Supporting Context Awareness in Ubiquitous Service Environments

Licentiate Thesis

By

Martin Jonsson
The FUSE research group
Department of Computer and Systems Sciences
Stockholm University and Royal Institute of Technology
Forum 100, 164 40 Kista

martinj@dsv.su.se

Abstract
Computing devices are becoming wireless, increasingly smaller and embedded into other artefacts. Some of them are mobile while others are built into the environment. The novel technologies are also becoming more dependent of communication with other computing devices over different kinds of networks. These interconnected devices constitute locally distributed computing environments that will set new requirements on the design of software systems.

A framework, which provides means to model and design this kind of computing environments, is introduced under the notion of *Ubiquitous Service Environments*, describing locally distributed systems with interconnected software services.

Two specific problems are addressed for this kind of computing environments: Firstly how to acquire and distribute information about a user's context in order to enable new kinds of application behaviours. The other problem that is being addressed concerns how to create means for context aware service discovery, e.g. how a mobile computing device can discover computational resources in the vicinity.

The Context Shadow system is presented, which makes it possible for services to ask questions about a person's current context, and specifically about the computational services that are relevant to that context. The system has also been extended with a sensor platform in order to support the deployment of sensors used to collect context information.

Acknowledgements

For supervision: Calle Jansson
For joint work and support. Several persons have actively participated in the work that led to this thesis: Patrik Werle, Johan Mattsson, Fredrik Kilander and Peter Lönnqvist.
For reading and commenting: Hillevi Sundholm and Li Wei.
For valuable input and coffee room chitchat (not to neglect!): Jakob Tholander, Annette Hulth, Patric Dahlqvist, Lisa Brouwers , Maria Croné and Karin Hansson.

and

For continuous encouragement, discussions and support: Hanna (love you!)
Finally, for putting things into perspective: Alba

List of papers

Paper A: Context Shadow: An Infrastructure for Context Aware Computing
Submitted for review based on invitation to the journal Applied Artificial Intelligence.
A shorter version of this paper was presented at the workshop Artificial Intelligence in Mobile Systems (AIMS) in conjunction with ECAI 2002, July 22nd, 2002. Lyon, France.

Paper B: Building extendable sensor infrastructures for pervasive computing environments
Technical Report 2002-019, Dept. of Computer and Systems Sciences, Stockholm University/KTH.

Paper C: A Ubiquitous Service Environment with Active Documents for Teamwork Support
Presented at the Ubicomp 2001 Conference, Atlanta, Georgia, September 2001.

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 a situation 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 computing or service centric computing (Espinoza 2002, 19). 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).

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. Even if this approach is promising in many ways, the actual deployment and design of such systems in an environment where computers are ubiquitous will create a number of new challenges that have to be dealt with, where this work will try and address only a small subset.

1.1 Problems and objectives

In this work an approach to design systems for ubiquitous or post pc computing environments is presented. This approach goes under the name of *Ubiquitous Service Environments* and can roughly be described as a combination of service oriented computing and context aware computing. This work focuses especially on two issues that are central to this approach:

Acquisition and provision of context information in Ubiquitous Service Environments.

Giving computer systems access to sensory information that in some sense describes the context in which an interaction with a computer system takes place, will open up for the creation of entirely new and interesting types of applications. These opportunities become even more interesting for ubiquitous computing systems where computing devices to a greater extent are integrated with our everyday environments.

The problem that this work addresses within this context regards if and how some generic support can be provided both for the deployment of new sensors in an environment as well as for the distribution of existing sensor information to applications in a meaningful way.

Context aware service discovery in Ubiquitous Service Environments.

In order to realize the kind of open service oriented systems that the Ubiquitous Service Environments approach describes, 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 a mechanism that uses context information in the service discovery process. In this way services could e.g. find other services in a shared location.

1.2 Methodology

The work in this thesis is very much 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. The research process has differed somewhat between the research issues presented in this thesis. An outline of the two work processes is presented briefly:

The first issue regards 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 through analyzing existing research within the area, as well as identifying the domain in which the work would take place. 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 second research problem, regarding how to collect and distribute sensor data to applications, followed a slightly different work process. The major difference regards the identification of the problem. In the previous example the problem was identified from evaluating the work of others. In this case on the other hand, the problem was identified merely from examining own experiences from earlier research projects. From the identification of the problem a system model was created. Only parts of functionalities of the system were then implemented. No thorough evaluation has so far been performed.

