit or format.

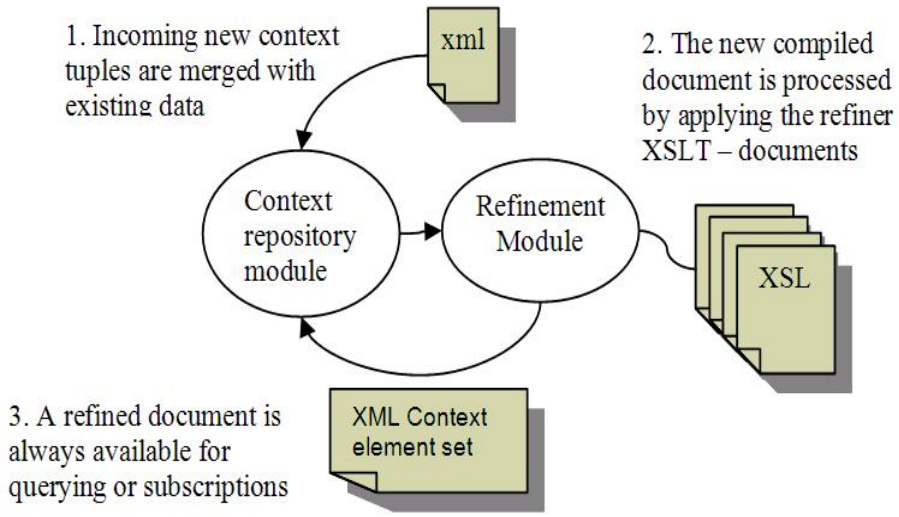


Figure 22. The context refiners transform the available context elements and posts the results back into the repository.

In the context refinement module the rules are expressed in XSLT which is normally encoded in an XML document. In this system the XML document format will be especially useful when rules are dynamically added to the system. Having the rules encoded in XML documents stored as files, also makes the system more robust since it can easily restore its state after a system crash.

To reduce the complexity that might be caused by interdependencies between rules an important limitation of the behaviour of the rules is declared:

Context refiner rules may only add new context elements to the context description, not change or remove items. This restriction will not only make the interplay between rules easier, it also makes sure that the original data is always available for alternative decision paths. Figure 22 describes the procedure of how the XSLT rules are applied to the context information.

Application based rule provision

Applications that subscribe for context information might have different requirements regarding which situations that should be detected, or how the context information should be encoded. It is not reasonable to assume that all possibly interesting formats or situations could be produced by general context refiner rules. Instead, each application that wants to subscribe for some specific information should provide its specific requirements to the system.

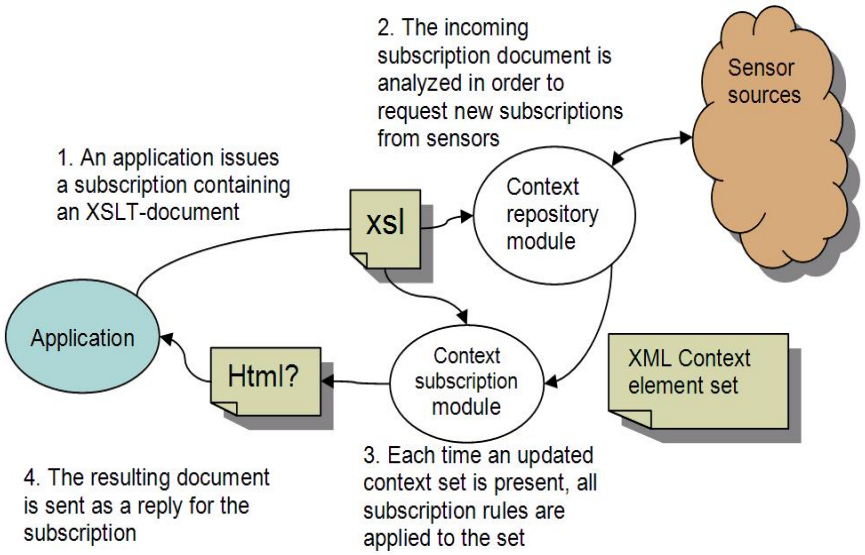


Figure 23. Schematics of an application based rule provisioning system

In this system, this is solved by letting the applications provide XSLT-rules describing the information that is sought, conditions or situations that has to be fulfilled, and finally, in which format the result should be encoded.

Having the possibility to control the encoding of the results of the subscriptions makes it possible to tailor the output to specific standards used by different legacy applications. One could e.g. imagine that information about a person's availability or whereabouts could be encoded in the iCalendar format, used by many of the existing legacy calendar programs. In this way the context information could be visualized through a standalone commercial application. Another example is that location information containing coordinate information maybe could be encoded according to the NMEA protocol, which is widely understood by map programs and plotters.

One could imagine situations where there are lots of sensors available that could produce sensor information. There is however no point in acquiring information that nobody is interested in. By analyzing the demands issued by the applications one could formulate new queries that could be used to set up subscriptions of new context information.

5.2.4 Evaluation with respect to the generality problem

The interpretation mechanisms of this design can be characterized as a *situational approach*, since the system triggers when certain templates match incoming "events" of sensor data. The context model, constrained by the Context Element format, is very simple and can basically be considered as *open* since the model contains very little semantics. Basically the only semantics that the model entails is that there are entities that may have some arbitrary properties. This makes it possible to add both new instances of entities and new types of entities or properties without having to make changes to the underlying structure. The solution with respect to *layer division* is probably the most central part of this mechanism, where application specific rules are created by which the context elements are interpreted, combined and/or abstracted. The actual interpretation process however takes place in the middleware layer. In this way several inconsistent application specific context models may coexist in the system, where new applications may invent new entities and properties without affecting other working parts of the infrastructure. The combination of the open model and the application specific rules makes the system highly *adaptable* to new application domains. The generality approach is thus addressed more through extensive support for adaptations to new application domains than by an initial broad coverage of domains.

The major drawback with the approach is the cumbersome process of creating new rules. The XSLT language easily becomes very complex when more advanced calculations have to be performed. One solution to this problem is to generate tools to simplify rule creation. Another approach which has already been taken on is to create a new rule description language in combination with specially designed interpreter.

A patchwork approach to handle inconsistencies

To have an infrastructure without an explicit context model of course have its risks. One obvious risk is that the rules provided with the application do not match the format of the incoming context elements provided from the sensors. What one has to bear in mind however when designing this kind of systems is that they will be deployed and maintained by real people as part of some work practice. This kind of incompatibility problems can then in many cases be circumvented by adding a "patch", a rule that transforms the incoming data to fit the sought format.

The end result will then not be the perfectly interoperable bottom up engineered system that so many people dream about in the ubicomp domain, but rather a patchwork of application specific solutions that are stitched together as well as possible.

5.3 Design approach 3: Collaborative location modeling

The Spots system targets the problems of setting up and maintaining context aware systems, with a special focus on location dependent systems. In the Spots system the users themselves define places so that they for example can share information about their whereabouts with other people. The users may add any meta-data to describe

each place or spot, and that information can then be shared with other members of a community. With this approach generality is achieved by letting the end users affect how a specific piece of sensor data (a location in this case) should be interpreted. In this way the users may more easily appropriate the technology to fit with their actual activities and social conventions.

5.3.1 Different approaches to location modeling

Location modelling is the task of describing the environment in some manner that can be used for describing properties of location in relation to some reference system and also describing the properties of that system. There is a tendency to distinguish between two groups of location models (Domnitcheva 2001). Models of the first group consist of 'physical' and 'geographical' location models. Physical location is related to a global geographic coordinate system and provides absolute, accurate, grid based position in form of <latitude, longitude> pair, with a third <altitude> coordinate that can be added if necessary (Domnitcheva 2001 and Becker and Dürr 2004). Geographical location deals with natural geographic objects with a hierarchical organization (Domnitcheva 2001). Both of these operate on a geo-location scale.

The second group of location models consists of 'geometric' and 'symbolic' models. A geometric model describes locations and located objects by sets of points, areas and volumes in relation to some reference coordinate system. Reference coordinate systems can be distinguished into global and local geometric coordinate systems according to Becker and Dürr gives the World Geodetic System 1984 (WSG84) as a global example and the Cartesian coordinates of the Active Bat (Ward et al.. 1997) as an example of a system that is typically valid only in a specific room equipped with such a system (2004). For physical, geographical and geometric models you can easily calculate the distance between two defined positions, and in the geographical and geometric model it is possible to determine whether two areas overlap, touch each other or if one area contains the other.

This is not the case with symbolic models. In symbolic models locations are represented by different abstract symbols, which can be of any type. For such models Becker and Dürr (2004) suggest something they call 'symbolic coordinates', for instance identifying values such as the Cell id in GSM networks and MAC address in WiFi networks, but also street names and room numbers or graphical symbols. Those are not coordinates in the manner that they provide any information about spatial properties, but as Becker and Dürr define a coordinate it is an appropriate use of the term. Their definition of a coordinate is:

"A coordinate X is an identifier which specifies the position of an object with a respect to a given coordinate system. A coordinate system is a set X of coordinates."

Position here can refer to membership of some specific set for instance. The proximity, containment and other relations between symbolic coordinates are not implicitly defined as it is in the other models. There is no natural set of relationships between them or a natural hierarchy of them. Such properties must in symbolic models be provided in some way if you want to be able to do such reasoning about the domain. In the semantic location model suggested by Pradhan which is based on associating semantic information to 'Uniform Resource Indicators' (URI) the

properties for liking URIs to one another and create some type of hierarchy are the same as used for the Internet DNS service with different domain levels (Pradhan 2000). That approach has been used for instance by Roth in the hierarchical location model he describes (2003).

The idea of using semantic symbols for describing location is not new, and is a concept that can be used in several ways. The URI model used by Pradhan mentioned above is one example of a semantic location model. Shared by all semantic models is that they do not focus on having the properties of physical coordinates. Instead they focus on entailing the *meaning* of a location (Roth 2003). The strengths of using a semantic symbolic location model compared to other location models lie in that they are easily extendable and that you can make them both user-definable and machine as well as human readable (Pradhan 2000).

5.3.2 Empowering users in the location models

In the Spots system the domain is restricted to the notion of places, a domain that in many cases is both limited and well defined enough to fit within the boundaries of a context model. One problem concerning these models is however that the design of the underlying model and the mechanisms controlling the interpretations and decision making in most cases are created by the system designer before the system is given to the users. What this means in reality, is that the system designer is the one assigning meaning to sensor information and in some sense restricts the ways in which the information can be interpreted.

There have however been other efforts on developing location-aware systems using models that give more control to the user with respect to how meaning is extracted from sensor information. Examples of such systems are for example the *comMotion* system that uses GPS positioning, focusing on discovering places of importance to the user (Marmasse and Schmandt 2000), *Reno* using GSM positioning focusing on social aspects of sharing location information (Smith et al. 2005) and *GeoNotes* (Espinoza et al. 2001, Persson et al. 2002) which in one version has been implemented using a very crude WiFi positioning sensor focusing on spreading and filtering information tied to location. These studies have shown that it is possible to build a system based on such a location model and have been used as influences when creating the Spots location model. Especially the GeoNotes study emphasizes the advantages of allowing users to share and reuse semantic descriptions of locations to allow the emergence of shared ontologies amongst the users have influenced the user controlled labeling approach used in the Spots system.

5.3.3 The Spots location model

The approach that we want to explore in this work is to start out with a very simple base-model, with basic elements that contain no semantics and that are viable for a broad range of application scenarios. The users are then free to add the semantics describing each location in order to make them understandable by other people or tailored to be machine readable by specific applications.

A key requirement for this kind of open ontology systems to work is that they are used within the borders of *groups sharing some contextual references*. In such a group context,

expressions can be used, which would be considered ambiguous in a larger setting. A group of employees of a small company could i.e. use an expression such as "the meeting room", without causing any confusion on what is referred to. Another aspect which has to be taken into account is how the naming practices will evolve over time. A hypothesis we have which is yet to be verified is that over time and due to the incorporation of the system in regular practices, standards for naming will evolve within the groups.

5.3.4 Defining places as Spots

To have an unambiguous description of the locations or places that exists in the Spots system, it was decided that describing them as places or locations was not suitable. It should be clear that there is a specific type of location/place that exists which is understood by the system. The idea was that the Spots system can identify Spots and a spot is something that is not just any place, a spot is a specific location/place with associated information that can be found by the sensor application in the Spots system, compared with for example a GPS-based system which can identify any location (at least outdoors). To make a location/place identifiable you need to define a spot at that location. What it actually means to be detected in a spot should be left open. The Spots in the Spots system are an entity representing a location/place that has;

- a unique identifying value.
- a Spot profile
- at least one Key-Value label

Spots can also act as containers of other spots. Every children of that spot will inherit all labels of the parent spot. The containment property is defined using the label entity. The above mentioned entities relate to each other as shown in the following very simple conceptual model.

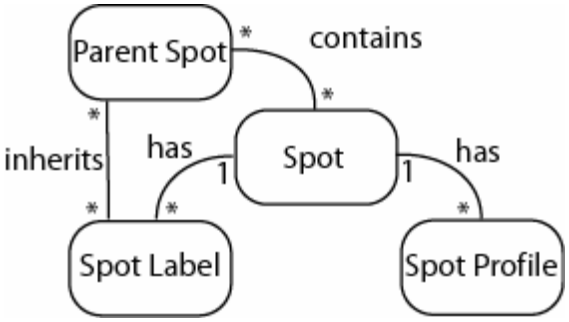


Figure 24. Conceptual location model design

5.3.5 Spot Labels

Spot Label is the user definable context entity that exists in the Spot System. It has the additional responsibility of being able to present itself semantically as a key-value pair to another plug-in application and can thus be defined as follows: *A Spot label in the*

context model of the Spots system is a specific key-value pair that is associated with a specific spot. Except from the key-value pair each label also contains information about the *owner* of the spot and information regarding whether the label should be shared with other users or not.

