Sensing and making sense

Designing middleware for context aware computing

Abstract:

Chapter 1: Introduction

A probable development of tomorrow's computer technology is a continuation of the ongoing trends of computing devices becoming increasingly smaller, wireless and embedded into other artefacts. Some of the devices will be carried around and some of them will be stationary. The novel technologies are also increasingly dependent of communication with other computing devices over different kinds of networks. 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.

This increasing inhabitation of technological devices in our everyday environments will not only alter the way we understand computers, but it will also require fundamental changes in the design of the software applications of tomorrow.

One design approach that is gaining ground in the new distributed computing environments can be referred to as service oriented or service centric computing (Papazoglou et. al 2003, Espinoza 2002). The basic assumption for this approach is that computational and hardware functions are 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 more complex computer systems.

New technologies also make it possible to move interaction with computers beyond the desktop and into the larger real world where we live and act. Mobile computing makes it possible to bring the power of computation to new situations and activities and thus in new ways support the tasks at hand. So far the devices of today are however not that well integrated with the physical world, nor do they adapt to situational and environmental changes very well. One solution to this is to try and provide the computer systems with information about the context in which the interaction 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 systems that can dynamically rearrange and recompose themselves with respect to the changing environments. In order to realize open service oriented systems, 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.

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 for example be decoupled from the device that runs the application, or you might 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.

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 sense-making 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 user empowerment. 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:

1.1.1 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 application that can be supported?

1.1.2 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?

1.1.3 Empowering the user in context aware systems

How can you support empowerment 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 some extent affected by a human computer interaction (HCI) perspective, even though the actual contributions relies more on traditional software engineering (SE) 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 Evaluating middleware

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 it hard to evaluate using existing evaluation techniques. This problem has earlier been addressed in a paper 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 number of different workarounds to this problem. For example you have to decide whether to implement rich real world applications or lightweight proof of concept applications 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, makes them 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.

1.2.2 Process

The contributions in this thesis are centred around the design of three software systems. So the research process can to a large extent be described in terms of how these system designs evolved and how they were evaluated. The first system that was designed mainly targeted the research problem of how to achieve context sensitive service discovery. The first step in this process was the identification of the problem, which was performed by starting out with our own experiences in service oriented systems, then through analyzing existing research within the area. The next step was to form a theoretical solution to the problem, including a matching model of a computer system. The model was then implemented as a first prototype. Since the prototype system can be seen as a backend system, aiming to support other system components, it becomes very hard to evaluate the system in isolation. The prototype was therefore incorporated in two larger systems that included user interface components and thereby means to perform user studies. Since the prototype would rather provide new functionalities than increase the performance of the larger systems, user studies was to prefer for evaluation before quantitative computational evaluation methods. The user study performed was qualitative in nature, where the participants were using the system over a longer period of time in a real work situation. Results were mainly gathered from observations of the usage of the system as well as from interviews of the participants.
The next system design is targeting a somewhat broader problem; a generic support for context aware applications. 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. Evaluation has mainly been performed by comparisons with other approaches. The third system started out with 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 system model was created supporting the initial ideas. Based on the model a prototype system was developed. Initially the system was designed with one specific application in mind. This application was then used in a user study where some of the initial hypotheses were evaluated. Later the system implementation was refined and other smaller proof of concept applications was implemented to examine and illuminate other properties of the system.

1.3 Thesis contributions

The contents of this work can be divided into three contributions:

1. An investigation of approaches to enable acquisition and provision of context information to support the creation of context aware applications. This

contribution is based on the design of three middleware system designs: The Context Shadow system, The ACAS architecture, and the Spots system.

2. An investigation of mechanisms to support context dependent service discovery. Using context information middleware to enable some key features in Ubiquitous Service Environments. One such key functionality is e.g. that it allows for software components to query the infrastructure for services that are in some sense relevant to the context of use. This functionality has been explored and evaluated within some example implementations.
3. An investigation of mechanisms for empowering the user in context aware systems. This approach is encompassed by a system called Spots, in which users themselves define and describe places according to their needs, or in accordance with the requirements of some application.

1.4 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 modelling. 2) How to support context aware service composition and 3) How to empower 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 can be empowered in context aware systems.

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/smart 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 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.

2.1.1 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.

2.1.2 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.

2.1.3 Defining 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 seems like a necessity. Following definition is loosely based on the definition of context used by Dey in (Dey 2001), but has been modified to better suit the framework of Ubiquitous Service Environments described above. The definition reads:

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

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

"Representations as well as properties of entities and the current relations between the entities"

Where the entities could be physical entities such as people and places, system components like software services or organizational entities like projects or groups.

2.1.4 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

2.2 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.) has 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. By embracing parts of the context aware computing domain described above one could enable the creation of software services that can act on events in the physical world as well as on events from other parts of the computer 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 contains a number of elements:

- Activity: The task performed by one or several persons that determines the borders of the USE.
- Services: Software entities performing different kinds of 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: Data that is handled 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 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.

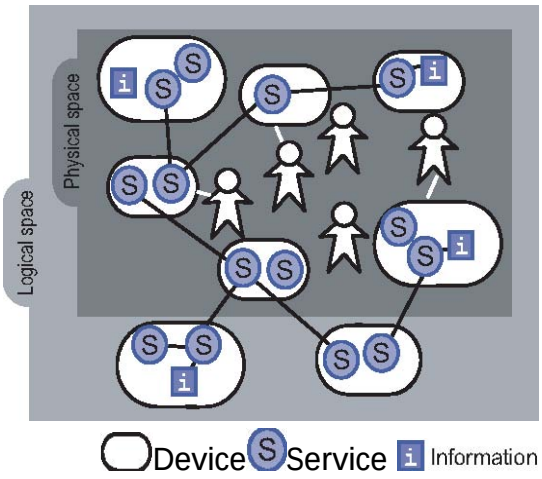


Figure 1. A snapshot of a USE supporting a specific activity

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.

Chapter 3: Investigating the problem space

This chapter will explain why middleware is useful to support the creation of context aware applications. An overview of three different context information middleware designs are introduced, which in different ways supports the creation of context aware applications. Three key problems will also be introduced, including a brief overview of how these problems have been addressed. 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.

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 (Dey …), (Hendricksson…), (Hong). 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 Hiding 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.

3.2 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, Krishnamurthy and Want, 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. (MicaMotes, etc.)

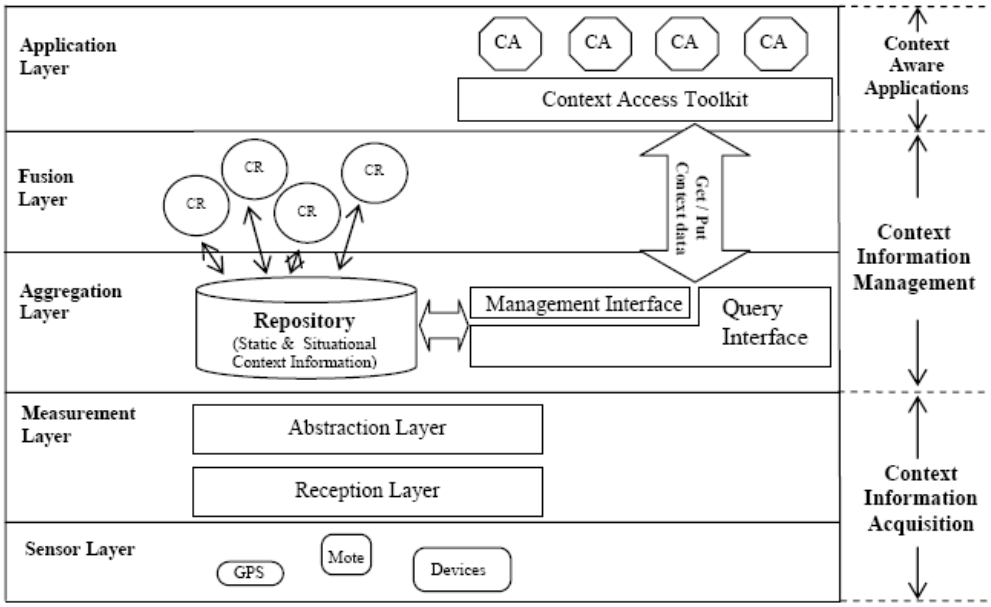
The solutions for the other category of problems, regarding management of context information, often consists of different kinds of software components e.g. in the form of toolkits (Salber et. al 1999) or infrastructures (Hong and Landay 2001). These systems can e.g. abstract or compose sensor data into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

3.3 General schematics of a context aware middleware

[Add some general schematics for context aware middleware, describing layers and functions on a general plane]



[Liwei's schematics, change to something similar, but slightly less detailed]

3.4 Three context information middleware designs

Below follows a brief overview of three middleware system designs that in different ways supports the creation of context aware systems. A more thorough description of each system will be provided in chapter 4.