1.3 Thesis contributions

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

1. *The introduction of the design approach called Ubiquitous Service Environments.* This design approach, which is formulated within this work, can be seen as one step towards a realization of the somewhat fuzzy visions embraced by the ubiquitous

computing research domain. The contribution consists both of a conceptual model describing how these future systems should be designed and thought of, as well as some implementations, such as the fuseONE system, that could act as examples of how instances of these kinds of systems could be designed.

2. *An approach to enable acquisition and provision of context information in Ubiquitous Service Environments.* This contribution consists of two concrete parts: Firstly a design and implementation of a system, Context shadow, that provides an infrastructure for provision of context information to applications. The second part consists of a design of a system as well as an implementation that supports the deployment of sensors in physical environments.
3. *An approach to enable context sensitive service discovery in Ubiquitous Service Environments.* A key functionality of the Context shadow system mentioned above is 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, such as the fuseONE system.

1.4 Summary of papers

1.4.1 Paper A. Context Shadow: An Infrastructure for Context Aware Computing

This paper consists of an in depth explanation of the Context Shadow system, as well as descriptions of two applications where the features of the Context Shadow has been utilized.

The Context Shadow system provides means for context sensitive service discovery in order to decide which services that are relevant to a specific usage situation. It can also provide context information such as which persons that are currently present in a room, as well as organisational context information such as project membership of certain people etc.

In the Context Shadow infrastructure, different kinds of stable entities (typically persons and places) can be provided with a context server. Sensors and services tied to those entities will then announce themselves in the corresponding context server. By dynamically creating links between the servers, e.g. based on information from sensors, a searchable web of services and context information is created.

The contributions of the co-authors of this paper mainly concern the example applications in which the Context Shadow was utilized and tested. The Context Shadow system itself is entirely a contribution of the author of this thesis.

This paper has been submitted to review based on invitation to the journal Applied Artificial Intelligence. A shorter version of this paper was presented at the workshop Artificial Intelligence in Mobile Systems (AIMS) in conjunction with ECAI 2002, July 22nd, 2002. Lyon, France.

1.4.2 Paper B. Building extendable sensor infrastructures for pervasive computing environments

In this paper experiences from earlier work with sensors in post desktop computing environments are described and summarized. Problems regarding reusability, scalability and ease of deployment of sensor systems are identified. An infrastructure is presented consisting of a sensor platform, to be embedded within the rooms of a building, which is then integrated with the Context Shadow system that is used for the distribution of sensor data. The infrastructure supports development of sensor-based applications on a low level and eases reusability of sensors and sharing of sensors between applications, as well as supports discovery, transport and refinement of sensor data.

The work behind this paper is a cooperation where the co-author Johan Mattson have contributed mainly on the design and implementations on the actual sensor platform.

This paper is published as a technical report at the Department of Computer and Systems Sciences, Stockholm University/KTH.

1.4.3 Paper C. A Ubiquitous Service Environment with Active Documents for Teamwork Support

In this paper the notion of Ubiquitous Service Environments appeared for the first time. Even though the meaning of the term has been more explicitly defined since then, the system described in the paper still fits well as an example of how such a USE system could look like.

The system described in the paper is a meeting support system called *fuseONE* which targets the problem of document management in project meetings. The system contains 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. This system relies heavily on information about the current locations of people. This information was provided by the Context Shadow infrastructure described above.

The paper also presents a pilot user study, describing the usage of the system in a real setting: A group of teachers planning a new course over a series of meetings. The results of the study show that the feature of seamless document exchange was highly appreciated. Some changes of the work process were also noticed. The meetings tended to a higher degree to start where the earlier meeting ended, thanks to the easy retrieval of documents from the Active Document service. Also a lot of the generation of documents were created in the meeting instead of in-between meetings.

This paper was the result of cooperation between several people, where the author of this thesis was responsible mainly for adding the context awareness functionality to the system components presented in the paper.

This paper was presented at the Ubicomp 2001 Conference, Atlanta, Georgia, September 2001.

1.5 Thesis overview

Chapter one in this thesis gives a brief background as well as presents the research questions, methodology, summary of papers etc. 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 design approach, which could be understood as a frame of the rest of the work, in terms of constraints and needs.