A user is allowed to add any number of labels to a spot, and is also free to define new types of key-value pairs when needed. In the implemented application, the model however contained on pre-defined key, "Name", which provided as a default key for human-readable naming of the spots.

5.3.6 Spot Profile

The Spot Profile entity encapsulates the detectable properties of a location that is used to separate locations from each other. The responsibility of the Spot Profile entity also includes being able to compare Spots and calculating similarity between Spots. The Spot profile is thus a container of a set of semantic location coordinates that belong to a specific spot. The spot profiles are *sensor dependent,* meaning that a profile created using e.g. a WiFi sensor can not be used to identify the location of e.g. a Bluetooth sensor.

5.3.7 Sharing of context information in user communities

The users can create and name places as they in accordance with their own needs, but they can also share place information with others. The Spots system is not used in isolation but in combination with some application. As has been presented earlier, these applications include both such that provide a degree of automation for the individual, but more often the applications deal with collaboration or coordination with people in some social setting.

A crucial part of the spots system thus, becomes how to share the self-created place definitions between people in a way that does not create a complete chaos. The approach that was chosen was to base the system on *communities*. So instead of sharing your place definitions globally, with anyone using the system, you only share the information within the community or communities you are part of.

So what is a community in this sense? One obvious type of community is created by the application you are using on top of the Spots system. Since the place information in most cases is designed to be used within a specific application, it makes no sense to share this information with people using the spots system with another application. In many cases you might however want to have communities within a larger application community. If you for example consider the Friend Finder application, a person might be part of different social communities where you would like to use different descriptions of places for each community. Office room numbers might for example make perfect sense to the person's colleagues, but doesn't mean anything to her family members.

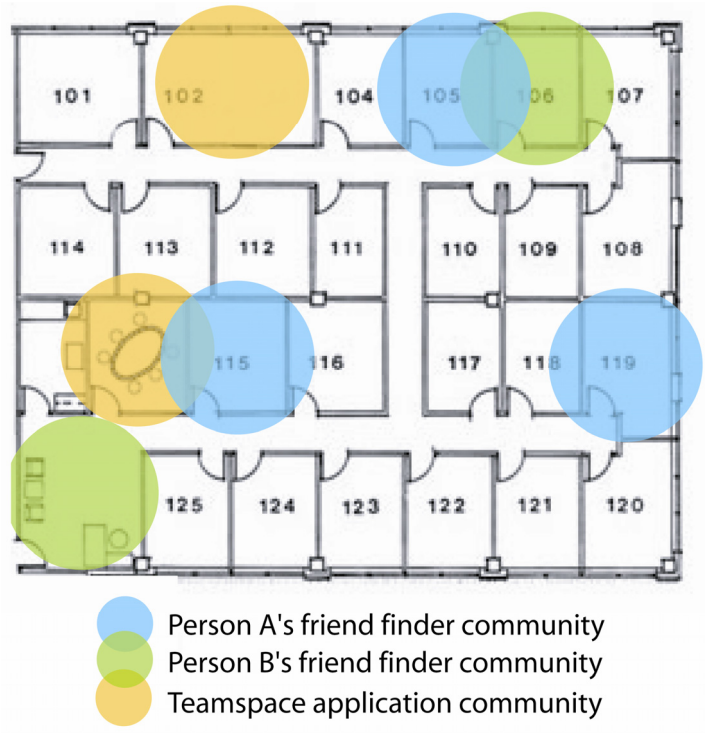


Figure 25: The defined spots might be overlapping, which is not a problem when used by different user or application communities.

The technical implementation of the community based sharing is achieved through the use of community servers. A community server is an application that is easily setup on any computer, and that will act as a shared repository for spot information for a specific community. To join an existing community you set up a subscription for specific key-names against such a server. The system will then continuously keep your personal repository in sync with the community server.

5.3.8 Evaluation with respect to the generality problem

The context interpretation mechanisms presented in this design can be described as a combination of a *representational* and *situational* system. It is representational since each location instance is represented in the system, but situational from the application perspective, since the application only receives sets of tags that can be matched against situation templates.

The interpretation mechanisms can be divided into two parts. One part concerns matching the incoming sensor data against a set of predefined location templates. The other part concerns what information should be passed on to the applications when a successful match has been detected. The general idea is that the sensor-template matching part should be free from all semantics describing the situation being detected. The meaning making process is seen as a separate activity where end users determines what the template represents in a way that is useful with respect to the context in which the system is being used. The system then becomes highly *adaptable*.

The biggest problem with this design with respect to the generality problem is that the application domain already is restricted to cover only certain location aware

applications. It is however possible to generalize the design concept to a fairly large extent. The sensing and template part of the design could theoretically be exchanged to almost any other kind of sensor technology. There are for example several technologies that automatically extract patterns from sensor data. Any such semantic-free systems could be combined with user controlled tagging mechanisms.

5.4 Evaluation of the approaches in terms of the design parameters

5.4.1 Richness

In this chapter an approach to achieve generality in context information middleware was presented that was based on the idea of using simple context models. So how can we show that choosing a simple model is a way to achieve generality? In this work we presented an implementation of a system that was based on a simple model, the Context Shadow system, together with some example applications. The fact that some applications has been successfully implemented using this approach does not however mean that the same applications could not have been implemented using a richer model. The only claim we can try and make is that a richer model becomes less generalizable per definition.

An immediate response to this claim could be that a simple model could be viewed as more incomplete than a rich model, and thus work less well. The notion of completeness does however not make any sense if you don't have a well defined application scenario that can be used to create the boundaries of the model. In the light of this claim you can argue that the simple model always can be extended in the application layer in order to create a more complete model.

The idea of a general base model that can be extended with domain or application specific models has earlier been addressed by different kinds of ontology based context modelling systems (de Bruijn 2003). The remaining problem with this approach is whether you can actually find a working base model. In the proposed base model from the CONON model (Gu et al. 2004) there are four basic concepts included, persons, locations, computing entities and activities. Each of these concepts then has a number of subcategories, location for example can be either indoor or outdoor, and an activity can be either scheduled or deduced. The concepts also include properties that sometimes specify how the different entities are related to each other.

Even for this very simple model you can however find application scenarios where the model is not applicable. Would you for example categorize a situation in a car or a bus as being indoor or outdoor? Is an interactive room a location or a computing entity etc?

5.4.2 Layer division

You can choose to split the context model over different system layers, in this case typically between the sensor layer, middleware layer and the application layer.

In the sensor layer, the typical action that is being made is to transform some internal sensor representation format into some well known data format, such as degrees Celsius if it is a temperature sensor or m/s^2 if it is an accelerometer. Other more complex sensors will most likely produce data in a more high level representation, such as e.g. the NMEA format for a GPS sensor.

Any transformation or inference to higher level abstractions would typically take place in either the middleware layer or in the application layer. These inferences typically require some kind of context model, so the question that remains is where to situate this information.

In the Context Shadow system presented above, the major part of the model was situated in the middleware layer. A number of applications were constructed on this system, targeting issues of service composition (see chapter 4 for details). A requirement for these applications was that the services using the middleware often were really basic and simple, with no capabilities for any inferences. Instead the middleware provided a simple API, by which information regarding relevant services for this context was provided, based on some predefined assumptions. So for this kind of scenarios the model and reasoning has to be contained mainly in the middleware.

In the ACAS architecture, the middleware does not contain any explicit context model. The only restriction is a context description language that determines how context information should be encoded but that don't contain any representations or concepts. So instead of trying to maintain some kind of knowledge representation the task is to try and transform incoming events to a format that is understandable for some application. The rules used for the transformation, that contains all the concepts needed, are application specific, but reside in the middleware layer. In this way the middleware can still take care of a lot of the computational work, but still act as a generic infrastructure. A drawback with this approach is that it might be hard to perform any complex reasoning using this approach, for example to infer high level activities where a lot of concepts and properties have to be present. Another more serious drawback is that the application specific rules have to know which sensor encodings to expect and how to interpret them. This problem does however exist also in approaches where the entire context model resides in the middleware layer.

5.4.3 Using representations of real world entities

One decision that has to be made when designing a context information middleware system is whether or not to use internal representations of instances of real world entities. An existing context information middleware that uses a representational approach is the context fabric system (Hong and Landay 2001), where different real world entities were represented by URLs pointing to xml pages describing the entity. In this examination two of the three middleware designs use representations. In the Context Shadow system, there are representations in the form of context servers, representing places, persons and groups. In the Spots system representations of places exist in terms of spots. The benefit with a representational system is that the information about these entities is persistent. In this way you can have continuous access to the state of the entity, by examining it with queries etc. In this way you can get information about for example how it is related to other entities.

The representational approach however has a number of drawbacks. In a distributed computing environment it might be hard to share the representation so that it is consistent between the different entities on the network. In the Context Shadow system this problem was solved by letting the representations become networked objects that could be examined over a query API. In the Spots system each person has his own version of each representation. The representations are then synchronized with other people via community servers keeping track of the representations shared by a certain group of people.

Another drawback with the representational approach is that it makes evolution of the models tricky, especially in a distributed scenario. Not only do you have to alter the actual representations, but also the APIs and methods in all instances of the applications using the model. In some cases you could imagine incremental changes that where the old examination interfaces could work in parallel with the new ones, but most likely even small changes would create extensive problems.

An alternative to the representational approach is what could be described as a *situational approach*. In this approach the context information arrives as a stream of events. These incoming events are then matched against templates describing certain situations that are of interest to some application. A typical example of a such approach is the Stick e Note system (Pascoe 1997), in which situations are defined in terms of specific conditions on the available information regarding time, coordinates and orientation.

The big advantage of the situational approach is that the application can achieve a correct behaviour even if there are context representations that the application does not recognize. The application simply ignores the information that does not fit with the template. In this way you can have several context formalizations deployed in parallel or extend or modify an existing context model. The only requirement is that each formalization must use unique identifiers so that there is no confusion on which context elements to use and which to ignore. Many systems also use combinations of the representational and situational approach. In the Spots system for example there exists representations of the actual places in the system, but from the perspective of an application using the system, they will only acquire a flow of labels describing the current location, a behaviour that very much resembles that of the situational approach.

5.4.4 Adaptability

Another design parameter with respect to the generality problem that has been examined concerns how easy it is to refit an existing context information infrastructure to suit new application areas. This kind of adaptability can be achieved on different levels.

On a basic level the design of the underlying context model can allow for changes and additions to different degrees. An example of this is for example the ontologies discussed earlier with a base model component that can be combined with application specific ontologies when needed.

Another issue concerns how complex the task of modifying the infrastructure to new applications is. Complexity might increase for example if there are instances of context models distributed over several entities which all have to be altered or if changes in the interpretation mechanisms also requires changes in existing applications. Finally there is a question concerning how the adaptations actually are made, and by whom. Is there a user interface to make changes?

The least adaptable of the three designs presented above is the Context Shadow design which only allow for adaptations of the context model on the instance level, but there is no support for changing the types in the model.

With their open context models both the Spots and ACAS designs are much more adaptable. To adapt the ACAS design to new application domains new interpretation rules have to be written, a doable but rather cumbersome task. In the Spots system the adaptability is more explicitly supported with user interfaces to make changes to the model tightly integrated with the system.

As have been discussed before, creating generic context models is problematic. Creating different kinds of support to change the internal models of context information middleware then becomes a crucial issue.

5.5 Chapter summary

In this chapter we have analyzed how to design middleware with context formalizations that can be generalized to previous unknown application scenarios. The generality problem has been approached in different ways in three different middleware designs. Some specific design choices has been investigated, such as the richness of the context models as well as issues regarding how to divide the models between different system layers and whether or not to use instance representations in the system.

Chapter 6
Using context information middleware to support service discovery

An interesting application domain for context information is to try and use it as a mechanism to manage larger sets of software services. Encapsulating computation in terms of services is an increasingly popular software engineering approach, especially in the domain of ubiquitous computing. As has been stated earlier, the ability to achieve a dynamic behaviour with respect to how services interact with each other due to available context information is an interesting problem.

6.1 Design approach 1: Creating meaningful collections of services

The service discovery in Context Shadow is based on 1) a query interface to acquire sets on services based on context information, and 2) a simple key value match to distinguish specific services.

The Context Shadow system can be used for different kinds of resource and service discovery tasks. For example, in Fig. 4, a jukebox service is associated with a person. When the person enters a location, the jukebox service will find a public speaker service when it queries the infrastructure for that kind of services. The jukebox service might make queries on an XML description embedded with the speaker service representation, or it can choose to try and match a specific type name describing the service.

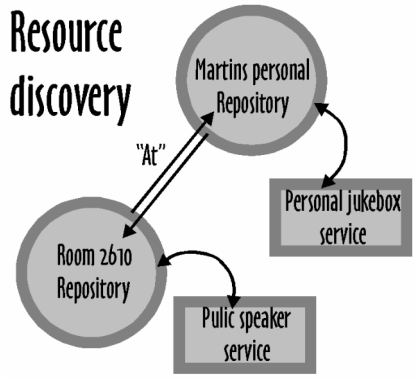


Figure 26. Using Context Shadow, a jukebox service finds a speaker service at the user's current location.

Using the built-in functionalities of the underlying TSpaces, you can make complex queries on the data in the context servers. Context Shadow provides an additional API to search the web of context servers created by the cross-references described above. The queries can be of the type. "Where am I" or "What other people are in the same location as me" and maybe more useful: "What services of type X are available and relevant to my current context".