3.4.1 Context Shadow: Supporting context aware service discovery with interlinked context servers

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 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.

3.4.2 The ACAS architecture: Dynamic linking of service environments

The ACAS 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.

3.4.3 The Spots system: Collaborative location awareness

While the two previous systems rely heavily on that the surrounding environments are enhanced with both sensor and software components, the Spots system is examining a way 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 system 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.

3.5 Identifying some key design problems for context information middleware

Now we have presented three systems that in different ways support the creation of context aware applications. These systems can be used to illuminate some key design issues that has to be dealt with in these kinds of systems, and that have been investigated in further depth. 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 the users can be empowered through the design of context information middleware. A basic functionality of most context aware systems is to take actions autonomously, without explicit user intervention. For systems encompassing such behaviour it is very important to make sure that any actions taken happens in correspondence with the user's wishes. By empowering the user we thus mean that the users should be given increased control over the parts of the system, in several aspects. In the following sections these three questions will be analyzed in further detail.

3.6 Interpretation in context information middleware systems

The basic functionality of a context aware system is that some application is changing its behaviour based on input acquired from some 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 we will discuss later) compared with a more straightforward approach, it is interesting to see where and why this kind of reasoning is needed.

3.6.1 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.

3.6.2 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 motion.
Often you want to draw different kinds of conclusions based on the available sensor data, where the data is 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.

3.6.3 Decision making based on user preferences

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.

3.6.4 How high to aim? Choosing an appropriate level of interpretation

When interpreting data coming from a set of sensors, you can strive for different levels of abstractions as the end result. In general the higher level of abstraction you strive for, the more probable it is that you get an erroneous result.
The reason behind this is that for each inference you make based on your data, you have to use a number of assumptions (or auxiliary hypotheses) regarding different aspects of the world. If you for example want to infer that a meeting is occurring 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. These kinds of high level activity inferences thus require a lot of information to be encoded in the system.
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 degree incorrectness 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 in terms of human activities etc. that the model is aiming at.
Using a complex context model has further drawbacks

- Computational overhead: Inference engines working on complex models will consume considerably more computations than a simple model (referens till context model survey pappret?)
- Maintenance problems: Somebody will have to encode these models and also keep them updated. In a distributed computing scenario, the models might also be scattered over a large number of devices.

In this work we will argue that a lot of useful actions can be taken based on sensor information, without trying to infer too high level activities or situations.

3.6.5 The problem of generality and models

Context models as described above 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. Within the research community of context aware systems, context models are sometimes described in terms of encoded knowledge of the world. This view is however potentially dangerous, since it somehow suggests that there is some kind of essential properties of entities in the world that can be extracted more or less successfully. What one has to bear in mind is however that any description of an entity is only meaningful in relation to a specific purpose.

In this work we try to create general middleware support for context aware applications that to some extent should encompass different functionalities sketched above.

3.7 Aspects of context formalization

There are several possible angels to examine the problem of context formalization. In a recent survey of different context models (Strang, Linhoff-Popien), six aspects were defined as being important for the success of a context model in the ubiquitous computing domain:

1. distributed composition (dc): Due to the distributed nature of ubiquitous computing systems, there should be no central instance being responsible for the creation, deployment and maintenance of context descriptions
2. partial validation (pv): It is highly desirable to be able to partially validate contextual knowledge on structure as well as on instance level against a context model
3. richness and quality of information (qua): A context model appropriate for usage in ubiquitous computing should inherently support quality and richness indication.
4. incompleteness and ambiguity (inc): The model should be able to handle the fact that sensor data is often incomplete and ambiguous, e.g. by interpolation of incomplete data.
5. level of formality (for): It is highly desirable, that each participating party in an ubiquitous computing interaction shares the same interpretation of the data exchanged and the meaning "behind" it (so called shared understanding).
6. applicability to existing environments (app): From the implementation perspective it is important that a context model must be applicable within the existing infrastructure of ubiquitous computing environments, e.g. a service framework such as Web Services.

The aspects listed above are to a large extent concerning issues that has to do with how context information is *encoded*, that is, how you chose to describe the context information that you have decided to be important. When focusing on the generality problem we are not so much focusing on the encoding of the information, but are instead looking at more general issues, such as how to divide the model between the different system layers, how rich the model should be etc.

3.8 Existing approaches for context modeling

Several approaches have been proposed to encode context information to support sharing of information and different kinds of inference tasks.

3.8.1 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 see [39]) 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.

3.8.2 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).

3.8.3 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. (Gu 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.

3.9 Proposed solutions to the interpretation problem

3.9.1 A Simplistic representational model

In this system 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. The only implemented relations were a "At"-relation between persons and places, and a "part of" relation between persons and groups. All sensors and applications using the Context Shadow system will have to conform to this model.
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.

3.9.2 Supporting application specific context models

The ACAS system 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.
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 architecture 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 approach in ACAS can shortly be described as follows: The sensor data you subscribe for from the sensors is encapsulated into small packets called “context elements”, representing the sensor data in a *close-to-sensor* format. Context Refiners modules can then be used that transform or combine the context element into new elements. The idea with the Context Refiners is thus that they are application specific, so that several refiners can be applied to the same data,

3.9.3 Collaborative location modeling

The approach that we wanted 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.

3.10 Using context information middleware to achieve dynamic service discovery

An interesting application domain for context information is to try and use it as a tool 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. 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.

3.10.1 Service environments and service discovery

[What is a service environment?]
[Focus on proximity based sd]

3.10.2 USE implementations [stryk eller skriv om?] Examples of 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 [ref] 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. Other possible technologies that we are currently investigating are web services using Web Services (a specification for using XML in simple message-based exchanges), and the JXTA protocol for peer to peer networking.

3.10.3 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.

3.10.4 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.

3.10.5 Context sensitive service discovery

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. (Currently, Bluetooth's service discovery protocol is being mapped to the Salutation architecture).

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.

3.11 Achieving context dependent service composition in ubiquitous service environments

One of the key issues that the middleware systems in this system targets is the issue of service discovery. A context information infrastructure could dynamically provide information about services that are relevant to the current context. Service discovery is basically the means with which one service can discover other services over a network, and also select the “right” service from a larger set of services.
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 representations of the world, 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.

3.11.1 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.

3.11.2 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.

3.11.3 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.
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.

3.12 Empowering the user in context information middleware

One common assumption with respect to the vision of ubiquitous and pervasive computing is that the goal is to create one widespread system, entirely interconnected and constantly evolving to meet new future needs and technical achievements. Typical scenarios concern situations where users encounter new environments containing different kinds of services that could potentially be offered to the user, where the goal is to create systems that can cope with this kind of situations. Providing context information about the physical world has been seen as a silver bullet in order to enable systems with this kind of functionality. The focus has so far been to create systems that operate on the users behalf with little requirements on active participation from the user. One problem with this approach is that it results in systems that must have the ability to handle and reason with any kind of information describing the world. An alternative approach is to make the user a more active participant in the systems dealing with context information. Instead of being only a user of the system the "user" instead becomes a participant in the system development process. One can imagine several ways in which the user could participate in order to make these evolving, pervasive and semiautomatic computer systems more realistically realizable.
By empowering the user we mean that the users should be given increased control over the system, in two major aspects:

1. Adaptability: The user should be able modify the behaviour of the system, both in order to make sure that the system actually behaves in accordance with the user's preferences, but the systems should also be easily extendable so that new functionalities could be added in an easy way.
2. Information ownership: By designing context information systems in a user centric way, users can be given increased control over how context information concerning the user is interpreted, disseminated and formatted.

3.12.1 User centric 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.

3.12.1.1 The notion of a personal context information system

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.

3.12.2 From system users to active participants

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.

3.12.2.1 Seamfulness in context information middleware

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

3.12.2.2 Community based instrumentation of shared environments

In the Spots system the users define places where they are needed, and may add service information related to that place. 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 about the tagged places can then be shared through communities.

3.12.3 Users as interpreters of meaning

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

3.12.3.1 Community defined location semantics

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.

3.13 Summary

In this chapter an introduction to context information middleware has been presented. A brief introduction was given of three different context information middleware systems. Finally, three specific design issues with respect to context information middleware were presented, together with the outlines of how these problems have been addressed.