Chapter three consists of an investigation of the problems regarding provision and acquisition of context information, both in general and with respect to the properties given by the Ubiquitous Service Environments domain.

In chapter four 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. The fifth chapter describes the Context shadow system and how it is used to tackle the problems of distribution of context information as well as the issue of context aware service discovery. Chapter six describes the design of a platform that supports easy deployment of sensors in a room. The work ends with some conclusions and discussions about future work in chapter seven.

Chapter 2

Ubiquitous Service Environments

The research problems considered in this thesis can be placed in the conjunction of three different research areas, which in this thesis will be referred to as; post desktop computing, context aware computing and service oriented computing. These areas will be introduced briefly. The conjunction of these research areas constitutes its own domain with its own specific properties and constraints.

2.1 Post desktop 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 the physical and social situation in which computational devices are embedded. One goal of context-aware computing is to acquire and utilize information about the context of a device to provide services that are appropriate to the particular people, place, time, events, etc. For example, a cell phone will 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

The domain of context aware computing, and the different problems regarding support for context aware computing will be explored in more detail in the next chapter.

2.3 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 do 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 a 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?

2.4 Ubiquitous Service Environments

The conjunction of the three areas above creates a new domain with some specific characteristics. This chapter will outline and try to define a design approach for designing systems within that domain, an approach that will be referred to as *Ubiquitous Service Environments*.

So far the manufacturers of novel 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, meaning standalone applications, maybe with some backend server support. Some applications might have to take into account some interoperability issues, such as how to export and import data to other applications etc. An example might be a calendar application on your PDA, a standalone application that sometimes are synchronised with the calendar application on a PC.

A more radical way to design these computing devices is to use a service oriented perspective, where the devices becomes potential nodes 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. By sharing devices 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. This approach would require that both the hardware and software in some ways should be designed to be open for communication with other software parts. On the hardware side some sort of network connection might be sufficient, a property already present in most novel computing gadgets[1]. 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.

2.4.1 Towards a new orientation

In order to examine some of these problems, as well as the benefits with these kinds of systems, we have come up with a design approach 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. Acknowledging that the

[1] An example of this is the Bluetooth technology, which is a wireless connectivity specification that allows wireless connectivity between computers, mobile phones and other portable devices. (www.bluetooth.org)

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 an implementation of the design approach. One such USE model contains a number of elements:

- Services: Software entities performing different kinds of tasks.
- Tools: A service or a composition of services providing distinguishable affordances for persons to support an activity.
- Devices: Physically standalone containers of hardware resources, user interface elements and services.
- Persons: Users of the USE, individuals or groups.
- Activity: One or several persons performing a task.
- 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 centred view is partly 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.

A USE differentiates between the *physical space* and the *logical space*. The physical space concerns the physical environment containing the supported activity. It includes properties and restrictions of non-technical 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 the exclusion of entities that are not relevant to the activity in focus, thereby describing the boundaries of the USE. In a scenario with a broad usage of these kinds 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.

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 have different hardware resources that services in the USE might want to utilize, e.g. some kind of user interface, such as audio or display functions. The devices can be public and shared and probably statically installed in some location, or they can be owned by some person.

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"*

The most central property of a service in a USE 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 making it possible to create compositions of services that can provide more complex tasks than the individual services.

Since the web of interconnected services can be hard to grasp by the users of a USE, the *tool* abstraction is introduced, providing clearly 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, and thereby resembling a normal software application.

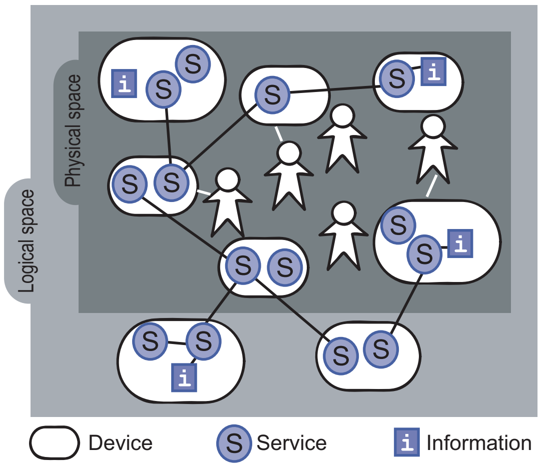

Figure 1: A snapshot of a USE supporting a specific activity.