There is also another powerful way to query the infrastructure using XQL (XML Query Language). In Context Shadow you can attach an XML document to every entity, containing various descriptions of the entity. Using XQL you can query the XML description of the entities. This way you can provide a very open interface towards service developers.

6.1.1 Examples

In the fuseOne application Context Shadow was used partly to discover Jini-based services that were relevant at a certain location. In this case the Jini services described themselves by providing a "service ID object". The querying service then used the information from Context Shadow to filter out services that were irrelevant to the actual situation. One set of components in the system consists of very simple but useful services that can receive and display documents of different kinds on the computer where the services reside. What these services actually do is that they accept arbitrary remote files and instruct the current operating system to launch the application with which the file's type is associated and then open the file. These services are numerous and reside both on personal and public devices. Another component is a context sensitive desktop, which makes the services described above and other services accessible to users via a GUI. This service queries Context Shadow about which other services that are relevant to its user's context, and displays them on the desktop. More specifically the context sensitive desktop achieves the service composition by querying its owner's Context Server for (in following order) "personal" services, local services and services owned by other persons in the room. Whenever a service is found, a service description object is returned to the browser application. The application has already found all Jini services in the network by using the lookup function that comes with Jini, and uses the service description from Context Shadow to filter out services that are not available in the actual room.

Another type of service that was developed is the Active Document agent. The service is active in the sense that it is autonomous (act independently), reactive (react on changes in the environment), and proactive (have its own goals and plans). In this prototype the Active Document service can identify when a certain project has gathered for a meeting, and then actively display information that it has stored from earlier meetings. The service uses information from Context Shadow about project membership, the users' locations, what people are in the room and what services that is available in the room.

In more detail, service composition with the Active Document service is achieved through the following process: Context Shadow monitors people that are members of the same project as the document. Based on the information given by Context Shadow, the Active Document acquires pointers to locations where at least two project members are present for the moment. If it finds such a location, the Active Document assumes that there is a project meeting taking place. The Active Document now tries to migrate to that location by asking Context Shadow for the nearest execution environment available (this is a specific type of service). When it has migrated to a suitable host where it can execute, the Active Document queries Context Shadow for an appropriate public display resource inside the room. If such a service is available, the document utilizes that public service to display its contents. If someone clicks on the icon representing the Active Document service on the context sensitive service desktop, the document is notified about who has clicked. The Active Document then asks Context Shadow for a suitable resource to display itself on for that user. Of course, that resource does not have to be available on the same device as the one that the user clicked on.

6.2 Design approach 2: Using a context enhanced meta service description format

The way services announced themselves in the previous system, Context Shadow, was a bit unpractical, since each service had to act as a beacon against one or several context servers. In the ACAS architecture on the other hand, service information is compiled in larger sets, each representing a *service infrastructure*, where a service infrastructure could be something like an organization or a department etc. which might have its own system management group that could be responsible for maintaining these service collections.

A new meta service description format has been developed, that acts as an umbrella description of services implemented using different technologies, also providing means to add context description elements to the service description.

Pointers to the service descriptions are provided through the context information infrastructure in the form of simple context elements. The service description is also made available through a service announcer service, through which you make queries against the contextually enhanced service descriptions.

The ACAS system uses a searchable xml service description. Each local infrastructure maintains an xml page of the available services in the infrastructure. This page can be

queried using XQL. The key components for the service composition approach in the ACAS architecture thus becomes:

- Service announcements are collected into meaningful collections, e.g. an infrastructure service aggregator that provides information about services related to that specific place or organization.
- References to the collections of service descriptions are passed around as context information
- A new meta level service description format with context information is presented

6.2.1 A new service description format

The proposed service description format consists of some different parts. Firstly, each service is identified with a service name and a service ID:

```
<service name="ProjectorControl7514"
id="hostname.dsv.su.se:foobar"
targetNamespace="urn:acas:servicedescription"
xmlns:acas="urn:acas:servicedescription">
```

Next part consists of static metadata concerning this instance of the service, relating mostly to the function of the service. The dominant part of this section is a keyword list that can contain any number of keywords to be used when querying for services.

The "readable description" tag becomes very important in semi automatic service discovery tasks, where a user has to make a selection among a selection of services.

```
<serviceDescription>
<readable-description>"Projector control service for the
projector in the lab 7514"</readable-description>
<serviceType>"projectorControl"</serviceType>
<keywordList>
<keyword>projector</keyword>
<keyword>projectorControl</keyword>
</keywordList>
</serviceDescription>
```

Since the description format is intended to be a meta-level format it will only contain information that is related to the service discovery process, and not so much information regarding how the service is to be invoked, which methods it uses etc. The document must however contain some pointers to the instance of the service, or the description becomes pointless. So the following section contains information about how the service is implemented or which protocol the service uses for

communication, and some protocol-specific information on where the service (or more information about the service) can be found.

```
<serviceImplementation protocol="Jini"
xmlns:jini="jini-namespace" xmlns:upnp = "upnp-namespace"
xmlns:sip = "sip-namespace" xmlns:eheap =
"eventheap-namespace" xmlns:webservice =
"webservice-namespace" >
```

If it is a Web-service, the following section will contain:

```
<webservice:serviceURL>URL...</webservice:serviceURL>
<webservice:WSDL>WSDL-URL</webservice:WSDL>
```

If it is a Jini service the following section would instead contain:

```
<jini:serviceID>unique-id</jini:serviceID>
<jini:codebase>url</jini:codebase>
<jini:lookupServerURL>url</jini:lookupServerURL>
```

SIP:

```
<sip:sipAddress>projector7514@sip.dsv.su.se</sip:sipAddress>
<sip:sipProxy>hostname.dsv.su.se</sip:sipProxy>
```

Or UPnP:

```
<upnp:serviceURL>URL...</upnp:serviceURL>
<upnp:advertismentURL>URL...</upnp:advertismentURL>
```

Finally one of the main reasons of inventing a new description format is to make it possible to provide information about the context in which the service is embedded. This information is encoded in the same context information protocol as the context information for the sensor data used within the rest of the ACAS architecture, namely in the form of context element records. By using the same context representation, the same rule machinery can be used to process the service description documents to enable context aware service discovery.

```
<serviceContext>
<acas:contextelement>…</acas:contextelement>
<acas:contextelement>…</acas:contextelement>
</serviceContext>
```

The context information embedded with the document could e.g. contain information about the location of the service, if it is own by a person etc.

6.2.2 The Service aggregator

Once we have the service description it has to somehow be made available for queries. Therefore the Service Aggregator is added as a component of the system. The Service Aggregator has following functions:

- **Maintain a collection of service descriptions**: According to the previous discussion, it is a point to collect the services into sets, where the selection has some kind of semantic meaning, e.g. representing a location. The set of services could be assembled manually, or build on top of other service discovery techniques and make the compilation automatically
- **Handle queries:** The service aggregator is the entry point for queries about services.
- **Produce context information:** The aggregator must be connected to some kind of sensor system, so that it can be detected (or detect) a roaming user.

6.2.3 A Discovery Service implementation using Rain

Rain is a lightweight XML-based messaging system produced by Intel Research Seattle. The two core concepts in Rain are Services and Messages. Services represent the computational components of the system and they communicate via messages. In Rain, messages are pieces of XML and as such, allow services to interact using semi-structured data. In Rain, message delivery is asynchronous. That is to say, the message is not guaranteed to have been delivered by the time the send method returns.

Services in Rain find each other via a special Rain service called the discovery service. Services in rain interact with the discovery service in two ways. First, they send the discovery service their own unique id (called a ServiceID) and their advertisement. In Rain, an advertisement is just a piece of XML that can contain any semi-structured data the service wants. Second, services in Rain can query the discovery service using XPath queries to find other services. XPath is a language for addressing parts of an XML document, and is primarily used in the XSLT language.

In our system a Rain discovery service provides the querying interface towards the service descriptions. The service is running in the network and is accessed over a socket based Java API, where the queries on the services are expressed in the XPath language. If a service description matches the query, the service description document is provided to the querying application for further processing. The URL to the service aggregator is embedded in a context element record, and is combined with other context information by the infrastructure context manager.

6.3 Design approach 3: User managed service stitching

The two earlier approaches for dynamic service composition described in this thesis both focus on the fact that in order to achieve context dependent service discovery

you need to in some way relate a pointer to the service with some kind of contextual semantics. As was discussed earlier some of this meta data can be extracted automatically, like e.g. network properties etc., but the more crucial information, such as physical location, ownership etc. must be added manually. In the earlier designs this information was supposed to be added by the owner of the service, typically either an organization or an individual user. The major drawbacks with this approach is that a group of users that would use a specific set of services might span over several independent workplaces, domestic environments etc. In the Spots system the approach to this problem is instead that the end users are allowed to add pointers to services at the places where they reside.

Services in the Spots system are described using labels containing simple sets of key-value pairs. Such sets of key values are expressive enough to be able to express pointers to service instances encoded in most existing service oriented technologies, such as Jini, UPnP and Web Services etc. Service composition in this case typically means that one service subscribes for information about some specific type of services at the users current location.

6.3.1 Service stitching

The Spots system also reflects a slightly different view on how service composition is supposed to be achieved in terms of interoperability. In many of the visions and proposed architectures describing pervasive computing systems, the assumption is made that independent computing resources that are designed without knowledge of each other, still can interoperate in a meaningful way. Deployments of services in such environments is often a matter of providing generic service descriptions and assume that the future services that wish to interoperate will have enough capabilities to interpret the information in a correct way. The role of the end user is rarely mentioned with respect to this process.

A more reasonable approach is however that people will have to actively incorporate the new technologies into their everyday lives, and also to make it work in a meaningful way with existing technologies and services. So instead of thinking of pervasive technology as an existing (pre-designed, pre-configured) web of interconnected devices, it should be seen more like a patchwork that has to be actively stitched together to fit the situations where it is to be used. This *stitching* is about making two technologies that were not designed to work together, to do that, and in a meaningful way. The stitching is performed either by the end user or by a community of users that in some sense share the same problems and goals.

6.3.2 Example

The location sensitive TeamSpace application was created by adding location awareness to an existing application. The TeamSpace software (Shih et al. 2004) is a toolkit for building applications in interactive work environments with decoupled artefacts and services. It contains functionalities such as controlling mouse and keyboards of other computers as well as sending and displaying documents to other computers. By connecting it to the Spots system the application would automatically connect to recourses in the local environment when the user entered a new spot. The

modifications needed to achieve a dynamic location aware behaviour for this service was that a small wrapper layer had to be created around the client application. The wrapper layer communicates with the Spots infrastructure and subscribes for labels containing information about available TeamSpace servers deployed at the user's current location. Whenever it is provided with such a label, it extracts the server address embedded in the value part of the label, and tries to connect the client application to that server. For every location where TeamSpace servers exist, a spot has to be defined and a service specific label has to be added, providing the address to the Team Space server instance.

6.3.3 Sharing service information through user communities

A basic idea of the Spots system is that the activity of deploying services in the infrastructure should be a collaborative task. In this way one person may add information about a service at one place then share this information, so that when another person enters this location the service composition happens automatically.

Sharing of service information is enabled by allowing users to form and join *communities*. Normally these communities are formed around the usage of specific applications. If a user for example wants to start using the location aware TeamSpace service described above, she can also choose to join a TeamSpace community and get information about services that has already been deployed at different locations. In practice, to join a community means to set up a subscription for some specific label information with a *community server*. The subscription is created from your Spot client application and then propagates to your personal server, where a Subscription Manager is situated. The Subscription Manager can manage several running subscriptions in parallel, and handles the sharing in a bidirectional manner, so that any newly defined spots containing specific labels are also shared with the community.

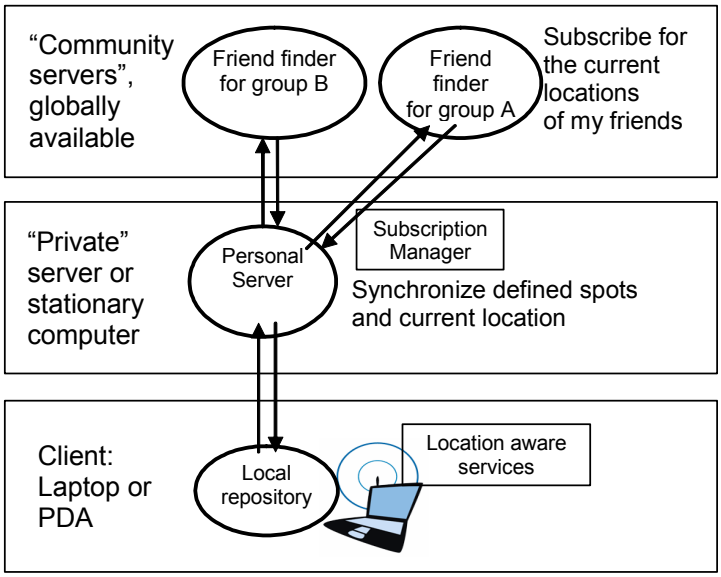


Figure 27. The three tire repository architecture of the Spots sharing mechanism

6.4 Evaluation of the design approaches in terms of the system requirements

Three different approaches to achieve context dependent dynamic service composition have been presented. Some example applications have been implemented as proof of concept.

6.4.1 Service adjustment requirements

Ideally a middleware for service composition should be able to be introduced without having to modify existing services. In most cases however the services will have to be adjusted to some extent to be able to benefit from the functionality of the middleware. The degree of adaptation required will however differ between different middleware designs.