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 middleware designs will be presented in further detail, together with some example applications.

4.1 The Context Shadow system: Supporting context dependent service discovery with interlinked context servers

In the Context Shadow 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.

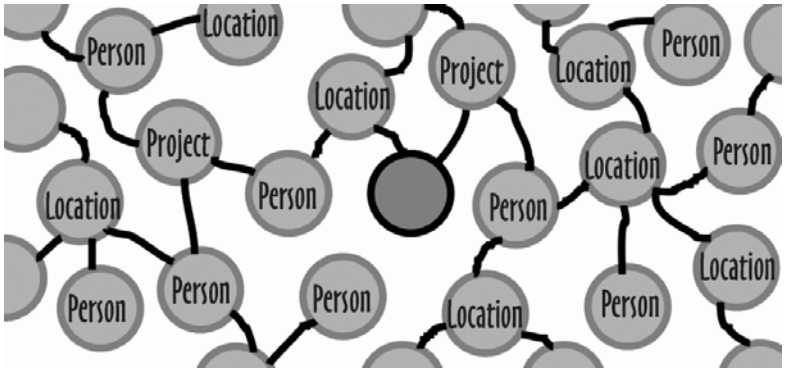


Figure 2: The linked context servers create a searchable space, where the topology of the space is part of the context information.

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.

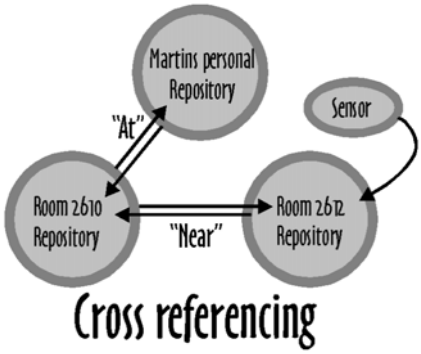


Figure 3. The context servers are linked with references. By following the links it is possible to acquire context information from other context servers than your starting point.

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.

4.1.2.1 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.

4.1.2.2 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.

4.1.2.3 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 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.

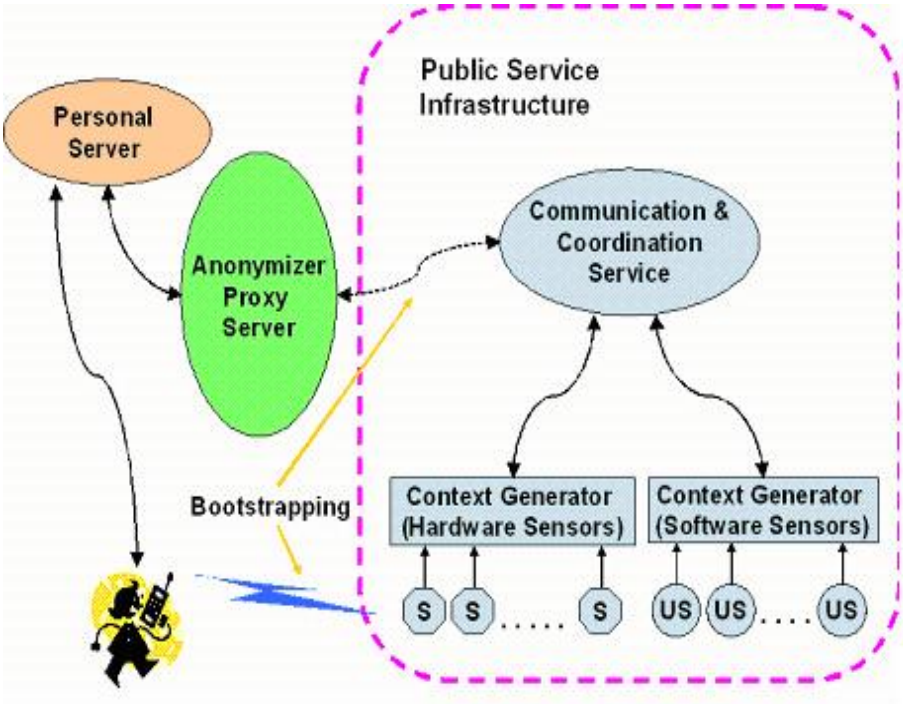
The hardware Sensor Context Generator consists of a set of services; each presents certain context information about the PSI (or a user inside the PSI). Such context information is derived (sensed or inferred) from various physical sensor input, e.g., a Personal Location Service could detect user's presence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

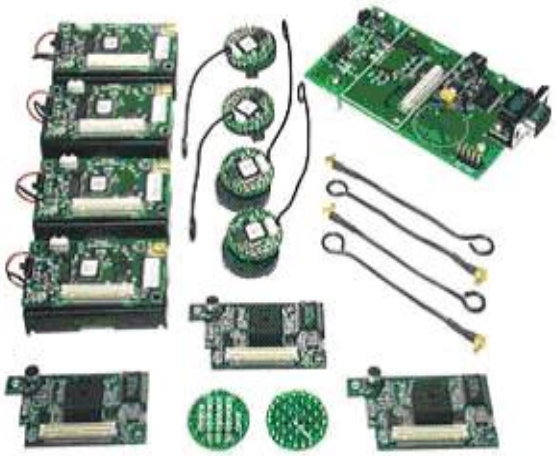
1. The user presents herself to the PSI through some means of direct or indirect action: a login interface, a broadcast identification or others. The PSI then contacts the PCS.
2. The user detects the PSI and sends a location code or broadcast identification to the PS. The PS then contacts the PSI.



Location information has been exploited as a very useful piece of context information as there are many location based systems [26] 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.

4.2.2.1 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.



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 (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 [Mica Mote positioning ref], but these are specialized solution that will not allow for simultaneous collection of sensor data.

4.2.2.2 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.

Please note, here we use low-level Bluetooth device address only for bootstrapping the association of local infrastructure with user’s home PCS systems, not for identifying users. The user and her PCS are addressed by the temporary token set by the user (or by autonomous applications) which is intentionally changeable according to user’s setup (such as once an hour). However, we are aware of the possibility of identifying user via the Bluetooth device address. The reason why we select Bluetooth for our prototype at this moment is due to many advantages of Bluetooth such as the wide support by many commodity devices such as mobile phones, and its low power consumption. And we also believe that fixed infrastructures will have more network accessibilities to connect to user’s home PS than mobile user devices.

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)
[7] and ”A Model for Presence and Instant Messaging” (RFC 2778) [8], as well as another work in progress draft ”Common Presence and Instant Messaging” (CPIM) [43], developed within the IMPP working group.
In addition, SIMPLE is working on some draft specifications e.g., Presence Event Package and Watcher Information Event Package. Some of these work in progress drafts have significant influence on our context information delivery infrastructure, such as the new Publish mechanism for presence [4], Event Notification Extension for Collections [32], and we have decided to support them in our context infrastructure design. The key concepts in the proposed SIP presence framework can be elicited below:

- 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 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 The Spots system

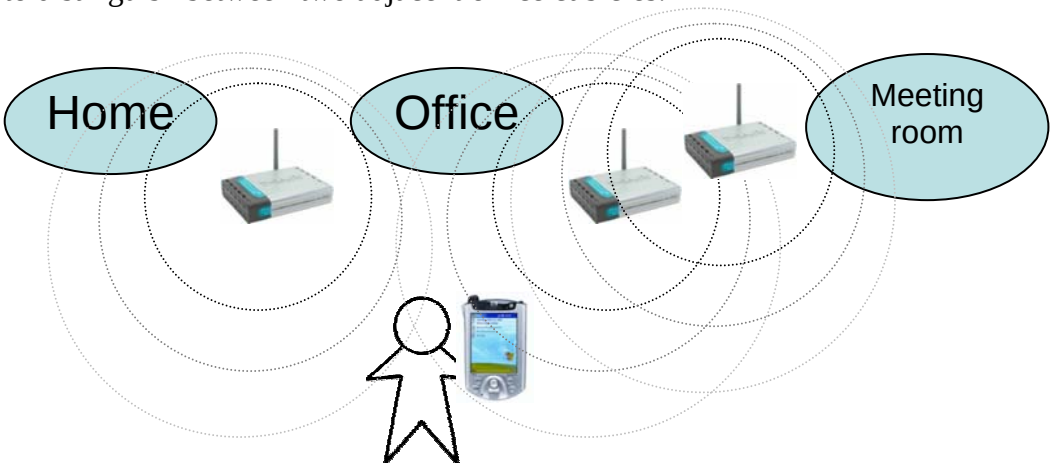
An implementation of a system named “Spots” that follows the design approaches sketched above is presented in this chapter. The basic functionality of the Spots system is that it makes it possible to detect when a person is present at some predefined place, where users themselves define the places they 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.