In order to create systems according to the model presented above, a number of problems have to be tackled. The most central problems concern how to make the software service environment open and dynamic, so that independent services can find each other and co-operate in a meaningful way, all this in a dynamically changing environment where people and services may come and go. Other central problems concern how to create new models for interaction required in the environment.

2.4.2 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 SOAP (a specification for using XML in simple message-based exchanges), and the JXTA protocol for peer to peer networking.

2.5 Conclusions

In this chapter the notion of Ubiquitous Service Environments has been introduced and explored to some extent. To summarize the concept of USEs, one could say that it provides both a terminology for describing locally distributed computer systems as well as a design approach, giving some indications on how such computer systems should be designed.

This contribution consists both of a theoretical framework as well as of some examples of successful implementations.

The concept appeared for the first time in the third paper of this thesis; paper C, where the notion was used without being fully explained. A more thorough description is then given in paper A.

Chapter 3

Acquisition and provision of context information

One of the research questions 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 analyse the concept of context and how it could be understood and used in relation to computer usage and applications.

3.1 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 mainly 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.

3.2 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, 1-34). 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 non-trivial 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 (Suchman 1987).

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* computing, but since the previous term is so widely used it will be continuously used in this thesis.

3.3 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.

3.4 Using context in applications

Context aware applications can be created that makes use of context information in many clever 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).

3.5 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 the ICQ[2]-instant messaging application, 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.

[2] The ICQ software by ICQ Inc, can be found at www.icq.com.

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[3] readers, fingerprint recognition sensors, short distance radio (RFID) sensors and different kinds of image processing sensors like face or object recognition sensors.

This identification method differs a bit from other more explicit methods e.g. such as the ones used for collecting information about the physical properties of a specific location. This kind of information is mainly gathered directly using different kinds of environment sensors, like thermometers, photo resistors, pressure sensors etc.

3.5.1 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.

3.5.2 Classification of sensors

A brief examination of the usage of sensors in pervasive computing applications shows that the majority of the sensors can be sorted into four groups according to some specific properties.

- Identifiers: Sensors that can acquire an ID or other limited information from a "foreign" object. Sensors of this kind include RFID readers, barcode readers, iButton readers etc. Identifiers could for example be used to identify persons and objects that arrive in a room or to various partially defined locations of the room. Some of the identifiers are able to carry data and others just an ID. The identifiers carrying data might hold specific information regarding their purpose and use.

[3] iButton is a product line by Dallas Semiconductors; www.ibutton.com

- Passive environment sensors: Sensors that measure something continuously. These sensors are passive in the sense that they do not react on changes of the data they measure. The passive sensors are mainly used when changes in the environment do not require immediate response, and when the fetching of sensor data is trigged by events that do not depend on the sensor value. Sensors of this kind could be temperature givers, linear photo resistors, sound level microphones, GPS receivers etc.
- Active environment sensors: Sensors that actively indicates changes in the environment. Sensors of this kind include different kinds of alarms, motion detectors etc.
- Streaming sensors: Sensors that capture series of data where not only the latest piece of data is of interest. This group of sensors include video cameras, microphones etc.

In the analysis, two other groups of gadgets also appear that are not technically sensors, but show some similarities to sensors especially regarding implementation:

- Interactors: Tangible interaction devices that can give input to local applications. This could be push buttons, knobs, sliders etc. The interactors can often be implemented using the same components as the other sensors mentioned.
- Actuators: Simple output devices that can be used to provide different kind of feedback and status information, often related to the sensors mentioned above. This could be light emitting diodes, simple speakers etc.

This categorisation of sensors and sensor usage has been used as a requirement specification for a system that supports the development of sensor based applications. This system will be described in detail in chapter five.

3.6 Existing support for context aware computing

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 has realised 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. Some examples of such systems will be given in chapter six.

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. Examples of these categories of systems will be presented in the related work section in chapter five.

3.7 Using context information in Ubiquitous Service Environments

When trying to create support for context awareness in the Ubiquitous Service Environments (USE) domain, one has to take into account the specific properties and constraints of this domain. To recapitulate some of the central aspects of a USE-environment one could state that it should embrace the following properties:

- A USE consists of physical environments that are in some senses enriched with different kinds of computing devices.
- Some devices can be seen as fixed parts of the environment, publicly available to use by anyone. Other devices are mobile and personal and can be carried around between different environments.
- The devices are inhabited by software services that are openly available to use by the users or other services.