In all the three systems presented here, the *resource* services (the services that are to be discovered) can be entirely unaffected by the introduction of the middleware. In all three systems these services are tied to the middleware by completely separate processes. The greater part of the changes is instead required on the consuming service part that is actually utilizing the middleware for the discovery process. In the Context Shadow system the consumer service has to conform to a specific Java interface, including a set of predefined service queries. The service thus has to know how to use the query API and also know what to do with the result. In the Spots system the service is provided information about available services through key value pairs embedded in an XML message that appears for every discovered location. In the ACAS architecture the interface against the service description is more open since you can use arbitrary XQL queries to access the information. The querying process will

result in protocol specific service descriptions or stubs so that the services then can be accessed entirely independent of the middleware system.

6.4.2 Service capability requirements

For many of the application scenarios described earlier, the middleware should support composition of fairly simple services. It is therefore interesting to see to what requirements the middleware puts on the services in terms of computational and reasoning capabilities.

The Context Shadow system is explicitly targeting composition of simple services, where the interaction with the middleware is constrained to a predefined API. In this way the services can perform context aware discovery just by making a single method call. The Spots system is constraining the task of service composition even more, by only providing information about services at the user's current location, thus putting low demands for any reasoning capabilities. The ACAS system on the other hand allows for more complex compositions but on the other hand requires more of the services. Not only must the services master the XQL querying to access the service description document, they must also be able to interpret the context elements attached to the service descriptions. This includes having access to the context model in which the context information is encoded.

6.4.3 Deployment overhead

If the deployment threshold of introducing new technology is too high it is a risk that it will never be realized. Deployment overhead includes both the amount of work required to introduce the actual middleware but also the amount of work needed to adapt existing services to work with the middleware.

For the Context Shadow system, you have to set up context servers for each person, place or group that is to be represented, and then tie sensors and services to each of these servers. If the number of servers and services is large this will become a fairly cumbersome task. The ACAS architecture scales a bit better since an entire local infrastructure can be compiled into one server component. Service information is also compiled in a similar way, making deployment and maintenance less resource demanding. The spots system is addressing these issues in two ways; firstly the system uses existing properties of the environments as sensor input (e.g. WiFi and Bluetooth signal strength profiles) reducing the need for extensive sensor installations. Secondly users are encouraged to perform a lot of the work of tying services to the infrastructure, and share this information through communities. In this way the deployment work is shared among the users of the system.

6.5 Comparison of approaches

If we compare the approaches described above, you can see that each approach is more or less suitable for different composition scenarios.

Context Shadow: With fairly high deployment costs and support for simple services, this system fits fairly small service environments where the composition is a matter of

establishing connections between light weight services in for example shared meeting environments.

ACAS: With higher demands on reasoning capabilities, and better scalability, this approach is more suitable for scenarios where you have more complex context aware services that want to get access to resources relevant to a user that is roaming between several high density service environments.

Spots: Suitable for simple bootstrapping tasks where applications on personal mobile devices automatically can connect to resources in the user's vicinity.

6.6 Chapter summary

In this chapter a number of a number of implementations have been presented that uses different mechanisms that in different ways supports context dependent service discovery. The mechanisms can be summarized as:

- Semantic clustering: A somewhat crude way of enabling context aware service discovery is to create clusters of services and connect them to various entities that can be discovered using sensors.
- A meta service description layer: By introducing a meta service description layer it is possible to extend existing services based on technologies like Jini, UPnP and Web Services, with complementary information regarding for example the physical environment in which the services are deployed.
- User community managed service stitching: The Spots system presents mechanisms that allow end users to connect services to specific locations, so that they will be discovered and/or triggered automatically next time the user enters the location.

Chapter 7
Engaging the users in context information middleware systems

In many discussions regarding user participation in context aware systems, there have been a focus on the application layer, discussing issues such as how the context information affecting specific application behaviours can be displayed to the user (Bellotti and Edwards 2001). But since parts of the interpretation and decision processes may happen in the middleware layer it is interesting to examine how different design choices in the middleware layer affect the issues related to user control and appropriation, and also how the middleware can be adapted in various ways. In this work we will try and revisit the role of the user with respect to the following problem areas, which were identified and motivated in chapter 3.4:

- **Ownership of context information:** Context information might be assembled from sources like your mobile phone, the usage of your computer or by specialized sensor equipment. Some of this information like the position of people's mobile phone is currently owned by companies and out of control for the user. By trying to collect context information describing the whereabouts of a user in a system that in some sense belongs to the user, the degree of control of the information can be increased. This increases the possibilities to have control over which information to share and with whom, and in what format.

- **Who assigns meaning to sensor information:** One aspect that is crucial in order to achieve user empowerment in context aware system is to allow users to have control over how incoming context information is interpreted. How can context aware systems be created that don't rely on fixed predefined models to determine how sensor information should be interpreted and what actions that should be triggered? One solution to this is to provide mechanisms that allow users to make their own interpretations of specific sensor data, as well as constructing their own models describing aspects of the world.
- **Supporting system adaptation and appropriation:** By making the users active participants in a continuous system redesign or adaptation, systems can be tailored to behave in accordance with the users' specific needs. This kind of adaptation can be supported by the system in different ways, but the basic idea is to keep the design open for alternative usages. The openness can be achieved on different levels, from low level such as providing open and general API:s and using open source licensing to higher level functionalities such as user interfaces that allows for tuning of system behaviour.

7.1.1 Who is the user?

When designing various forms of middleware there are two user groups that will have to be addressed. One group is the end users that will use the middleware mainly through the use of some application that uses some of the functionalities enabled by the middleware. The other group of users is the developers of these applications.

The users we are focusing on in this chapter are the first of these groups, the end users of the context aware applications. The distinction between these groups will however be blurred due to the suggested approaches where the end users take on some of the roles originally possessed by system developers.

7.2 User controlled context information management

By empowering the user through engagement we mean that the users should be given increased control over the system, in several aspects: Choosing a user centric system design in which the user is the owner and to some extent also the administrator of his/her part of the system is one way to handle problems related to ownership and maintenance. Having control over where the system is deployed and the possibility to shut it down at any time are key features to ensure user control. By designing a system where the user has increased control over the context information, privacy intrusion issues can be avoided or mitigated, for example by being able to control both with whom to share the information but control the format of the context information. All context information related to a person could then be assembled in this *personal system*, where decisions are made regarding with whom to share the information etc.

7.2.1 The notion of a personal context repository

Both in the Context Shadow system and in the ACAS system, each user has her own context repository, containing context information related to that user. In the Context Shadow system, the personal repository is a completely passive storage entity that can be examined through remote queries. The repository consists of a small java program that can easily be set up on an end users personal computer.

The ACAS architecture has a more complex system where the repository is embedded in a Context Management Entity (CME) that also contains entities for handling communication and subscriptions of context information. Even if the ACAS CME component is more complex than the Context Shadow repository, the idea is that it still should be able to run on a user's personal computer.

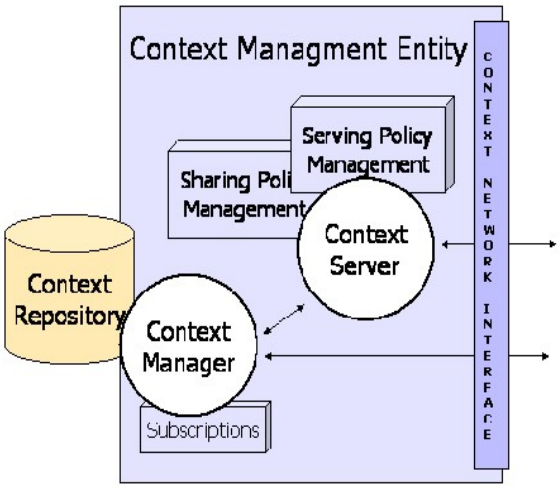


Figure 28. Schematics of the ACAS Context Management Entity

7.2.2 User controlled sharing of context information

In the Spots system the information about available locations is shared between users through the use of community servers. The servers are small footprint Java programs that easily run on a personal computer. A community server can thus be set up either by an individual user on her personal computer, or it could be running on a corporate server. A subscription with a community server is set up from the spots client GUI. In this GUI the users define whether to only pull information from the server or whether to also share their own location information. Several subscriptions can be setup to run simultaneously.

A community server could typically be shared by a limited number of users who share the usage of some specific application. Users of the FriendFinder application may join several community servers each representing a specific group of people that want to share information about their whereabouts with each other. With this architecture the users becomes empowered in the sense that they themselves can set up the limits of with whom to share information.

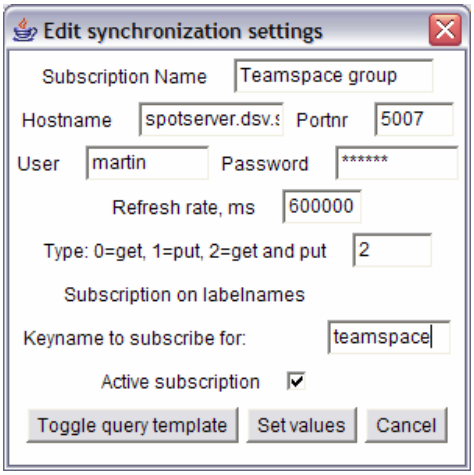


Figure 29. Setting up a subscription in the Spots client interface

7.3 Supporting appropriation by collaborative location tagging

As have been pointed out by for example Höök (Höök 2006) the usages of applications in the new ubiquitous computing environments become increasingly interweaved with our everyday social life. Mobile artifacts and applications are for example carried around and used in several different social contexts such as various work situations or settings with family or friends. This broad range of usage situations makes it hard for application designers to foresee all possible use cases, increasing the need for openness that allows for a certain degree of appropriation.

This is a problem for most applications in ubiquitous computing environments but becomes especially problematic in context aware applications due to the fact that most of these systems contain internal representations and assumptions about the situations of users as well as about various preferences. One problem that might appear in such situations is that the designer of the system has interpreted the sensor data in a different way than the end user would do.

An approach to deal with these problems is to involve the users in the processes of interpreting sensor information and choosing representation formats that makes sense in their current situation. In this work an approach is examined where users in a collaborative fashion can attach name labels to places.

7.3.1 Folksonomies as opposed to ontologies

The notion of folksonomies was originally coined by Vander Wal (Vander Wal 1995) as a way of describing a new type of services on the internet where end users provide keywords or 'tags' to describe various entities. Two widely cited examples of websites using folksonomic tagging are Flickr[1] and del.icio.us[1]. Where Flickr uses user specified

[1] Available at: http://www.flickr.com

tags to describe images, del.icio.us uses the same principle to describe web pages. The latter can be viewed as a kind of bookmarks manager, which is in no way a new type of service. What seems to be relatively new and different is the emphasis on user added keywords as a fundamental organizational construct. These keywords or tags make it possible for the users to describe and organize content with any vocabulary they choose. The primary function of the tags in these services is to support content retrieval. By browsing or making queries based on these tags you can get other results than an ordinary search engine would provide.

Folksonomies are often put forward as an alternative to or reaction against categorizations based on formal taxonomies or ontologies. One basic difference between these two approaches is whether a categorization or model is seen as something static that can be created in advance as in the ontology case, or if it is something that is created and updated as a part of an ongoing activity, as in the folksonomy case. Another difference lies in how the model is structured. Where ontologies often are based on hierarchical structures, a folksonomy creates an entirely flat namespace. Where hierarchical structures provide much more expressiveness and support for reasoning of various kinds, they are also more sensitive to changes. Adding a new and unforeseen entity might for example require restructuring of the entire hierarchy. Adding, removing or changing tags in a flat structure is much less problematic.

The namespace in a folksonomy is normally entirely open. Users are free to choose whatever tags they want to describe an entity. This will of course create all sorts of problems, for example due to usage of synonyms, abbreviations, misspellings etc. There are approaches to deal with these problems, for example to let the system provide suggestions on how to tag an entity, based on how other users have tagged it.

One can of course not say that one of these approaches is intrinsically better than the other. The interesting question to pose is instead for which application domains each approach is actually useful. In this paper we will examine how a folksonomy-like collaborative tagging can be applied to the application domain of location aware systems.

7.3.2 Making sense of location

Describing locations or places is a popular domain for different kinds of ontologies or taxonomies (Becker and Dürr 2004). Location can be described in terms of a continuous space or as a set of interrelated discrete places, as hierarchies based on geographical properties or based on more arbitrary divisions, such as addresses and zip codes. Locations can also be categorized based on the activities taking place there, such as "office", "pub", etc. The rich flora of different taxonomies to describe location suggests that it is a domain where it is hard to find one grand approach that will cover all special cases. Of course different application domains may have specific requirements that make it possible to agree on a certain taxonomy within that domain. But even within such specific application domains fixed ontologies might be hard to use due to for example how activities might change over time.

---

[1] Available at: http://del.icio.us

In this work we have been examining the application domain of location based presence sharing. The basic idea for this domain is that you have a sensor that can sense a user's location, and from this information provide information to other people about that users 'whereabouts'. In most of the existing systems of this kind the users will share information about their location to others by providing a map that points out the actual user's current location or a textual description using for example street names[1].

It is however not obvious that this is always the best approach to represent the users' whereabouts to other people. As have been pointed out for example in (Hong and Landay 2001), a person might want provide information about her location on different levels of detail depending on who requests the information.

Another issue might be that location information provided for example in terms of street names might not be meaningful to the receiver if the receiver of the information is not familiar with the area in question. In many cases it might also not be the geographical location you want to share with other people but rather some clues about what activity you are engaged in. To provide this kind of information the pure geographical properties of the location where you are situated is not as interesting as information about what *type of place* it is. The distinction between place and space has been thoroughly discussed by e.g. (Harrison and Dourish 1996), where in general the notion of place is seen as something much more semantically rich than a geographically constrained area. In for example (McCullough 2001), places are described in terms of spaces that support different types of activities. Some very common types of places are for example "home" or "work". This way of representing places are often referred to as semantic or symbolic models.