4.3.1.1 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 Spot 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 Spot 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 Spot 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.

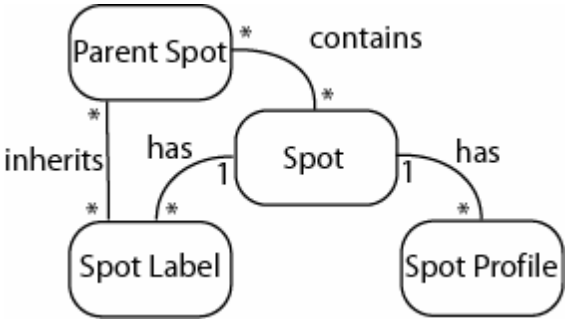


Figure x: Conceptual location model design

4.3.1.2 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 Spot 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.

4.3.1.3 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 Spot system implementation

This implementation of the Spots system was compiled to test the Spot 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.

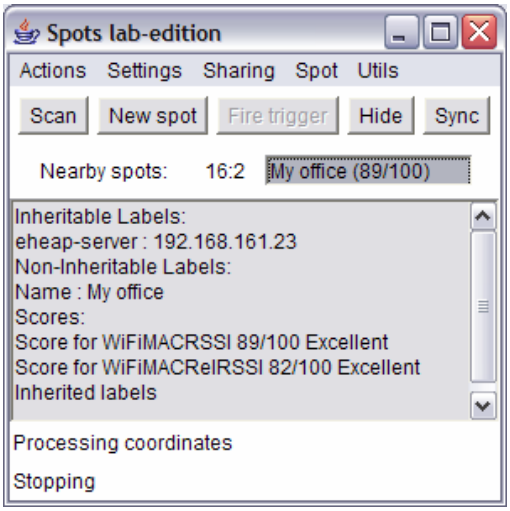


Figure 3: A snapshot from the Spot sensor and designer client

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.

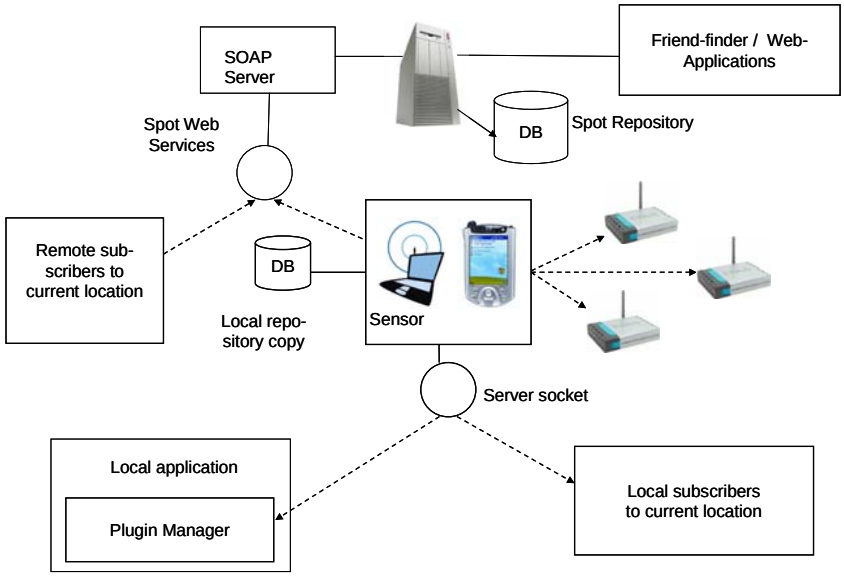


Figure 1: Spot system's architecture

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 Spot 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 Spot 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 Spot 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 Spot 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.

4.3.4.1 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	

4.3.4.2 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.

4.3.4.3 Location aware TeamSpace

This application was created by location awareness to an existing application. The TeamSpace (teamspace ref) 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

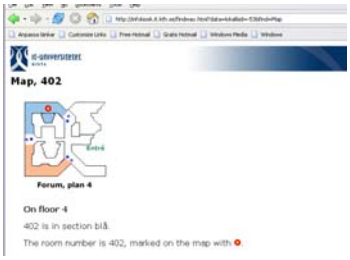
system the application would automatically connect to recourses in the local environment when the user entered a new Spot.

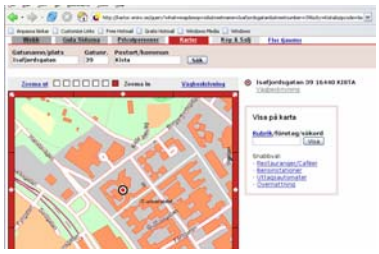
4.3.4.4 Interactive room configuration with a physical cursor

In this application the Spot system was used to identify the different devices present in an interactive room designed to support local collaboration [iLounge ref]. 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.

4.3.4.5 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 criterias 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





4.4 Summary

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. Among the similarities are for example the idea 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 to empower the user

Chapter 5: Designing context interpretation processes to maximize generality

[A recap of the problems defined in 3]

5.1.1 Evaluation criteria

Managing heterogeneity
Expressiveness
Computation overhead

5.1.2 Design parameters

Richness: 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. 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.
Layer division: It is not obvious how large part of the context model that should reside in the middleware layer, and how much that should reside in the application layer. A larger model in the middleware layer makes inferencing possible in the middleware, but might have negative effects on the generality.
Representations: The generality aspect might also be affected by whether you choose a situational or representational encoding of the context model? Whereas a formalization that contains representations of real world entities might become more coherent, a situational or event based encoding makes it possible to get results with inconsistent and incomplete models.
Rigidness: If it is hard to find a general context model that works for a broad range of applications, an alternative approach is to make the models flexible or extendable in different ways.

5.2 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.

5.2.1 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.

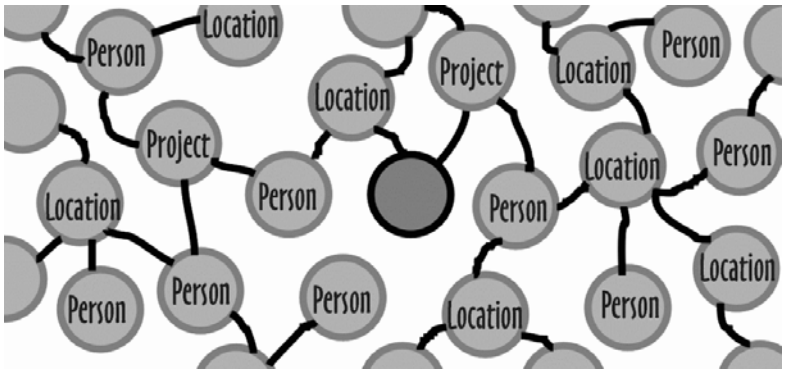


Figure 4: The linked context servers create a searchable space, where the topology of the space is part of the context information.

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.

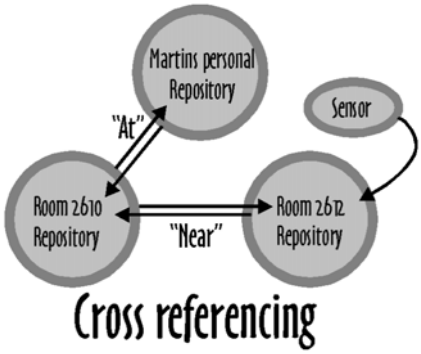


Figure 5. The context servers are linked with references. By following the links it is possible to acquire context information from other context servers than your starting point.

5.2.2 Context model: Versatility through simplicity

In this system 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. The only implemented relations were a "At"-relation between persons and places, and a "part of" relation between persons and groups. All sensors and applications using the Context Shadow system will have to conform to this model. 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.

5.3 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 such a way. Different components have however been tested separately in order to prove that the architecture is viable.

5.3.1 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.

5.3.2 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. A good id would probably include 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.

5.3.3 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.

5.3.3.1 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.

5.3.3.2 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.

5.3.3.3 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.