The specific problem to solve is thus how to provide the different services within the USE with information about relevant information of other entities in the USE, be it the location of people or the temperature in a specific room. One of the most important functions that a support system for context aware computing could fill in a USE is to provide support for context aware service discovery, i.e. how to find other services in the USE that are in some sense relevant to the querying service. This specific problem will however be discussed in detail in the next chapter.

Providing context information to services might also enable the creation of more complex services, services that might use context information to in order to make different kinds of decisions. Such reasoning services or agents can play important roles in the design of a USEs. Agents could be used both in order to support the basic functionality of the USE, as well as to provide advanced services to users. Problems related to the basic functionality of the USE could be the selection of relevant services as well as figuring out what each service actually does, problems that could be tackled using different kind of reasoning strategies. Agent based services could also monitor the behaviours of persons in order to try and identify the occurrence of specific activities, and then try to present an appropriate set of tools to support that activity.

In order for the agent based services to make these clever decisions they need reliable information regarding the physical and computational environments. Since USEs are open systems that can be infinitely extended with new services, persons and places, the knowledge about the real world and real world entities can not be statically encoded within the agent, but must be extended and modified by continuously interpreting information regarding the agent's computational and physical context. One way to do this is by connecting the agent directly to different kinds of sensors that could capture properties of both the physical environment as well as the logical or computational environment. This approach requires the agent to perform lots of interpretations in order to extract meaningful information that could be used in some decision making task. In order to minimize the interpretation effort, the context information could instead be provided by some external source that would provide a shared view of the USE containing dynamically updated representations of users, places and services.

3.7.1 Using an infrastructure approach for context sharing

One way of creating the kind of support described above is to provide the information through some kind of infrastructure that the different services would share. It has been pointed out that there is a general need for infrastructure to support ubiquitous computing environments (Norman 1999; Huang et al. 1999), and specifically to support context aware computing (Dey et al. 1999). An infrastructure can be described as a well-established, pervasive, reliable and publicly accessible set of technologies that act as a support for other systems. There are several reasons why an infrastructure could be a good approach to provide context information to applications. Infrastructures generally provide means of sharing information as well as computational services. In the USE setting this would include the sharing of information about the physical world collected by different kinds of sensors, as well as information about available services. An infrastructure can also be seen as a way to decouple the software services from the sensor hardware, making it easier to perform modifications or extensions of different parts of the software system.

3.7.2 Sharing context model

The idea of sharing some general context model between applications is an approach with both pros and cons. Lots of discussions within the context aware computing community concern how to find general representations of context data. These discussions will of course never lead to a "final solution" since the context representations in themselves are part of a context and has to be created with a specific usage in mind. Any description of the world will be incomplete since a complete description would have to consist of the world itself. Even if one could agree on a shared model or ontology within a small community, the model is a static representation of a dynamically changing world, where every change in the world would require a renegotiation of the ontology within the community. Therefore one could argue that sharing a common context model between applications that are designed to support different tasks is generally a bad idea.

A shared model approach however also has benefits. A comparison could be made with the successful use of *standards* within different areas of computer science, which could be seen as a kind of shared ontology or model. The broad agreements on the standards regarding the Internet communication protocols are probably a large part of the reason that the Internet has become such a success.

In most cases when trying to incorporate sensor data into applications, a context model is also absolutely necessary. Without connecting the sensor data to some entity that the sensor in some sense is describing, the output from the sensor will only be meaningless numbers. Thus sensor data becomes context information first when you can claim that it describes a property of an entity, be it a real world object or an organisational or computational entity. By creating this mapping between a sensor and an entity, a context model has been created, consisting of a representation of a real world entity as well as a description of some property of that entity.

The open nature of the USE approach makes a shared context model almost a necessity, if one wants to create functions such as location based service discovery or other similar functions. To avoid the problems regarding context models described above, the most viable approach seems to be to use a shared context model that is simple enough to suite most needs but still detailed enough to make the distributed sensor data meaningful. This approach then makes it possible for services in the USE to create more extensive context models on top of the basic shared model, which can then be more explicitly targeted to a specific task.