The use of semantic/symbolic location models has been studied in several previous efforts such as those by Roth (Roth 2003) and Pradhan (Pradhan 2000). Roth states that the idea of using semantic location models focusing on entailing the meaning of a location. The strengths of such a model lie in that they are easily extendable and in that you can make them both user-definable and machine as well as human interpretable. Some geographical properties of a location (street names, zip code etc.) can in most cases fairly easily be induced automatically from location sensor information. This is due to the fairly stable and well established mappings that exist between the coordinates you can receive from for example a GPS sensor and entities like streets, houses, zip code areas etc. The same kind of mapping is much harder to achieve for the place type properties described above for a number of reasons. The strongest reason for this is that the same place can be perceived differently from person to person. The place that a maid would characterize as "workplace" is for example perceived as "home" by someone else. Another reason is that the activities characterizing a place might change quite rapidly.

Another situation in which the approach with displaying location information on a map is not feasible is where the location sensors don't provide coordinates that can be used as keys against some existing map system. There are for example numerous ways of deducing that you are at or near a place based on sensors that can detect that you

[1] TeliaSonera, Friend Finder, http://friendfinder.telia.se/

are nearby some uniquely identifiable entity. Examples of such entities are for example GSM cells, WiFi or Bluetooth base stations etc.

To summarize the discussion above there are a number of properties of this application domain that makes it interesting to investigate a folksonomy based tagging approach:

- The perceived characteristics of a place are subjective.
- The characteristics of a place might change over time.
- People might not want to reveal their geographic location.
- There might be other properties of a place than geographical that are more useful to the receiver
- The sensors might not provide the key (coordinates) needed to acquire geographical map data.

7.3.3 An experiment on user controlled location tagging

In a lab assignment some groups of computer science university students were presented to the Spots system that has been described in further detail in chapter four. In the Spots system users can 'define' places using a GUI on their mobile device. Each place or spot is named without constraints by the user using any number of labels. When returning to a predefined spot, the labels are sent any application subscribing for location information. The most basic application to be used with the Spots system is a FriendFinder application that displays the current whereabouts of other people using the Spots system.

In the assignment the students had to define a number of spots and name them using a number of labels. One part of the assignment was to test the FriendFinder application and another part was to design a simple context aware application based on the Spots system by adapting an existing application. This application was the TeamSpace (Shih et al. 2004) software, which contains functionalities such as controlling mouse and keyboards of other computers as well as sending and displaying documents to other computers. To solve the assignment the students had to add some application targeted information to the spots and use it to adapt the TeamSpace software. The Spots system, when introduced to the students, worked only on laptops.

Being a very controlled use case occurring over a fairly short period of time, no general conclusions can be drawn from the usage of the system. One aspect which however is interesting to examine closer is how the students worked with the labelling of the Spots. To note when looking through usage logs from the students is that most student groups tended to use the Spots system exclusively to solve the lab assignment. Only a few groups/individuals defined Spots outside the campus building or used the system outside lab sessions, even though they were encouraged to do so (but not in any way forced). It is hard from this to say much about the naming conventions they would use in more realistic situations where a need of sharing information about your whereabouts to a group of people is more important, but looking at the spot names they have assigned to locations, some distinct categories could be discerned:

Table 3. Categorization of place labels chosen by the users

Naming based on local naming conventions:	"Rum 504" "Room 503“
Naming based on well known place names:	"Forum library" "World Class“
Naming based on personal or group identity:	"Jan's Room" "Kalle’s apartment"
Ambiguous naming:	"Hallway" "bus" "5th floor toilets“
Relativistic naming:	"Outside 503“
Naming according to place type or function:	"sushi" "Living room" "bus“
Naming not related to locations:	"Jeanette & Anna“
Activity related naming:	“Laptoplab402”
Differences in granularity:	"gary’s door" "BlueSofa2" "Stockholm"
Application targeted naming:	"85.224.129.193"|"TeamSpaceAddress“
Unknown meaning:	"wagon meeting"

Looking at this list, names in the category ‘Naming based on personal or group identity’ were common. (No exact list of how large a fraction of the total number of labels each category has been compiled since the usage does not reflect real life conditions. Application targeted naming would likely have an unrepresentatively high fraction due to the lab assignment conditions). This is something also shown in a similar analysis of usage of the GeoNotes system (Espinoza et al. 2001). Person and Fagerberg called this category ‘Expressiveness’ and commented that authors of these types of names apparently wanted to be seen or remembered by others (Persson et al. 2002). An alternative interpretation is that the authors of these names seem to like to ‘claim’ Spots as their own, similar to some sort of territorial marking resembling graffiti in its purpose. Another observation is that the names in some cases have got nothing at all to do with the location in which the spot is situated. In a system where you want to ‘keep track’ of each other there might be other aspects than location that could be more interesting to use to describe some ones whereabouts.

You can also reflect on the need of support for hierarchical structures, to address problems concerning ambiguity. Several names are general enough to be likely to re-occur within several different contexts, such as “5th floor meeting room”, a name that have to be accompanied by additional information unless it was obvious to the intended recipient which building that is referred to. As have been argued, it has never been a goal for the Spots system to be able to produce entirely unambiguous location information according to some fixed model. But for the Spots model to be useful, the naming must be meaningful within the smaller groups with whom each user is sharing the context information. We thus introduced a simple form of structure based on containment in the system, so that you can create “parent spots” carrying labels of e.g. city or building names, in order to minimize some of the risks of ambiguity. Finally,

what is most striking when examining the naming data is the great diversity in the ways people have chosen to name places. It is hard to imagine a coherent location model expressive enough to incorporate all the properties of the places described by these names.

7.4 Providing means for adaptation

As was described in chapter 3.4, adaptation of context information middleware can be achieved on different levels. Here we will provide examples of how support can be provided for system adaptation through on one hand the design of graphical user interfaces and on the other by providing means for programmers to adapt the system.

7.4.1 Supporting adaptation through user interfaces

Of the three different middleware systems presented in this thesis it is only the Spots system that is provided with a graphical user interface (Figure 30). Through this user interface the users can perform a variety of activities such as creating new spots and assigning labels to them, modify existing spots, set up and manage communication with various servers, Adjusting the sensitivity of the WiFi sensor, etc.

It is not always easy to draw a clear line between the activities of *using* the middleware and that of *adapting* it. The activity of creating new spots using the GUI is for example perfectly in line with the normal expected usage of the system and should thus not be accounted for adaptions of the system. The very similar task of modifying the labels of an existing spot could however be seen as an activity making changes to the underlying structure of the system for example in order to *adapt* the usage of the system to a new application domain.

The graphical user interface has another function besides providing means for changes of the system, namely as a tool for *inspection*. This is related to the concept of intelligibility that was identified as a crucial feature of a context aware system by Bellotti and Edwads (2001). By providing means for the end users to examine the state of the system it becomes easier to comprehend the system behavior. The Spots system allows for inspection of both the current state of the system though information about the spots that are currently being discovered, and more static properties like other defined spots and various settings for how information is being shared.

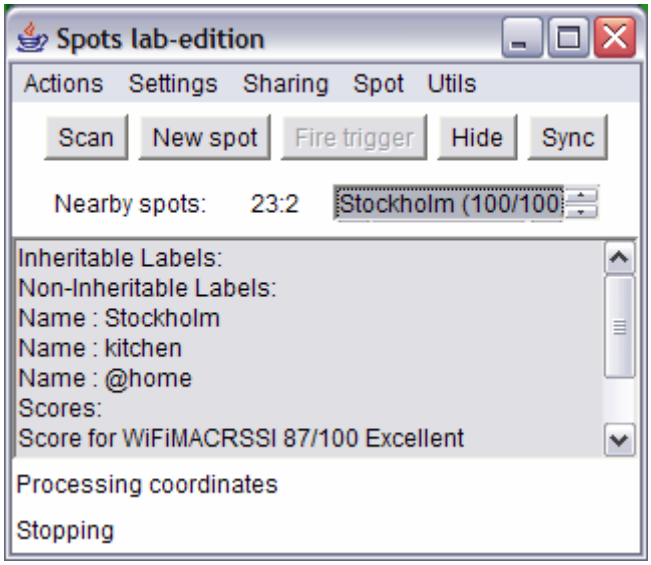


Figure 30. The main user interface of the Spots client

Experiments with adaptations through the user interface have been carried out in various lab assignments with university students, where they for example were to examine the precision of the location sensing while altering various parameters in the sensing process or to extend the usage of the system by combining it with other applications. No formal user study has been performed that have addressed this aspect specifically, but based on the performance of the students in combination with a fair amount of discussions during lab guidance sessions and examinations, some observations have still been made.

Local versus remote adaptations: From the graphical user interfaces most of the adaptations concerned the behavior of the local client running on the device where the user interface was presented. There were however some functions controlling the behavior of remote servers in the infrastructure, including for example subscriptions that can be set up between servers in the system. It turned out that many users had a hard time to comprehend these functions of the system in a correct way. A common mistake was the belief that the subscriptions that they created were tied to the local client instead of to a remote server.

One conclusion that was made based on this observation was that any backend entity that is to be adapted will have to be more explicitly represented in the user interface, and that the user interfaces controlling the different entities should be clearly separated. In the next version of the system the adaptation of the backend servers will be performed through user interfaces provided through internet browsers. In this way we believe that it will become clearer that the entities being modified is not situated on the local device.

Open ended versus fixed options: As was discussed in chapter 3.4 one of the major reasons to provide means for adaptation of the middleware is to make it possible to adapt the system in accordance with new requirements arising from new and unforeseen applications that are built on top of the middleware. In general the adaptations that can be made through a GUI will however to some extent have to be foreseen in advance. There is a specific selection of system variables that is presented

and that can be modified through the interface. If too many variables were to be included the GUI would become cluttered and hard to use. In many cases this problem is addressed by software applications by providing configuration files containing a richer set of variables to modify.

During the work in the lab settings there were several requests for alterations of system behavior that could not be performed without additional programming and modifications in the user interface. One parameter within the Spots system that is left a bit more open is the labels that can be attached to the spots. In one sense this parameter can be described in terms of open sets of key-value pairs, which are used in the communication with other system components. In some of the applications developed during the lab sessions these labels were adapted in accordance with the specific requirements of applications that were to be combined with the Spots system. One example is when URLs were encoded in the labels in order to create a location sensitive web browser. For this approach to be useful there must however be other system components that can respond and understand any new property descriptions. Even if there exist system components whose standards one can chose to adhere to, some tinkering in the form of programming is needed in most cases to stitch these two systems together.

7.4.2 Supporting adaptation through programming

In chapter 3.4 the notion of *hacker style adaptation* was introduced to describe a bit more advanced approach adapt some existing technology and tweak or extend it to make it behave in ways it was not designed for. A couple of arbitrary examples of such hacker style appropriation are for example attempts to install Linux on iPod[1] devices, or to get a Nintendo Wii remote control to control applications on your PC[2]. There are also a great number of examples of extensions or plugins to extend existing software applications, which are being shared over the internet often as open source.

This way of adapting a computer system requires a bit more technical skills, typically (but not necessarily) some kind of programming skills. There might be several underlying reasons for doing these adaptations, ranging from actual needs for some specific functions to more to social aspects such as gaining status in communities where technical skill is appreciated.

The ability to organize into communities seems to be a crucial factor for the success of these endeavors. These communities provide means for communication in the forms of forums, web pages and source code repositories. These communication channels are used on one hand for collaboration in order to solve the actual problem and on the other hand to support for dissemination of the solution to people sharing the need for a solution but lacking the skills to do it themselves.

Providing various forms of support for this kind of community based collaboration is important in order to realize some of the visions of ubiquitous computing. Consider for example the vision of interoperating devices and application. Instead of thinking of

[1] http://www.ipodlinux.org

[2] http://www.wiili.org

these pervasive technologies as an existing (pre-designed, pre-configured) web of interconnected devices, it should be seen more like a patchwork that has to be actively stitched together to fit the situations where it is to be used. This *stitching* is about adapting technologies that were not designed to work together, to do that in a meaningful way. In chapter 6.3 some approaches were presented that addressed the specific problem of achieving service discovery through a partially manual stitching approach.

Here we will instead focus more broadly on how support can be provided to adapt and extend context information middleware systems through means of *programming*. Making the system a bit more open for programmers can be beneficial to the end users even if they don't have any programming skills. By encouraging the organisation into joint developer and user communities people sharing the same needs or wishes for specific functionalities can form groups where a few persons with programming skills can create solutions that will benefit all the users in the group.

Middleware adaptation through programming can be supported in different ways. By making the source code publicly available the behaviour of the system can be completely altered. The major drawbacks with this approach are that even smaller alterations might become complex tasks and that alterations of the core parts might render the system incompatible with other instances. Another approach is to provide programming APIs to give access to the system internals. The API approach constrains the degree to which the system can be altered, but makes the programming task much easier.

In the Spots system several different APIs are provided whereby the system can be modified and extended. As have been shown in chapter 4.3 several applications have been built on top of the Spots middleware based on the provided APIs. Where most of these applications were developed in line with the foreseen application domains there have been cases where the system has been adapted to scenarios quite far from the original ideas. One example of such an adaptation is the Physical Cursors application which was presented in section 4.3.4 In this application the spots no longer represent places in the sense of rooms or other geographical areas, but are instead tied to RFID tags. Each spot can thus represent a device in an interactive room or tangible action cards representing certain actions that will be performed when detected. This rather extensive change in application domain could be achieved without changing the original system architecture. Much of the application semantics could be encoded in the spot labels where a specific taxonomy where defined and instantiated through the existing Spots GUI. An example of a minor adaptation of the Spots system was an extension of the system by which geographical coordinates can be added to spots using a user interface that allows users to place predefined spots on a map (Figure 31).