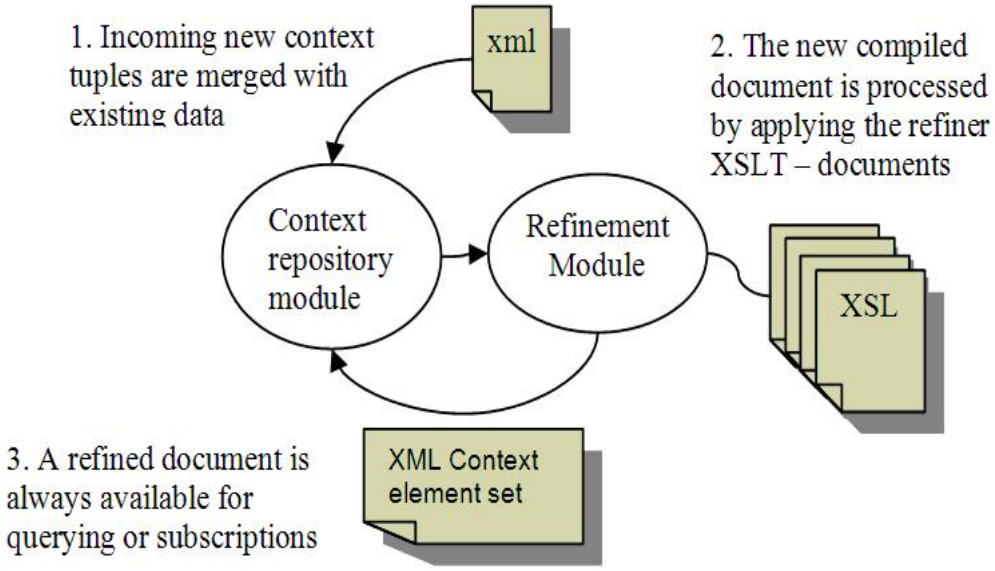


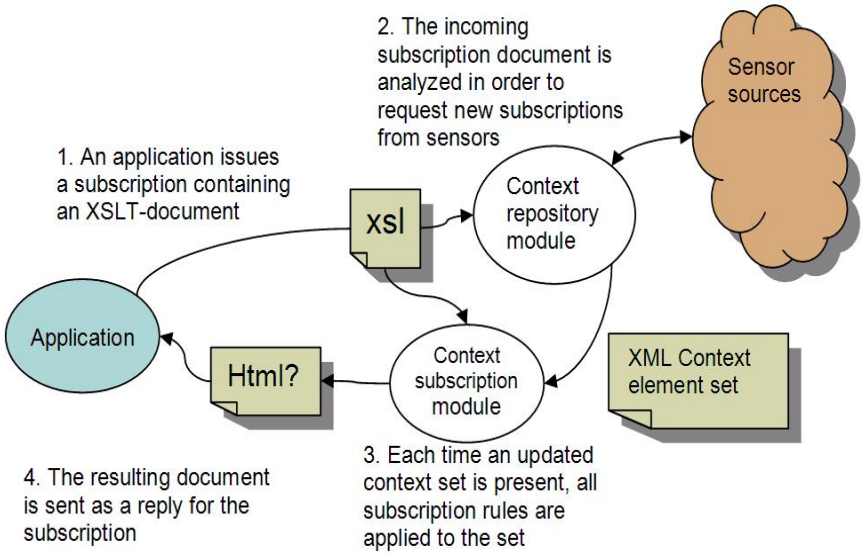
Figure 6. The context refiners transform the available context elements and posts the results back into the repository.

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 3.1 describes the procedure of how the XSLT rules are applied to the context information.

5.3.3.4 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.



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.

5.4 Design approach 3: Collaborative location modeling

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

5.4.1 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).

5.4.2 Empowering users in the location models

In a typical context aware system in a pervasive computing setting, information is collected from some sensors, interpreted to various extents, and the result is then fed into an application which can act on the information in different ways. The interpretation part of this process may be as simple as transforming a thermometer reading from Fahrenheit to Celsius, or it could be a complex reasoning task such as deducing that a user is in a meeting based on a compilation of information from several sensors. Most context aware systems aim for being able to perform different kinds of interpretations on the set of available sensor data, such as being able to infer both that a person is in a certain location, but also that he is close to a printer. These kinds of generalized interpretations require that there is some kind of underlying model that the sensor data is matched against and that makes the data meaningful. This model is then used to create rules or transformations in order to enable the kind of interpretations described above.

In well defined and limited domains the modelling and decision process is if not trivial, at least possible. The problem is that most domains concerning mobile or pervasive technology, is neither well defined nor limited. Mobile devices are often used both in private and work settings, and pervasive applications tend to be open and general in nature. For complex or dynamic (real life) settings, the modelling and reasoning becomes very complex if not impossible, which has been proven both by the lack of success stories for such applications, as well as by critique from a theoretical perspective by e.g. Lucy Suchman ( ) and Paul Dourish ( ).

Given this line of reasoning one can either chose to design systems for well defined domains, where the context model is simple and the setting is fixed. This would probably generate working applications, which might however not be so interesting or very flexible. The more challenging option is to try to tackle the problems of a dynamically changing domain.

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 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 Spot system.

5.4.3 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.

5.4.4 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 Spot 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 Spot 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 Spot 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.

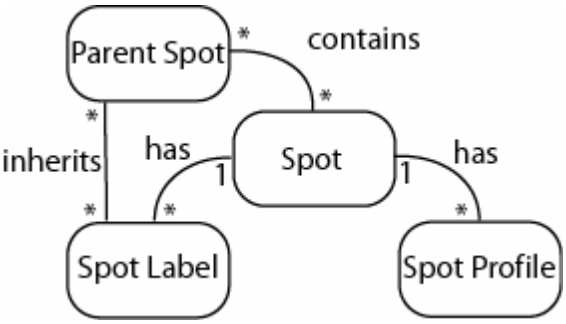


Figure 2: Conceptual location model design

5.4.5 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 Spot 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.

5.4.6 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.

5.4.7 Sharing of context 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 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.

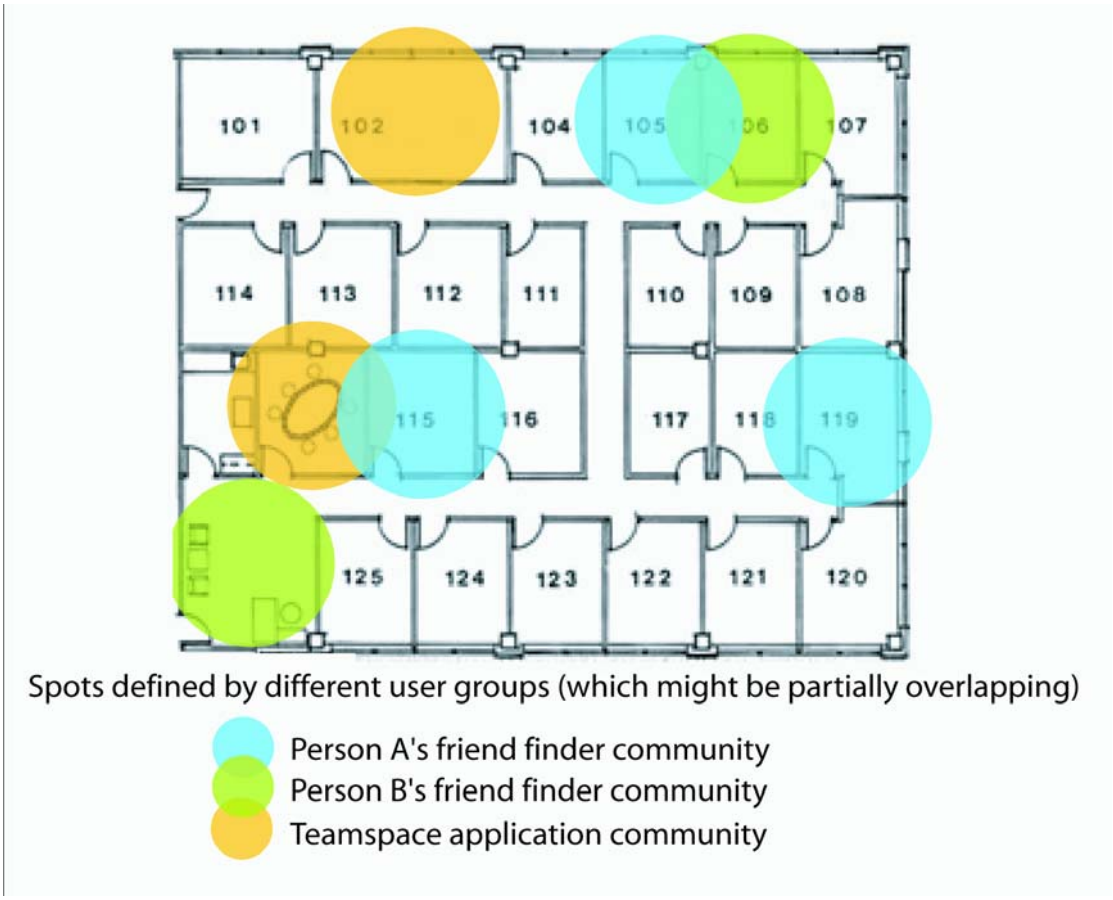


Figure 7: The defined spots might be overlapping, which is not a problem when used by different user or application communities.