In the Context Shadow infrastructure that will be described in chapter 5, such a basic context model consists of unique representations of a few real world entities, such as people and places, as well as the relations between these entities.

3.7.3 Deploying sensors in a USE

One central idea of the USE-approach is that not only computational elements but also the physical space can be tailored in order to support a specific activity. One part of such a tailoring of a physical space might include the addition of some set of sensors that could be used by different services as part of a more specific tool. Certain basic functionalities in the USE, such as service discovery, could also utilize certain sensors installed in the different spaces available. This implies a need for physical spaces to be provided with some sensor capabilities that provides support for the basic context dependent functionalities of the USE, as well as means to easy extend these sensory capabilities in order to support some more specific activity.

Given the service oriented nature of the USE approach, it seems natural to implement a sensor in terms of a service that provides the data in question to other services in the environment. As we will see later on in this work, some of the sensor information is however needed in order to keep the simple context model up-to-date, motivating a tighter coupling to the context infrastructure compared with other services.

Chapter 4

Context sensitive service discovery

A key problem when trying to realize the concept of Service Oriented Computing is the problem of service discovery. 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.

4.1 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[4] 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.

4.1.1 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.

[4] Universal description discovery and integration (UDDI). http://www.uddi.org, 2001.

4.1.2 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.

4.1.3 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.

4.1.4 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.

4.1.5 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.

4.1.6 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.

4.1.7 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.

4.1.8 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.

4.2 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[5] 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.

4.3 Service discovery in Ubiquitous Service Environments

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.

[5] Bluetooth is a wireless connectivity specification that enables electronic devices to talk spontaneously and allows instant wireless connectivity between computers, mobile phones, and portable devices, (www.bluetooth.org).

Chapter 5

The Context Shadow infrastructure

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:

1. Support for context aware service discovery.
2. Organization of services and context information in meaningful collections.
3. Context information for applications derived from sensors and other services
4. 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.

5.1 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.

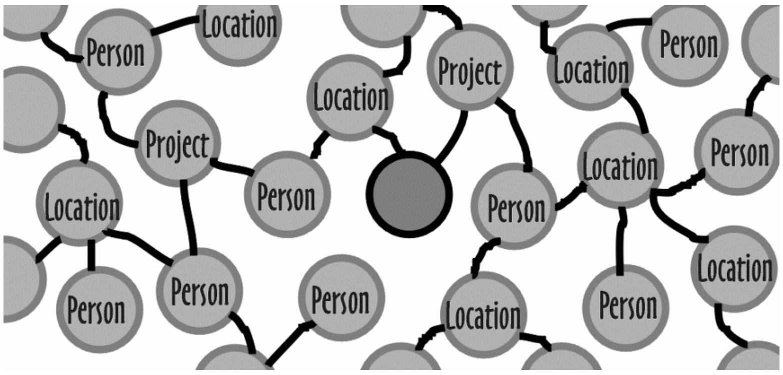

Figure 2: The linked context servers create a searchable space, where the topology of the space is part of the context information.

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.

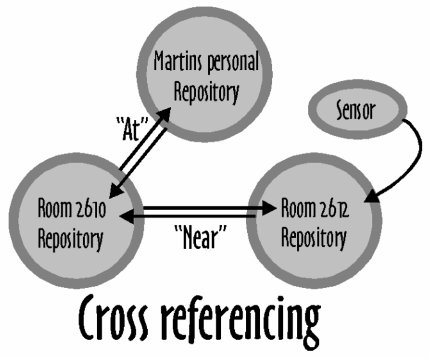

Figure 3. The context servers are linked with references. By following the links it is possible to acquire context information from other context servers than your starting point.

5.2 Context sensitive service discovery

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.

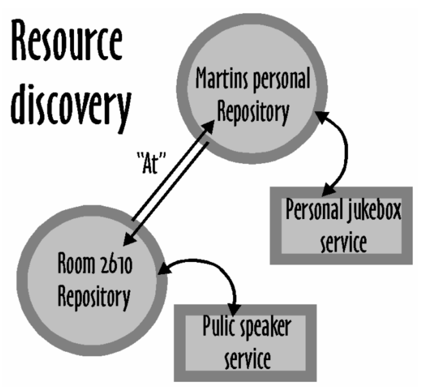

Figure 4. Using Context Shadow, a jukebox service finds a speaker service at the user's current location.