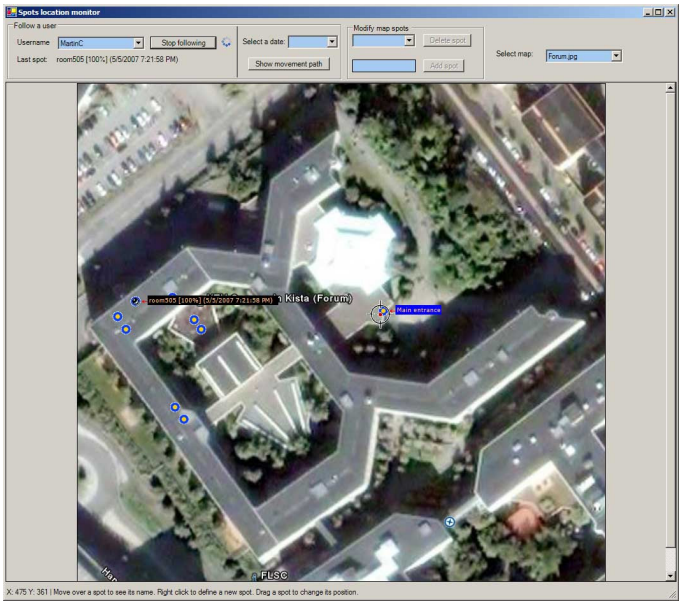


Figure 31. Screenshot from an extension to the Spots system in which geographical coordinates can be attached to previously defined spots

An attempt was also made to try and examine the degree of adaptability of the Spots system. As part of a lab assignment in a course in ubiquitous computing, groups of students had to present an application scenario together with some partial implementations where the Spots system was encouraged to be part of their design. The lab was divided into two phases, one where the students were to provide an application idea within the scope of location aware computing, and a second part in which they were to implement parts of their ideas. After the groups had presented their system designs they were presented with a questionnaire asking questions with respect to their experiences of the Spots system. A hypothesis was that the open internal representation format for location information would make it easy to adapt the system to different application scenarios.

After the course ended 18 questionnaires were handed in. The results from the questionnaire showed a fairly broad range of application scenarios, ranging from location based notifications to messaging systems and games. Most of the groups also claimed that they were able to adapt the Spots system to realize their initial application ideas. The problems reported with the Spots system had mainly to do with system performance and location precision and not so much about whether the architecture imposed any constraints. There is however very hard to make any generalisations based on these results, mainly because the students in the experiment had been presented with the Spots system before presenting their application ideas and thus probably adjusted their ideas to the properties of the system.

7.5 Chapter summary

In this chapter several approaches have been examined whereby the users of context information middleware systems extend their roles as passive users to instead take a more active part in the maintenance and evolution of the middleware system.

User centric system architectures have been presented that allows for end users to get an increased control over the dissemination and format of their own context information.

A mechanism has also been presented that allows user communities to express information about their current location using their own preferred vocabulary, using a method of "tagging" of places that resembles the concept of folksonomies that have been used in various web applications. This approach is believed to support the appropriation of the system. Study of the usage of the system shows a great richness in how users tag various places.

Finally various approaches to support system adaptation have been presented, both through making the middleware available through a graphical user interface and by making it adaptable by means of programming.

Chapter 8
Concluding remarks

In this thesis research is presented within the areas of Ubiquitous Computing and more specifically on Context Aware and Service Oriented systems. The thesis touches on many subjects, such as phenomenological views on sensing and perception, the role of the user in context information middleware, supporting and modeling service oriented systems, supporting context aware service discovery, representation of context information, supporting adaptability and appropriation, etc. The main focus however lies on the design of context information middleware, which can be described as support systems for the creation of context aware applications.

8.1 Summary of results

8.1.1 The USE framework to describe ubiquitous service environments

A framework has been introduced whereby ubiquitous computing environments based on service oriented software architectures can be modeled and analyzed. The framework centers around the idea that at any given moment in time a distributed computing system can be described in terms of sets of services spread out over several devices and composed to support specific activities. The framework can be used to describe any distributed system but is specifically targeting systems based on service oriented system architectures.

8.1.2 Implementations of context information middleware

This work has presented three different context information middleware systems, which each have been implemented and tested to different extents:

- The Context Shadow System, mainly targeting issues of context aware service discovery. A fully implemented system that has been put in successful use in applications targeting co-located collaboration.
- The ACAS architecture, targeting issues of representation, communication and transformation of context information. A larger system architecture that have been piecewise implemented and tested.
- The Spots system, focusing on active user participation with respect to system maintenance and interpretation of context information. A fully implemented system that have been put in successful use in applications for presence sharing, local collaboration, location sensitive reminders and configuration of interactive environments.

The main results with respect to these systems stems from the examinations of three key design issues for the design of context information middleware systems.

8.1.3 Means to improve CIM system versatility

From the analysis of the generality problem, a number of mechanisms were identified that in different ways addresses the question of how to achieve versatility in a context information middleware system. These mechanisms have also been implemented and tested in various combinations.

Mechanisms to avoid encoding semantics into the system

One way to achieve versatility in a CIM is to avoid hard wiring any semantics into the middleware layer. A specific solution has been presented that achieves this by contains a combination of two mechanisms:

1. Context Elements: A generic context information description format that contains mostly structure and very little semantics.
2. A context information refinement system with application specific transformation rules that are dynamically inserted into the middleware.

With these mechanisms you get the benefits of having computation taking place in the middleware in combination with a versatile and application independent system.

End users influence on internal representations of context information

In many cases there is not only the application domain that determines how the context information should be represented in the system but also subjective preferences from the end users. An example of this is how we choose to represent location, where certain names might make sense for one person but not for another. In the Spots system this was addressed by allowing end users themselves to assign labels describing places. In this way the representations used in the system can be adapted to fit not only new application domains but also to individual usage patterns,

making it easier for end users to appropriate the technologies into their existing practices.

8.1.4 Mechanisms for context aware service discovery

Three different mechanisms to achieve context aware service discovery have been presented with corresponding implementations and tests.

Semantic clustering

A somewhat crude way of enabling context aware service discovery is to create clusters of services and connect them to various entities that can be discovered using sensors. This approach was successfully implemented in the Context Shadow system. The most obvious example of this mechanism is to have clusters of services connected to locations, but experiments have also shown that it is possible to successfully cluster services to entities representing situations that can be deduced from sensor information. An example of this was the assignment of services to project groups, where the services belonging to the group for example can be discoverable when a critical amount of group members gathered in a shared space.

A meta service description layer

Another mechanism for service discovery focuses on the fact that services are often connected to various organizational infrastructures. By introducing a meta service description layer it is possible to extend existing services, using techniques like Jini, UPnP and Web Services, with complementary information regarding for example the physical environment in which the services are deployed. An XML format for such a meta service description have been presented, that makes it possible to combine pointers to the service instances with sets of context information elements. Implementations and experiments with remote queries using QXL show that it is easy and quick to query for services based on available context information.

User community managed service stitching

Often software services are globally available on the internet. Which services that are perceived to be useful in any given situation is something that differ between people and over time. The Spots system presents mechanisms that allow end users to tie (stitch) pointers to services to specific locations, so that they will be discovered and/or triggered automatically next time the user enters the location.

A complementary mechanism that has been developed is the ability to share the newly created location-service combo with other people. This is achieved using a web of community servers through which these service-location pairs are shared. Sharing takes place within smaller communities of people sharing a common physical environment or the usage of some specific applications.

8.1.5 Mechanisms extending the user's role in CIM systems

By incorporating end users in issues related to the design and maintenance of context aware systems, several problems can be alleviated.

Ownership of context information

Sensor information regarding the activities of a person can in many cases be obtained using personal and mobile technologies like mobile phones. Today, much of this information is owned by the companies owning the infrastructure. This thesis has explored approaches whereby users can assemble context information in repositories that are owned and controlled by the users themselves. Several system designs and implementations of *personal repositories* have been presented and implemented.

Community driven system management

A central mechanism for encouraging end users to participate more in system management issues is to provide means for users to organize themselves into *user communities*. This will have several advantages. Firstly it becomes possible to use community specific representations of context information, with the benefits of versatility and adaptability described above. Secondly by organizing into groups with shared interests you can create situations where a small effort from each person is enough to keep the system updated.

Support for middleware adaptation

Various approaches to support system adaptation have been presented, both by making the middleware available through a graphical user interface and by making it adaptable by means of programming. User experiments with extensions and adaptations of the Spots system indicates that an open ended internal representation format in combination with a graphical user interface whereby these representations can be modified makes the system flexible with respect to how well it can be adapted to various application domains.

8.2 General implications

During the work with this thesis some issues have come to stand out as more important than others:

Context aware computing is not about describing the "true situation" of a user. There are just too many possible truths and situations to describe. Instead the sensed pieces of context information should be seen as being purely instrumental, with respect to certain behaviors that are sought. The effects should always be determined in relation to a specific application. It is the usage of the application that determines the usefulness and meaning of the sensed information. One implication that this has to the design of context information middleware is that there should be support for parallel and possibly inconsistent interpretations and representations of the same sensor data.

Engage the users. For context aware computing applications to be successful the users will have to be engaged in various aspects of system design and maintenance. For

users to be able to appropriate the context aware applications they must be given control over the ways the sensor information is interpreted by the system as well as control over the actions taken based on the information.

Support user communities. There are several reasons to support the organization of users into user communities.

- It becomes much easier to agree on community specific standards for example with respect to how to represent context information when you don't have to generalize over too broad groups and interests. Support can be provided to let these standards evolve within the group like other social protocols.
- You can create incentives for users to do additional work on for example development and maintenance of the systems, by creating groups with common goals and interests.

Support appropriation: The exact usage forms of a system might be hard to foresee. Therefore the systems should be designed with a certain *openness* making it easy to adapt the systems to new usage areas. Such openness can be achieved by providing means for adaptability for example in the forms of openly available source code or programming APIs. Important is also to try and avoid internal representations that creates unnecessary restrictions on the usages of the system.

8.3 The future of context aware computing

An estimation of the future development of context aware computing can roughly be divided into the following trails:

1) *Advanced recognition in specialized domains*: Advances in the analysis of sensor data through various machine learning approaches makes it possible to draw more accurate conclusions based on rich sensor data. These applications will however still be restricted to fairly stable and well defined application domains. Expect to see an increase of context aware system monitoring applications within various industrial settings.
2) *Simple mobile phone based services*: Mobile phones are becoming increasingly more powerful and equipped with more sensing capabilities. Expect an increasing use of location based services using location information for example in order to simplify the interaction on a web page by filtering out information that are relevant to your current location.
3) *Community based systems*. With the continuous trend of Web 2.0 with users as content providers we will see an increasing use of systems similar to *war driving* where users collaboratively detect open WiFi networks and share the coordinates on the internet. Specialized simple applications will be developed to support the interests of specific user groups.
4) *Context tagging*. Adding context information as metadata to various forms of digital content is both simple and useful. The metadata will mainly be used to support various forms of queries. The non-intrusive behaviour also creates

high acceptance factor among users. Especially interesting for information published on the web. A current example is the Merkitus-Meaning[1] application where mobile phone images is tagged with context information and uploaded to Flickr.

5) *RFID based systems*. Cheap technology and simple to use. The broad usage of this technology within industry will spill over into the realms of personal computing.

Predictions of the future however in most cases turn out wrong. A more probable scenario is that a completely unforeseen application will appear which will be appropriated in unexpected ways by some teenagers turning it into the ultimate killer application.

[1] http://meaning.3xi.org/

Bibliography

Abowd, G. D., Atkeson, C.G., Hong, J., Long, S., Kooper, R. and Pinkerton, M. (1997). Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks* 3(5): pp. 421-433. October 1997

Bahl and Padmanabhan, (2000). RADAR: An In-Building RF-Based User Location and Tracking System*, Proceedings of IEEE Infocom 00*, April 2000.

Barkhuus, L. and Dey, A., (2003). Is context-aware computing taking control away from the user? Three levels of interactivity examined, *in Ubicomp 2003: Ubiquitous Computing*, vol. 2864, Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2003, pp. 149-156.

Beigl, M. (2000). MemoClip: A location-based remembrance appliance. *Personal Technologies*, 4(4):230-233, September 2000.

Bellotti, V., and Edwards, K. (2001). Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Journal of Human-Computer Interaction* 16, 2001 pp. 193-212.

Brotherton, J. A. Abowd, G. D. (1998). Rooms take note: Room takes notes! *In Proceedings of the 1998 Spring AAAI Symposium on Intelligent Environments,* pages 23-30, 1998. Published as AAAI Technical Report SS-98-02.

Brown, P. J., Bovwey, J. D. and Chen, X. (1997). Context-aware Applications: from the Laboratory to the Marketplace. *IEEE Personal Communications* 4, 5 October 1997, 58–64.

Becker, B. and Dürr (2004). On location models for ubiquitous computing, *Personal Ubiquitous Computing* 9, 2005, Springer-Verlag, 20-31.

Caswell, D. and Debaty, P. (2000). Creating Web Representations for Places. *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, Bristol, UK, 114-126.