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.

5.5 Evaluation

5.5.1 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?

5.5.2 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.

5.5.3 Representations

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 (CONTEXT FABRIC REFERNCE) , where different real world entities were represented by URLs pointing to xml pages describing the entity. In this chapter 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 PostIt (?) (REF, kolla upp vad det heter) system, in which situations are defined in terms of specific conditions on the available information regarding time, coordinates and orientation. In this chapter the ACAS architecture implements the situational approach.

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, you will only acquire a flow of labels describing the current location, a behaviour that very much resembles that of the situational approach.

5.6 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 attacked 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 achieve dynamic service discovery

An interesting application domain for context information is to try and use it as a tool 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 Delimiting the problem

Since the notion of service environments is a fairly broad concept and is still rather new and immature, it is necessary to specify exactly what service environments we are talking about and what kind of behaviours we are looking for.

6.1.1 How should the notion of service composition be understood?

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. In chapter two, the notion of ubiquitous service environments or USEs was introduced, as being a specific assembly of computational services that are used by a user or a set of users to support a specific task.
A key issue to make these USEs a reality is the means to assemble the set 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.

6.1.2 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.

6.1.3 Service discovery

[Introduce service discovery]

6.1.4 Context dependent service discovery

[Introduce how context information can be used to achieve service discovery]
[Explain that it is mainly location that has been adressed]

6.2 Context dependent service discovery

6.2.1 Sub problems

Associating services with descriptive context information
Discovery versus selection
Granularity
Static vs. dynamic (?)

6.2.2 Other approaches to context aware service discovery

MSDA-middleware (Raverdy et al.), Strong focus on bridging heterogeneous networks and heterogeneous protocols. Minimizing network traffic etc.

Context Attributes: An Approach to Enable Context-awareness for Service Discovery.. - Lee, Helal – 2003 Extending Jini

Context-Aware Service Composition in Pervasive Computing Environments
Sonia Ben Mokhtar, Damien Fournier, Nikolaos Georgantas, Val´erie Issarny

6.2.3 Evaluating three design approachs

Context awareness and ubiquitous service environments

Why use context information in USEs? Context awareness can be used to enable support some of the key functionalities in USEs, such as dynamic service composition, service discovery, service bootstrapping and personalization.

6.2.4 USE implementations

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 host names and port numbers.
The first implemented USE, named fuseONE, described in detail in the last paper included in this thesis, 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. Other possible technologies that we are currently investigating are web services using Web Services (a specification for using XML in simple message-based exchanges), and the JXTA protocol for peer to peer networking.

6.3 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. These different service discovery techniques will be presented briefly and discussed with respect to their suitability for usage in Ubiquitous Service Environments.

6.3.1 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.

6.3.2 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.

6.3.3 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.

6.3.4 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.

6.3.5 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.

6.3.6 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.

6.3.7 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.

6.3.8 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 organisational 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.

6.3.9 Context sensitive service discovery

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 organizational 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 organizational 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. (Currently, Bluetooth's service discovery protocol is being mapped to the Salutation architecture).
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.

6.4 Achieving context dependent service discovery in USEs

One of the key issues that the middleware systems in this system targets is the issue of service discovery. A context information infrastructure could dynamically provide information about services that are relevant to the current context. Service discovery is basically the means with which one service can discover other services over a network, and also select the “right” service from a larger set of services.
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 representations of the world, 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 application

6.5 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.

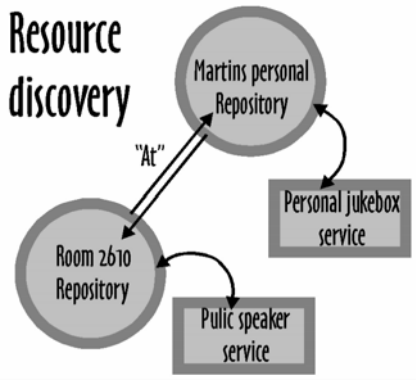


Figure 8. Using Context Shadow, a jukebox service finds a speaker service at the user's current location.

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.

6.5.1 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.