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.

In one of the prototypes described below, 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 Jini services that were irrelevant to the actual situation.

5.3 Refinement of context data

Any service should be able to use any data posted into a repository. This raises questions on the format of the data. Different services might want the data in different formats or at different levels of abstraction. Introducing Context Refiners solves this problem. A Context Refiner reads data from the context server then transforms it and adds the new information. The refiners can also be used to handle contradictive data. One example of such a Context Refiner deals with sensor information about a user's current location. Since a person only can be at one place a time, location data that differs is contradictive per se. In the implemented refiner each piece of location data comes with a certainty factor and a timestamp. The Context Refiner uses this information to calculate the most probable location and then removes the more improbable location data.

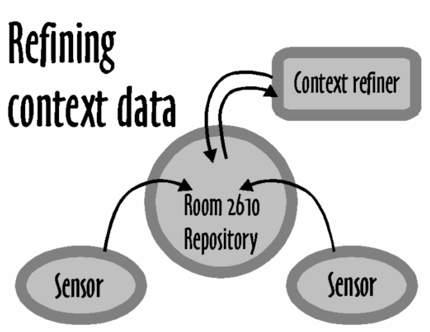

Figure 5. A Context Refiner reads information from a context server, performs some operation or transformation on the data and then posts the result back to the server.

5.4 Example implementations

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.

5.4.1 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.

5.4.2 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 what other services that are relevant to its user's context, 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 2003), 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.

5.4.3 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 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.

5.5 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). The context toolkit is primarily supporting the creation of context aware systems on an application level by introducing context widgets, representing specific pieces of context information, which makes it possible to incorporate sensor information in application similarly to how you work with e.g. GUI-elements. The Context Toolkit also provides means for aggregation of data from different sensors, as well as support for translation of low-level sensor data to high-level data that can be used by applications directly. 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 event heaps into an infrastructure, nor will the event heap communicate with services that have other properties than being in the room.

5.6 Conclusions

This chapter has presented the Context Shadow system, which aims to tackle two main problems: The enabling of context sensitive service discovery as well as distribution of context information to applications.

In the Context Shadow infrastructure services and sensors are gathered in collections related to real world entities. The collections are then dynamically interconnected so that the services in one collection can get easy access to the entities in the other collection. The system provides a query interface through which the services can get information about e.g. the persons and services in a user's current location.

For service discovery it could act as a complementary addition to traditional network based service discovery protocols. The approach with interconnected collections allows for really simple services to still display a context aware behaviour, since the services using the system does not have to implement a specific context model to use the infrastructure. The distributed nature of the system also makes the system scale well.

Context Shadow has been implemented and tested with several different applications where it has enabled several new and useful functions.

The Context Shadow system is primarily described in paper A, while the example implementations where more extensively covered in paper C.

Chapter 6

A platform for acquisition of sensor data

In order to deal with the problems with deployment and sharing of sensor data we propose an infrastructure supporting the development of sensor-based applications, and especially the development of so called intelligent buildings. The architecture we have considered includes an embedded computer platform with support for a number of different sensors as well as higher-level support for distribution of sensor data to applications.

Since a sensor could be anything between a video camera and a push button, it is very hard to create a support system that can handle all kinds of sensors. We have therefore restricted the supported sensors to typical instances of some of the different groups of sensors identified earlier, where each group puts different requirements on the system:

Identifiers: To support these sensors the system needs to be able to listen for new objects that arrive, ask for currently present objects and read data and IDs from these objects.

Active environment sensors: For these sensors the system has to be able to generate some sort of events when a sensor value is changed or a threshold is reached. The event should notify or trigger listening applications, so that they can react on the changes immediately.

Passive environment sensors: The system must provide means for applications to fetch the values from these sensors with small latencies. It should also be possible to change a passive sensor to an active sensor, so that it will generate events when the values reach certain thresholds. The system must therefore provide means to set these thresholds.

Interactors: We choose to include simpler forms of such interaction devices among the supported sensors, with the constraint that it should be possible to model the interactor in terms of the sensor types above.

6.1 Sensor platform architecture

The proposed infrastructure consists of two parts: a sensor platform and a software infrastructure for distribution of sensor data. The sensor platform consists of a small, network enabled and programmable micro-controller device where a number of sensors can be attached in parallel. The micro-controller contains firmware and drivers for the sensors as well as support for network communication.