Chen, H., Finin, T., and Joshi, A., (2003). Using OWL in a Pervasive Computing Broker. *In Proceedings of Workshop on Ontologies in Open Agent Systems (AAMAS 2003).*

Cheng, Chawathe, LaMarca and Krumm, (2005). Accuracy Characterization for Metropolitan-scale Wi-Fi Localization*, Proceedings of the ACM/USENIX Conference on Mobile Systems, Applications and Services (MobiSys)*, Seattle, WA, January 2005.

Conner, S., Krishnamurthy, L. and Want, R. (2001). Making Everyday Life Easier Using Dense Sensor Networks. *In Proceedings of ACM Ubicomp*, Atlanta Georgia.

Clark., J., (1999). *Xml transformations (xslt) version 1.0*, November 1999.

Christiaens, S., (2006). Metadata mechanisms: From ontology to folksonomy...and back. *In: OTM Workshops 2006. Lecture Notes in Computer Science*, Springer-Verlag (2006) 199-207.

De Bruijn, J. (2003). *Using Ontologies – Enabling Knowledge Sharing and Reuse on the Semantic Web.* Tech. Rep. Technical Report DERI-2003-10-29, Digital Enterprise Research Institute (DERI), Austria, October 2003.

Dey, A.K. (2000) *Providing Architectural Support for Building Context-Aware Applications*, Ph.D. thesis, December 2000, College of Computing, Georgia Institute of Technology.

Dey, A.K., Hamid, R., Beckmann, C., Li, I., Hsu, D. (2004). a CAPpella: Programming by Demonstration of Context-Aware Applications. CHI 2004, ACM Conference on Human Factors in Computing Systems, *CHI Letters* 6(1): April 24-29, 2004.

Dey, A.K., Abowd, G. and Salber, D.(1999). A Context-Based Infrastructure for Smart Environments. *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, Dublin, Ireland. 114-128.

Dey, A.K. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing* 5(1).

Dey, A.K., Hamid, R., Beckmann, C., Li, I., Hsu, D.,(2004). a CAPpella: Programming by demonstration of context-aware applications. *In: ACM Conference on Human Factors in Computing Systems (CHI)*, Vienna (2004)

Dourish, P. (2001). *Where the Action Is, The Foundations of Embodied Interaction.* Cambridge, Massachusetts, MIT Press.

Dourish, P. (2004) What We Talk About When We Talk About Context. *Personal and Ubiquitous Computing*, 8 (1). Springer-Verlag. pp 19-30

Duranti, A., and Goodwin, C., eds. (1992). *Rethinking context. Language as an interactive phenomenon.* Cambridge University Press.

Edwards, K. W. Bellotti, V., Dey, A. (2003). Stuck in the middle: Bridging the gap between design evaluation and middleware. *Proceedings of the 2003 Conference on Human Factors in Computing Systems (CHI 2003),* Fort Lauderdale, FL, April 5-10.

Espinoza, Fredrik. (2002). *Individual Service Provisioning.* Stockholm: Akademitryck AB, Doctoral Thesis, Stockholm University, Dept. of Computer and Systems Sciences.

Espinoza, Persson, Sandin, Nyström, Cacciatore, Bylund, (2001). *GeoNotes : Social and Navigational Aspects of Location-Based Information Systems,* in Abowd, Brumitt, Shafer (eds.) Ubicomp 2001: Ubiquitous Computing International Conference, Atlanda, Georgia, September 30 – October 2, Berlin : Springer, pp 2-17

Fox, A., Johanson, B., Hanrahan, P., Winograd, T. (2000). Integrating Information Appliances into an Interactive Workspace. *IEEE Computer Graphics & Applications, 20(3*)

Gellersen, H-W., Schmidt, A. and Beigl, M.,(2002) Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts. *ACM journal Mobile Networks and Applications (MONET*), Vol. 7, No. 5. October 2002.

Greenberg, Saul (2001): Context as a Dynamic Construct. In Human-Computer Interaction, 16 (2) p. 257-268

Gu, T., Wang, X. H., Pung, H. K., and Zhang, D. Q., (2004). Ontology Based Context Modelling and Reasoning using OWL. *In Proceedings of the 2004 Communication Networks and Distributed Systems Modelling and Simulation Conference (CNDS2004)* (San Diego, CA, USA, January 2004).

Guttman, E. (1999). Service Location Protocol: Automatic Discovery of IP Network Services. *IEEE Internet Computing.* 4(4): p. 71-80.

Harrison, S., & Dourish, P. (1996). Re-Place-ing Space: The Roles of Place and Space in Collaborative Systems. In Q. Jones, and C. Halverson, (Eds) *Proceedings of CSCW'96: ACM Conference on Computer Supported Cooperative Work* (pp.67-76), Cambridge MA, ACM Press.

Held, A., Buchholz, S., AND SCHILL, A. (2002). Modelling of context information for pervasive computing applications. *In Proceedings of SCI 2002/ISAS 2002.*

Hightower, LaMarca and Smith (2006), Practical Lessons from Place Lab. *IEEE Pervasive Computing,* vol. 5, no. 3

Hill, H., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., Pister, K.S.J. (2000) System Architecture Directions for Networked Sensors, *Architectural Support for Programming Languages and Operating Systems*, pp. 93-104, 2000

Holmquist, L.E. Mattern, F. Schiele, B. Alahuhta, P. Beigl M. and Gellersen, H.W. (2001). Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts, *Proc. of UBICOMP 2001*, Atlanta, GA, USA.

Howes, T. A., Smith, M. (1995). *A scalable, deployable directory service framework for the internet.* Technical report, Center for Information Technology Integration, Univerity of Michigan.

Huang, A.C., Ling, B.C., Ponnekanti, S., Fox, A. (1999). Pervasive Computing: What Is It Good For? *Proceedings of the Workshop on Mobile Data Management (MobiDE)* in conjunction with ACM MobiCom '99, Seattle, WA.

Fielding, R., Gettys, J., Mogul,J., Frystyk, H., Masinter, L., Leach,P. and BernersLee., T., (1999). RFC 2616: *Hypertext transfer protocol* – http/1.1, June 1999.

Gong. L. (2001) JXTA: A network programming environment. *IEEE Internet Computing*, v. 5, pp. 88-95.

Greenberg, Saul (2001): Context as a Dynamic Construct. In Human-Computer Interaction, 16 (2) p. 257-268

Gustafsson, H. and Jonsson, M. (1999). Collaborative Services Using Local and Personal Facts. *Proceedings of the Personal Computing and Communication Workshop,* Lund, Sweden.

Haeberlen, Rudys, Flannery, Wallach, Ladd and Kavraki, (2004). Practical Robust Localization over Large-Scale 802.11 Wireless Networks, *Proceedings of Mobicom 2004*

Hardian, B., Indulska, J. and Henricksen, K., (2006). Balancing Autonomy and User Control in Context-Aware Systems - a Survey. *In 3rd International Workshop on Context Modelling and Reasoning (CoMoRea).* IEEE Computer Society, March 2006.

Henricksen, K., Indulska, J., McFadden, T., Balasubramaniam, S., (2005). Middleware for distributed context-aware systems. *International Symposium on Distributed Objects and Applications (DOA)*

Hong, J., I., and Landay J., A. (2001). An Infrastructure Approach to Context-Aware Computing, *Human Computer Interaction,* 2001, Volume 16, pp287-303.

Horvitz, E. (1999). Principles of mixed-initiative user interfaces. *Proc. CHI '99*. 159-166.

Höök, Kristina (2006) Designing Familiar Open Surfaces, *In Proceedings of NordiCHI 2006* Oslo, Norway, October 2006, ACM press.

Ishii, H., Ullmer, B. (1997) Tangible bits: towards seamless interfaces between people, bits and atoms. *In proceedings of CHI'97*, pages 234 -- 241, 1997.

Microsoft Corporation, (2000) *Universal Plug and Play Device Architecture*, White Paper, Version 1.0, June 6, 2000. Available at: http://www.upnp.org

Jonsson, M. (2002). Context Shadow: An Infrastructure for Context Aware Computing. *Proceedings of the Workshop on Artificial Intelligence in Mobile Systems (AIMS),* in conjunction with ECAI 2002, Lyon, France.

Jonsson, M., Jansson, C., Lönnquist, P., Werle, P., Kilander, F. (2002). *Achieving Non-Intrusive Environments for Local Collaboration,* Technical Report 2002-021, Dept. of Computer and Systems Sciences, Stockholm University/KTH.

Jonsson, M and Mattsson, J. (2002) *Building extendable sensor infrastructures for pervasive computing environments*, Technical Report 2002-019, Dept. of Computer and Systems Sciences, Stockholm University/KTH.

José, R. and Davies, N. (1999). Scalable and Flexible Location-Based Services for Ubiquitous Information Access. *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC'99)*, Karlsruhe, Germany. 52-56.

Klensin, J. (2001). RFC 2821: *SMTP: Simple mail transfer protocol*, April 2001.

Krumm, J. and Hinckley, K. (2004), The NearMe Wireless Proximity Server, *Lecture Notes in Computer Science : UbiComp 2004: Ubiquitous Computing*, pp. 283-300.

Lamarca A, Chawathe Y, Consolvo S, Hightower J, Smith I, Scott J, Sohn T, Howard J, Hughes J, Potter F, Tabert J, Powledge P, Borriello G, Schilit B, (2005). Place Lab: Device Positioning Using Radio Beacons in the Wild, *Pervasive Computing*, pp. 116-133.

Lamming, M. and Flynn, M. (1994). Forget-me-not: Intimate Computing in Support of Human Memory, *in Proceedings of International Symposium on Next Generation Human Interface* 1994.

Lieberman, H. (2001). *Your wish is my command: Programming by example*. Morgan Kaufman. 2001.

Long, S., Kooper, R. Abowd G, D. Atkeson C, G. (1996). Rapid prototyping of mobile context-aware applications: The Cyberguide case study. *In the Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking (MobiCom '96)*, pp. 97¬107, White Plains, NY, ACM. November 10-12, 1996.

Marmasse, N. and Schmandt, C.,(2000). Location aware information delivery with commotion, *HUC 2000 Proceedings,* pp 157-171, Springer-Verlag.

McCarthy, J., (1987). Generality in artificial intelligence, *Communications of the ACM*, 30(12):1030-1035.

McCarthy, J., (1993). Notes on Formalizing Context., *In Proceeding of The Thirteenth International Joint Conference on Artificial intelligence*, Pages 555-560, Chambery

McCarthy, J. & Buvac, S., (1997). Formalizing Context (Expanded Notes) *Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language*

McCullough, Malcolm (2001). On Typologies of Situated Action. *Human-Computer Interaction* 16(2-4): 337-349.

Merleau-Ponty, M. (2002) *Phenomenology of Perception* trans. by Colin Smith, (New York: Humanities Press, 1962) and (London: Routledge & Kegan Paul, 1962) translation revised by Forrest Williams, 1981; reprinted, 2002)

Moran, P. T., and Dourish, P., (2001). Introduction to This Special Issue on Context-Aware Computing. *Human-Computer Interaction* 16: 87-95.

Nardi, B.A., ed.(1996) *Context and Consciousness: Activity Theory and Human-Computer Interaction*, MIT Press: Cambridge, Mass.

Norman, D. (1999). *The invisible computer.* Cambridge University Press

Novatchev, D. (2003). Functional programming in xslt using the fxsl library. *Presented at Extreme Markup Languages 2003*.

Papazoglou, M. and Georgakapoulos, D. (2003). Service oriented computing. *Communications of the ACM, 46(10)*:24--28, October 2003

Pascoe, B. (1999) *Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun*. Salutation Consortium, White Paper.

Pascoe, J. (1997) The Stick-e Note Architecture: Extending the interface beyond the user. Proc. *Intelligent User Interfaces*, 261-264.

Persson, P. and Fagerberg, P., *GeoNotes : a real-use study of a public location-aware community system,* SICS Technical Report T2002:27, ISSN 1100-3154, IRSN:SICS.T-2002/27-SE

Pham, T., Schneider, G. and Goose, S. (2000). Exploiting Location-Based Composite Devices to Support and Facilitate Situated Ubiquitous Computing. *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, Bristol, UK, 2000. 143-156.

Pradhan, S. (2000), Semantic Location, *Personal Technologies* 4, pp 213-216, Springer-Verlag.

Rhodes, B. (1997). The wearable remembrance agent: a system for augmented memory. In: *Personal Technologies, Special Issue on Wearable Computing*, Springer, Vol. 1, No. 4, pp. 218-224.

Roach, A. (2002). *RFC 3265: Session initiation protocol (sip)specific event notification*, The Internet Society, Network Working Group, June 2002

Roth (2003), Flexible Positioning for Location-Based Services, *IADIS International Journal of WWW/Internet,* Vol 1, Nr 2 pp 18-32

Salber, D., Dey, A.K., & Abowd, G.D. (1999). The context toolkit: aiding the development of context-enabled applications. *Proceedings of the CHI99 conference on Human factors in computing systems: the CHI is the limit,* 434-441.

Salvador, T. and K. Anderson. (2003) Practical Considerations of Context for Context Based Systems: An Example from an Ethnographic Case Study of a Man Diagnosed with Early Onset Alzheimer's Disease. *In proceedings of Ubicomp 2003*. pp. 243-255.

Scott, S., et al.(2002). Investigating human-computer optimization. *Proc. CHI 2002*. 155-162.

Schilit, B., N. Adams, N., and Want, R. (1994). Context-aware computing applications. *First International Workshop on Mobile Computing Systems and Applications,* 85-90. (Santa Cruz, CA, US, 1994).

Schmidt, A., Takaluoma, A. and Mntyjrvi, J.(2000) Context-Aware Telephony over WAP. *Personal Technologies Volume* 4(4), September 2000. pp225-229

Schmidt, A., and Van Laerhoven, K.. (2001). How to Build Smart Appliances? *IEEE Personal Communications* 8(4), 66-71.