6.6 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

6.6.1 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>
```

Etc...

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 system, 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.

6.6.2 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.

6.6.3 A Discovery Service implementation using Rain

Rain [Rain 2004] 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.

6.7 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 pointer to services at the places where they reside.

Services in the Spots system are described using labels containing simple sets of key-value pairs. Service composition in this case typically means that one service subscribes for information about some specific type of services at the users current location.

6.7.1 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. [referens till web-service baserat pervasive system]

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 (predesigned, preconfigured) 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.

6.7.2 Example

The location sensitive TeamSpace application was created by adding location awareness to an existing application. The TeamSpace [19] 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. 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.

6.7.3 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 letting users form *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.

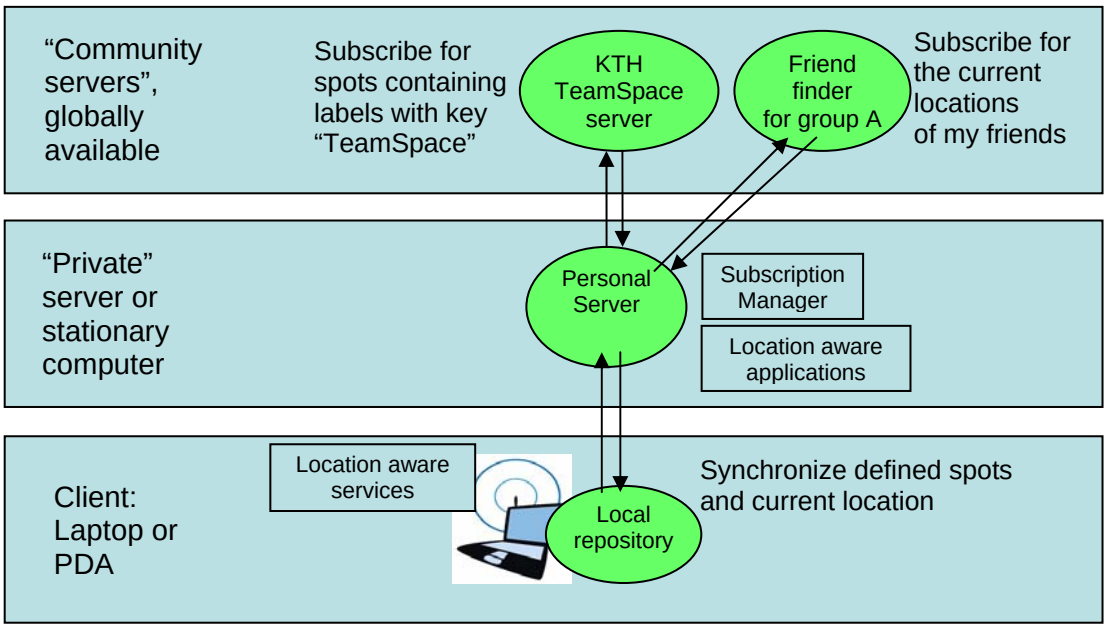


Figure 9. The three tire repository architecture of the Spots sharing mechanism

6.8 Evaluation

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

6.8.1 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.

6.8.2 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.

6.8.3 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.

6.8.4 Conclusions

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 a personal application tries to connect to resources in the user’s vicinity.

Chapter 7: Empowering the user in context information middleware systems

7.1 Delimiting the problem

Since user empowerment is a fairly broad concept it is necessary to specify more precisely how this concept should be understood. Some aspects related to user empowerment that has been addressed before within the context awareness research community [hendricksen ref]: the notion of *privacy*, concerning how to make sure that the context information is not shared with others in a way that is not ok with the user, and aspects related to *autonomy* and how to make sure that the system actions is in accordance with user preferences. There have been several approaches to deal with privacy issues through system design. Examples of design issues that have been considered include deciding *what* information should be shared, *how* to display it and to decide with *whom* to share the information.
Autonomy related issues concern e.g. how the degree of autonomy affects the feeling of control from the user perspective, [Heijden] [Dey] and also issues of accountability concerning how the system displays the causes for its actions [bellotti].

7.1.1 How can users be empowered through aspects of middleware design?

In what ways can you affect the user control by the design of a context information middleware?
[Tydliggör vad som gäller applikationsnivå och vad som gäller middleware]

In this work we focus on three problem areas that affect user empowerment in different ways

7.1.2 Let the user be the owner of the 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 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, and in what format.

7.1.3 Let the user assign meaning to sensor 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

7.1.4 Choosing level of visibility of the context information middleware

In many existing context aware systems the mechanisms that take care of the interpretation of the context information is more or less hidden. There is thus a predefined mapping between the sensed context information and the resulting output.
Design for seamfulness to enable appropriation

7.1.5 [fel ställe?] Supporting flexibility and adaptability for application designers

7.1.6 [Stryk?]Support end user programming

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.

7.2 Existing approaches to achieve user empowerment in context aware systems

One example of a context aware system that uses end-user programming to empower the users is the aCAPpella [] system, 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.

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.

In a typical context aware system in a pervasive computing setting like [], information is collected from some sensors, interpreted to various extents, and the result is then fed into an application which can act on the information in different ways.
The interpretation part of this process may be as simple as transforming a thermometer reading from Fahrenheit to Celsius, or it could be a complex reasoning task such as deducing that a user is in a meeting based on a compilation of information from several sensors. Most context aware systems aim for being able to perform different kinds of interpretations on the set of available sensor data, such as being able to infer both that a person is in a certain location, but also that he is close to a printer. These kinds of generalized interpretations require that there is some kind of underlying model that the sensor data is matched against and that makes the data meaningful. This model is then used to create rules or transformations in order to enable the kind of interpretations described above.
In well defined and limited domains the modelling and decision process is if not trivial, at least possible. The problem is that most domains concerning mobile or pervasive technology, is neither well defined nor limited. Mobile devices are often

used both in private and work settings, and pervasive applications tend to be open and general in nature.
For complex or dynamic (real life) settings, the modelling and reasoning becomes very complex if not impossible, which has been proven both by the lack of success stories for such applications, as well as by critique from a theoretical perspective by e.g. Suchman, Dourish, etc… [ref]
Given this line of reasoning one can on one hand chose to design systems for well defined domains, where the context model is simple and the setting is fixed. This would probably generate working applications, which might however not be so interesting. On the other hand, one could try to tackle the problems of a dynamically changing domain.
One approach that has been explored by some researchers is to have a simple and static base-model, with basic elements that is viable for every possible application scenario, and then domain or application specific models used by different applications. The drawbacks with this approach are firstly that it creates applications that are partially incompatible with other applications in the same infrastructure; secondly it is not obvious that it is possible to find an underlying shared model that would work for every scenario.
Another serious problem shared by all approaches described so far is that the design of the underlying model and the mechanisms controlling the interpretations and decision making 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 restrict the ways in which the information can be interpreted. The designer also in most cases selects which of the
The approach taken in this project is to move some of the control of interpretation of context information to the users of the system, allowing them to continuously adapt the system to their current needs.

from Paul Dourish:
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.

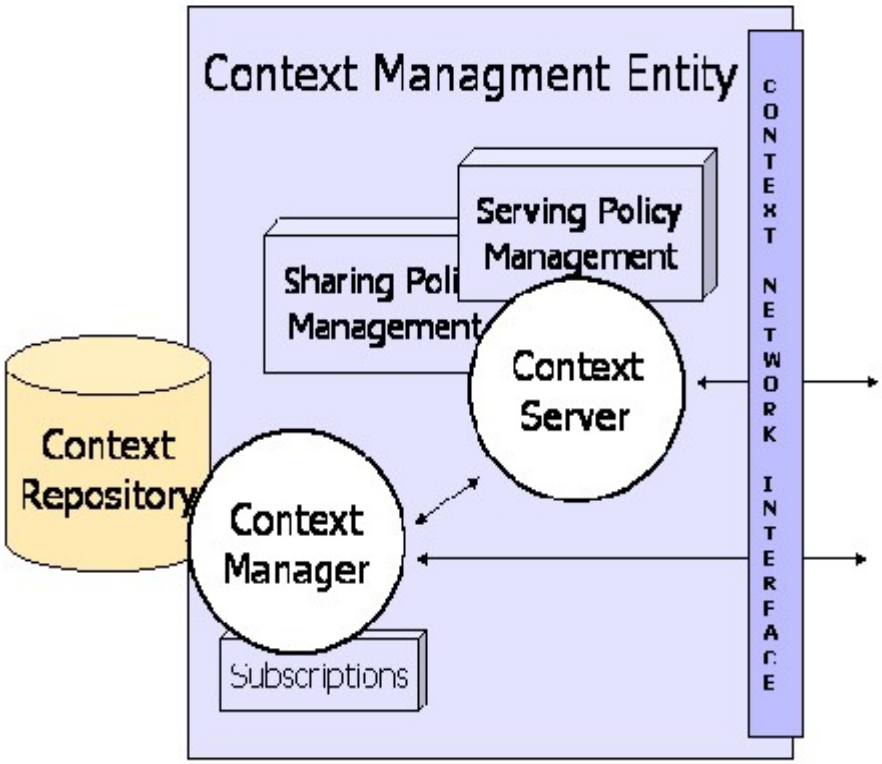
7.3 Information ownership

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. 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.

7.3.1 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.



7.3.2 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.

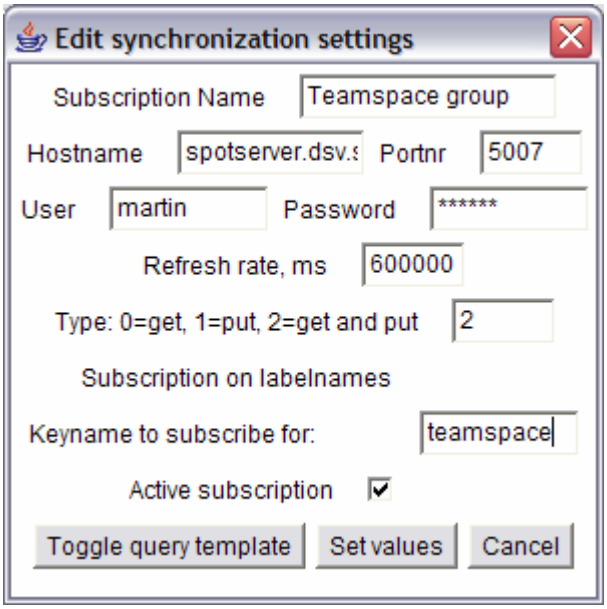


Figure 10. Setting up a subscription in the Spots client interface

7.4 Users as interpreters of meaning

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 contextual information.

7.4.1 User controlled management of context information

One common assumption with respect to the vision of ubiquitous and pervasive computing is that the goal is to create one widespread system, entirely interconnected and constantly evolving to meet new future needs and technical achievements. Typical scenarios concern situations where users encounter new environments containing different kinds of services that could potentially be offered to the user, where the goal is to create systems that can cope with this kind of situations. Providing context information about the physical world has been seen as a silver bullet in order to enable systems with this kind of functionality. The focus has so far been to create systems that operate on the users behalf with little requirements on active participation from the user. The problem with this approach is that it results in systems that must have the ability to handle and reason with any kind of information describing the world. An alternative approach is to make the user a more active participant in the systems dealing with context information.

7.4.2 An experiment on user controlled location naming

In a lab assignment some groups of computer science university students were presented to the Spots system. As one part of the assignment, the students were supposed to design a simple context aware application based on the Spots system, by adapting an existing application, the TeamSpace [18] software, which contain

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 define one or more Spots at some locations, add some application targeted information to that Spot and use it to adapt the TeamSpace software. The Spots system, when introduced to the students, worked only on laptops and contained only the FriendFinder application. The notion of parent Spots and inheritance was not part of the system at that time. The lab assignment spanned over approximately two weeks of time.

Being a very controlled use case occurring over a fairly short period of time, we will not try to draw any general conclusions from the usage of the system. One aspect which however is interesting to examine closer is how the students worked with the labeling of the Spots. To note when looking through usage logs from the students is that most student groups tended to use the Spot 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 Spots, some distinct categories could be discerned:

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" "Stockholm" "BlueSofa2"
Application targeted naming:	"85.224.129.193"|"TeamSpaceAddress“
Unknown meaning:	"wagon meeting"

Table 1. Categorization of spot names

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. 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 [10]. 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.

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.

7.5 Supporting system adaptation

For a context information middleware to be successful, a crucial feature is how easy it is to adapt the system to new applications or usage patterns. There are several examples of existing information technology that to various extents can be adapted to fit several usage areas:

- **email:** simple system used for note-taking, to-do’s, scheduling...
- **http:** simple connection protocol
- **SMS:** used for everything from advertising to dating to collective political action
- **Wikis:** open web communities
- **RSS:** consistent data behind the presentation on websites
- **Blogs:** simple content management changes structure of the web
- **Spreadsheets:** simple tables adapted to many uses.

The adaptability of the middleware can be supported in different ways. In chapter five there was an examination of how the design of the context model will influence the generality of the middleware. Other approaches that could potentially support adaptation include the use of open APIs and protocols, sharing code through open source etc.

7.5.1 A study of Spots system adaptability

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. After the groups presented their system designs they were presented with a questionnaire asking questions regarding their experiences with the Spots system.
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.
The results from the questionnaire showed that most of the groups could adapt the Spots system to realize their initial application ideas.
…etc…

7.6

7.7 Conclusions

Chapter 8: Discussion and concluding remarks

The future of context aware computing

Three trails:
1) Highly specialized domains (recognizing specific actions) and 2) Simple widespread applications mainly based on mobile phones and user participation (geocoding, etc)
3) Web 2.0, user participation, communities
Geocoding

The role of middleware

Bibliography

Edwards, K. W. Bellotti, V., Dey, A. Stuck in the middle: Bridging the gap between design evaluation and middleware.

Abowd, Gregory D., Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper and Mike Pinkerton (1997). Cyberguide: A mobile context-aware tour guide. ACM Wireless Networks 3(5): pp. 421-433. October 1997
Beigl, M. 2000. MemoClip: A location-based remembrance appliance. Personal Technologies, 4(4):230-233, September 2000.
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.
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.
Conner, S., Krishnamurthy, L. and Want, R. 2001. Making Everyday Life Easier Using Dense Sensor Networks. In Proceedings of ACM Ubicomp, Atlanta Georgia.
Dallas Semiconductor. 2003. iButton home page [On-line]. Available at :http://www.ibutton.com/
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).
Dourish, Paul. 2001. Where the Action Is, The Foundations of Embodied Interaction. Cambridge, Massachusetts, MIT Press.

Duranti, A., and Goodwin, C., eds. 1992. Rethinking context. Language as an interactive phenomenon. Cambridge University Press.
Espinoza, Fredrik. 2002. Individual Service Provisioning. Stockholm: Akademitryck AB, Doctoral Thesis, Stockholm University, Dept. of Computer and Systems Sciences.
Fox, A., Johanson, B., Hanrahan, P., Winograd, T. 2000. Integrating Information Appliances into an Interactive Workspace. IEEE Computer Graphics & Applications, 20(3)
Guttman, E. 1999. Service Location Protocol: Automatic Discovery of IP Network Services. IEEE Internet Computing, 1999. 4(4): p. 71-80.
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. and 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.
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.
Hong, J., I., and Landay J., A. 2001. An Infrastructure Approach to Context-Aware Computing, Human Computer Interaction, 2001, Volume 16, pp287-303.
Microsoft Corporation, "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.
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.
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.

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. Context and Consciousness: Activity Theory and Human-Computer Interaction. 1996, MIT Press: Cambridge, Mass.
Norman, D. 1999. The invisible computer. Cambridge University Press)
Pascoe, B. 1999. Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun. Salutation Consortium, White Paper.
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.
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.
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.
Schilit, B., N. Adams, N., and Want, R. 1994. Context-aware computing applications. First International Workshop on Mobile Computing Systems and Applications, 85-90.
Schmidt, A., Takaluoma, A. and Mntyjrvi, J.. 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.
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.
Suchman, L. 1987. Plans and Situated Actions. Cambridge: Cambridge University Press.
Waldo, J. 1999. "The Jini Architecture for Network-Centric Computing," Communications of the ACM, vol. 42, no. 7, July 1999, p. 76-82.
Weiser, M. 1991.The Computer for the 21 st Century. Scientific America, 265(3), 94-104.
Werle, P.,Kilander, F.,Jonsson, M.,Lönnqvist, P. and Jansson, C. 2001 A Ubiquitous Service Environment with Active Documents for Teamwork Support. Proceedings of the Ubicomp 2001 Conference, Atlanta, Georgia, September 2001.
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.

Referenser som behöver hyfsas till

Bob Hardian, Jadwiga Indulska, and Karen Henricksen. Balancing Autonomy and User Control in Context-Aware Systems - a Survey. In 3rd International Workshop on Context Modelling and Reasoning (CoMoRea) (to appear). IEEE Computer Society, March 2006.

DE BRUIJN, J. 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.

WANG, X. H., ZHANG, D. Q., GU, T., AND PUNG, H. K. Ontology Based Context Modelling and Reasoning using OWL. In Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004) (Orlando, FL, USA, March 2004), pp. 18–22.

GU, T., WANG, X. H., PUNG, H. K., AND ZHANG, D. Q. 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).

STRANG, T., LINNHOFF-POPIEN, C., AND FRANK, K. Applications of a Context Ontology Language. In Proceedings of International Conference on Software, Telecommunications and Computer Networks (SoftCom2003) (Split/Croatia, Venice/Italy, Ancona/Italy, Dubrovnik/Croatia, October 2003), D. Begusic and N. Rozic, Eds., Faculty of Electrical Engineering, Mechanical Engineering and Näaval Architecture, University of Split, Croatia, pp. 14–18.

STRANG, T., LINNHOFF-POPIEN, C., AND FRANK,K. 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) (Paris/France, November 2003), J.-B. Stefani, I. Dameure, and D. Hagimont, Eds., vol. 2893 of Lecture Notes in Computer Science

Marmasse and Schmandt, *Location aware information delivery with commotion* , MIT Media Laboratory, HUC 2000 Proceedings, pp 157-171, Springer-Verlag.
Smith, Consolvo, LaMarca, Hightower, Scott, Sohn, Hughes, Iachello and Abowd, *Social Disclosure of Place : From Location Technology to Communication Practises,* Intel Research, To appear in Proceedings of the 3rd International Conference of Pervasive Computing (Pervasive 2005), Munich, Germany, LNCS 3468, Springer-Verlag
Espinoza, Persson, Sandin, Nyström, Cacciatore and Bylund, *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
Persson and Fagerberg, *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
Dimitre Novatchev. Functional programming in xslt using the fxsl library. Presented at Extreme Markup Languages 2003.

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

Krumm and Hinckley (2004), *The NearMe Wireless Proximity Server,* Microsoft Research, presented in UbiComp 2004. The 6th International Conference of Ubiquitous Computing, September 7-10, 2004, Nottingham, England

17. 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.

Bahl and Padmanabhan, *RADAR: An In-Building RF-Based User Location and Tracking System*, Proceedings of IEEE Infocom 00, April 2000.
Haeberlen, Rudys, Flannery, Wallach, Ladd and Kavraki (2004), *Practical Robust Localization over Large-Scale 802.11 Wireless Networks*, Mobicom 2004
Tao, Rudys, Ladd and Wallach, *Wireless LAN location-sensing for security applications*, Proceedings of the 2nd ACM Workshop on Wireless Security, San Diego, CA, September 2003

Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian E. Smith, James Scott, Timothy Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, Bill N. Schilit (2005), *Place Lab: Device Positioning Using Radio Beacons in the Wild*, Proceedings of the Third International Conference, PERVASIVE 2005, Munich, Germany, May 8-13, 2005

A. Roach. RFC 3265: Session initiation protocol (sip)specific event notification, June 2002

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

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

B. Campbell and Rosenberg J. RFC 3428: *Session initiation protocol (sip) extension for instant messaging*, December 2002. IETF.

Want, R., Hopper, A., Falcao, V., Gibbons, J. *The active badge location system*. ACM Trans. Inf. Syst. 10(1) (1992) 91--102

BROWN, P. J., BOVEY, J. D., AND CHEN, X.
Context-aware Applications: from the Laboratory to the Marketplace. IEEE Personal Communications 4, 5 (October 1997), 58–64.

SCHILIT, B. N., ADAMS, N. L., AND WANT, R. Context-aware computing applications. In IEEE Workshop on Mobile Computing Systems and Applications (Santa Cruz, CA, US, 1994).

GU, T., WANG, X. H., PUNG, H. K., AND ZHANG, D. Q. 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).

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

CHEN, H., FININ, T., AND JOSHI, A. Using OWL in a Pervasive Computing Broker. In Proceedings of Workshop on Ontologies in Open Agent Systems (AAMAS 2003) (2003).

M. Papazoglou and D. Georgakapoulos. *Service oriented computing*. Communications of the ACM, 46(10):24--28, October 2003
Salvador, T Anderson, K. Practical considerations of Context…. Ubicomp 2003. Slå upp!!!