Schmidt, A. (2002) *Ubiquitous Computing–Computing in Context*. A thesis submitted to Lancaster University for the degree of Ph.D. in Computer Science, November, 2002

Schutz, A. (1975). *Collected Papers III. Studies in Phenomenological Philosophy*. The Hague: Martinus Nijhoff.

Shih C., Szybalski A, Fox A., Winograd T., Croné M. (2004). *TeamSpace: a A Simple, Low-Cost and Self-Sufficient Workspace for Small-Group Collaborative Computing*. Poster, CSCW'04, Chicago, USA.

Smith, Consolvo, LaMarca, Hightower, Scott, Sohn, Hughes, Iachello and Abowd, (2005) Social Disclosure of Place : From Location Technology to Communication Practises*, In Proceedings of the 3rd International Conference of Pervasive Computing* (Pervasive 2005), Munich, Germany, LNCS 3468, Springer-Verlag

Starner, T., Kirsch D., and Assefa, S. (1997) The Locust Swarm: An environmentally-powered, networkless location and messaging system. *Proceedings of the First International Symposium on Wearable Computers, ISWC'97*, Boston, USA.

Strang, T., Linhoff-Popien, C., Frank, K. (2003) Applications of a Context Ontology Language. *In Proceedings of International Conference on Software, Telecommunications and Computer Networks* (SoftCom2003), D. Begusic and N. Rozic, Eds., Faculty of Electrical Engineering, Mechanical Engineering and Näaval Architecture, University of Split, Croatia, pp. 14–18.

Strang, T., Linhoff-Popien, C., Frank, K. (2003). CoOL: A Context Ontology Language to enable Contextual Interoperability. *In LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems* (DAIS2003) vol. 2893 of Lecture Notes in Computer Science

Suchman, L. (1987). *Plans and Situated Actions*. Cambridge: Cambridge University Press.

Säljö, R. (2000) *Lärande I Praktiken*. Stockholm, Bokförlaget Prisma. ISBN: 91-518-3728-5

Tao, Rudys, Ladd and Wallach, (2003). Wireless LAN location-sensing for security applications*, Proceedings of the 2nd ACM Workshop on Wireless Security*, San Diego, CA, September 2003

Vander Wal, T. (2005) *Explaining and showing broad and narrow folksonomies :* Personal infocloud. http://www.vanderwal.net/random/entrysel.php?blog=1635 (2005) Accessed February 28, 2007.

Waldo, J. (1999). The Jini Architecture for Network-Centric Computing, *Communications of the ACM*, vol. 42, no. 7, July 1999, p. 76-82.

Wang, X. H., Zhang, D. Q., Gu, T., Pung, H. K. Ontology Based Context Modelling and Reasoning using OWL (2004). *In Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications* (PerCom2004) (Orlando, FL, USA, March 2004), pp. 18–22.

Want, R., Hopper, A., Falcao, V., Gibbons, J., (1992). The active badge location system. *ACM Trans. Inf. Syst.* 10(1) 91-102.

Weiser, M. (1991).The Computer for the 21 st Century. *Scientific America*, 265(3), 94-104.

Werle, P.,Kilander, F.,Jonsson, M.,Lönnqvist, P, Jansson, C. (2001). A Ubiquitous Service Environment with Active Documents for Teamwork Support. *Proceedings of the Ubicomp 2001 Conference*, Atlanta, Georgia, September 2001.

Wei, L. Jonsson, M. (2006) Providing User Self-Contained Location Information Using Mobile Phones. Advances in Pervasive Computing 2006. *Adjunct Proceedings of the 4th International Conference on Pervasive Computing*. Vol. 207, Austrian Computer Society (OCG): Vienna, 2006. 244 pp.

Wyckoff, P. (1998). Tspaces. *IBM Systems J*.37(3). 454-474. Available from World Wide Web: <http://www.almaden.ibm.com /cs/Tspaces>

Yoshimi, B., (2000). On Sensor Frameworks for Pervasive Systems, *Proceedings of the SEWPC Workshop in conjunction with the ICSE2000 conference. Limerick, Ireland.*

DEPARTMENT OF COMPUTER AND SYSTEMS SCIENCES
Stockholm University/KTH
www.dsv.su.se/eng/publikationer/index.html
Ph.D. theses:

No 91-004 **Olsson, Jan**
An Architecture for Diagnostic Reasoning Based on Causal Models
No 93-008 **Orci, Terttu**
Temporal Reasoning and Data Bases
No 93-009 **Eriksson, Lars-Henrik**
Finitary Partial Definitions and General Logic
No 93-010 **Johannesson, Paul**
Schema Integration Schema Translation, and Interoperability in Federated Information Systems
No 93-018 **Wangler, Benkt**
Contributions to Functional Requirements Modelling
No 93-019 **Boman, Magnus**
A Logical Specification for Federated Information Systems
No 93-024 **Rayner, Manny**
Abductive Equivalential Translation and its Application to Natural-Language Database Interfacing
No 93-025 **Idestam-Almquist, Peter**
Generalization of Clauses
No 93-026 **Aronsson, Martin**
GCLA: The Design, Use, and Implementation of a Program Development
No 93-029 **Boström, Henrik**
Explanation-Based Transformation of Logic programs
No 94-001 **Samuelsson, Christer**
Fast Natural Language Parsing Using Explanation-Based Learning
No 94-003 **Ekenberg, Love**
Decision Support in Numerically Imprecise Domains
No 94-004 **Kowalski, Stewart**
IT Insecurity: A Multi-disciplinary Inquiry
No 94-007 **Asker, Lars**
Partial Explanations as a Basis for Learning
No 94-009 **Kjellin, Harald**
A Method for Acquiring and Refining Knowledge in Weak Theory Domains
No 94-011 **Britts, Stefan**
Object Database Design
No 94-014 **Kilander, Fredrik**
Incremental Conceptual Clustering in an On-Line Application
No 95-019 **Song, Wei**
Schema Integration: - Principles, Methods and Applications
No 95-050 **Johansson, Anna-Lena**
Logic Program Synthesis Using Schema Instantiation in an Interactive Environment
No 95-054 **Stensmo, Magnus**
Adaptive Automated Diagnosis
No 96-004 **Wærn, Annika**
Recognising Human Plans: Issues for Plan Recognition in Human - Computer Interaction
No 96-006 **Orsvärn, Klas**
Knowledge Modelling with Libraries of Task Decomposition Methods
No 96-008 **Dalianis, Hercules**
Concise Natural Language Generation from Formal Specifications
No 96-009 **Holm, Peter**
On the Design and Usage of Information Technology and the Structuring of Communication and Work

No 96-018 **Höök, Kristina**
A Glass Box Approach to Adaptive Hypermedia
No 96-021 **Yngström, Louise**
A Systemic-Holistic Approach to Academic Programmes in IT Security
No 97-005 **Wohed, Rolf**
A Language for Enterprise and Information System Modelling
No 97-008 **Gambäck, Björn**
Processing Swedish Sentences: A Unification-Based Grammar and Some Applications
No 97-010 **Kapidzic Cicovic, Nada**
Extended Certificate Management System: Design and Protocols
No 97-011 **Danielson, Mats**
Computational Decision Analysis
No 97-012 **Wijkman, Pierre**
Contributions to Evolutionary Computation
No 97-017 **Zhang, Ying**
Multi-Temporal Database Management with a Visual Query Interface
No 98-001 **Essler, Ulf**
Analyzing Groupware Adoption: A Framework and Three Case Studies in Lotus Notes Deployment
No 98-008 **Koistinen, Jari**
Contributions in Distributed Object Systems Engineering
No 99-009 **Hakkarainen, Sari**
Dynamic Aspects and Semantic Enrichment in Schema Comparison
No 99-015 **Magnusson, Christer**
Hedging Shareholder Value in an IT dependent Business society - the Framework BRITS
No 00-004 **Verhagen, Henricus**
Norm Autonomous Agents
No 00-006 **Wohed, Petia**
Schema Quality, Schema Enrichment, and Reuse in Information Systems Analysis
No 01-001 **Hökenhammar, Peter**
Integrerad Beställningsprocess vid Datasystemutveckling
No 01-008 **von Schéele, Fabian**
Controlling Time and Communication in Service Economy
No 01-015 **Kajko-Mattsson, Mira**
Corrective Maintenance Maturity Model: Problem Management
No 01-019 **Stirna, Janis**
The Influence of Intentional and Situational Factors on Enterprise Modelling Tool Acquisition in Organisations
No 01-020 **Persson, Anne**
Enterprise Modelling in Practice: Situational Factors and their Influence on Adopting a Participative Approach
No 02-003 **Sneiders, Eriks**
Automated Question Answering: Template-Based Approach
No 02-005 **Eineborg, Martin**
Inductive Logic Programming for Part-of-Speech Tagging
No 02-006 **Bider, Ilia**
State-Oriented Business Process Modelling: Principles, Theory and Practice
No 02-007 **Malmberg, Åke**
Notations Supporting Knowledge Acquisition from Multiple Sources
No 02-012 **Männikkö-Barbutiu, Sirkku**
SENIOR CYBORGS- About Appropriation of Personal Computers Among Some Swedish Elderly People
No 02-028 **Brash, Danny**
Reuse in Information Systems Development: A Qualitative Inquiry
No 03-001 **Svensson, Martin**
Designing, Defining and Evaluating Social Navigation

No 03-002 **Espinoza, Fredrik**
Individual Service Provisioning
No 03-004 **Eriksson-Granskog, Agneta**
General Metarules for Interactive Modular Construction of Natural Deduction Proofs
No 03-005 **De Zoysa, T. Nandika Kasun**
A Model of Security Architecture for Multi-Party Transactions
No 03-008 **Tholander, Jakob**
Constructing to Learn, Learning to Construct - Studies on Computational Tools for Learning
No 03-009 **Karlgren, Klas**
Mastering the Use of Gobbledygook - Studies on the Development of Expertise Through Exposure to Experienced Practitioners' Deliberation on Authentic Problems
No 03-014 **Kjellman, Arne**
Constructive Systems Science - The Only Remaining Alternative?
No 03-015 **Rydberg Fåhræus, Eva**
A Triple Helix of Learning Processes - How to cultivate learning, communication and collaboration among distance-education learners
No 03-016 **Zemke, Stefan**
Data Mining for Prediction - Financial Series Case
No 04-002 **Hulth, Anette**
Combining Machine Learning and Natural Language Processing for Automatic Keyword Extraction
No 04-011 **Jayaweera, Prasad M**.
A Unified Framework for e-Commerce Systems Development: *Business Process Patterns Perspective*
No 04-013 **Söderström, Eva**
B2B Standards Implementation: Issues and Solutions
No 04-014 **Backlund, Per**
Development Process Knowledge Transfer through Method Adaptation, Implementation, and Use
No 05-003 **Davies, Guy**
Mapping and Integration of Schema Representations of Component Specifications
No 05-004 **Jansson, Eva**
Working Together when Being Apart – An Analysis of Distributed Collaborative Work through ICT from an Organizational and Psychosocial Perspective
No 05-007 **Cöster, Rickard**
Algorithms and Representations for Personalised Information Access
No 05-009 **Ciobanu Morogan, Matei**
Security System for Ad-hoc Wireless Networks based on Generic Secure Objects
No 05-010 **Björck, Fredrik**
Discovering Information Security Management
No 05-012 **Brouwers, Lisa**
Microsimulation Models for Disaster Policy Making
No 05-014 **Näckros, Kjell**
Visualising Security through Computer Games
Investigating Game-Based Instruction in ICT Security: an Experimental approach
No 05-015 **Bylund, Markus**
A Design Rationale for Pervasive Computing
No 05-016 **Strand, Mattias**
External Data Incorporation into Data Warehouses
No 05-020 **Casmir, Respickius**
A Dynamic and Adaptive Information Security Awareness (DAISA) approach
No 05-021 **Svensson, Harald**
Developing Support for Agile and Plan-Driven Methods
No 05-022 **Rudström, Åsa**
Co-Construction of Hybrid Spaces

No 06-005 **Lindgren, Tony**
Methods of Solving Conflicts among Induced Rules
No 06-009 **Wrigstad, Tobias**
Owner-Based Alias Management
No 06-011 **Skoglund, Mats**
Curbing Dependencies in Software Evolution
No 06-012 **Zdravkovic, Jelena**
Process Integration for the Extended Enterprise
No 06-013 **Olsson Neve, Theresia**
Capturing and Analysing Emotions to Support Organisational Learning:
The Affect Based Learning Matrix
No 06-016 **Chaula, Job Asheri**
A Socio-Technical Analysis of Information Systems Security Assurance
A Case Study for Effective Assurance
No 06-017 **Tarimo, Charles N.**
ICT Security Readiness Checklist for Developing Countries:
A Social-Technical Approac**h**
No 06-020 **Kifle Gelan, Mengistu**
A Theoretical Model for Telemedicine
- Social and Value Outcomes in Sub-Saharan Africa
No 07-001 **Fernaeus, Ylva**
Let's Make a Digital Patchwork
Designing for Children's Creative Play with Programming Materials
No 07-003 **Bakari, Jabiri Kuwe**
A Holistic Approach for Managing ICT Security in Non-Commercial Organisations
A Case Study in a Developing Country
No 07-004 **Sundholm, Hillevi**
Spaces within Spaces: The Construction of a Collaborative Reality
No 07-005 **Hansson, Karin**
A Framework for Evaluation of Flood Management Strategies
No 07-007 **Aidemark, Jan**
Strategic Planning of Knowledge Management Systems
- A Problem Exploration Approach