The idea is that these platforms could be integrated in a limited physical space such as the rooms of a building. A number of different sensors could then be connected to the platform according to the current needs.

The platform could later easily be modified to cover future modifications and additions of applications in that locale. To make the sensor platform reusable and easy to adapt to new sensors we propose that the controller should support a modular high level programming language. The high level programming language increases the possibilities to create abstractions and different kinds of sensor templates. In our reference implementation we have been using the TINI board from Dallas Semiconductors, which is a micro controller supporting Java Me.

Since the computing power of micro-controllers is limited, we propose that manipulation and distribution of sensor data is performed on an external sensor server, running on a more powerful computer somewhere on the network.

For the software on the micro-controller we propose a layered architecture as following:

Sensor connection library: Software that enables communication with the connected sensors in a unified way. The API contains a set of routines needed to communicate over any of the available communication ports. The ports or communication channels could be e.g.:

- Instructions to read and write to general I/O ports on a micro controller
- RS-232 comm. port
- Dallas OneWire network
- IR or IRDA ports
- RF or Bluetooth

Sensor drivers: Native implementations to generalize the most basic instruction set for some standard sensors or peripherals. Examples of the operations that have to be served by the driver are read and write data operations, initialization of sensors and forwarding of interrupts.

Sensor address layer: Initializes the sensors and associates specific hardware addresses with access IDs and symbolic descriptions that makes it possible to access the sensors without digging into low-level details of how they are connected.

Sensor access layer: provides a middleware API to the sensors and implements strategies for sharing the raw sensor data. The access layer provides methods to request sensor data and keeps track of event- or poll-based subscriptions for sensor data.

Sensor communication server: Controls the communication channels for distribution of raw sensor data and administrates the connections to one or several context servers. This layer might also convert the sensor data to fit the communication channel i.e. translations back and forth to byte-packets for each transmission of data.

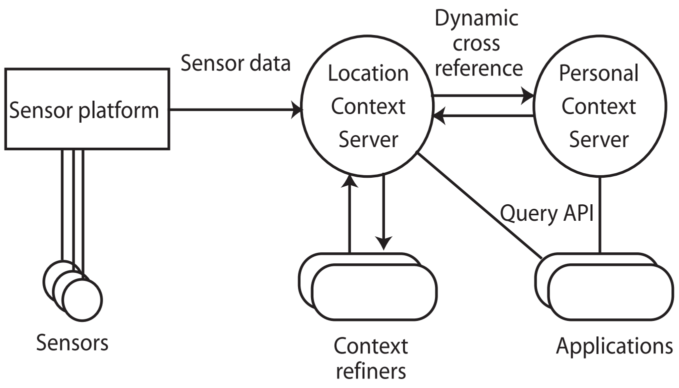

Figure 6. A schematic view of the sensor infrastructure

6.2 Example implementation

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. There was also a knob in the middle of the table that the participants could use to select an acceptable “disturbance level” for the meeting. The knob was created using a potentiometer connected to an embedded computer according to the sensor platform design described above. The microcontroller used in this implementation was a TINI[6] board from Dallas Semiconductors, which is a network enabled computer board that runs Java Me.

Not all of the functionalities of the sensor platform design described above were actually implemented, but only those needed to realise the knob application. An evaluation of this implementation showed that the chosen microcontroller was not entirely suitable to act as a sensor platform according to the design above. The computational and especially the memory capabilities were slightly too limited to allow for a complete instantiation of the system design, the platform could e.g. not handle a direct communication with the context shadow repositories. Another feature that would have been useful on the TINI board is support for

[6] TINI is a product line by Dallas Semiconductors; www.ibutton.com/TINI/

wireless communication, since a wireless board would further increase the ease of deployment.

6.3 Related work

Other projects have tackled the problem of providing multi purpose sensor platforms. Smart-its (Holmquist et. al 2001) is a modular sensor platform that uses wireless communication to allow multiple smart-its to communicate to with each other. The sensor platform hosts a set of common sensors: a 2-axis accelerometer, pressure sensor, light sensor, temperature, microphone and a ball switch / motion-detector. There are also actuators, a LED and a piezo speaker. The platform can be extended with additional sensors using a set of I/O ports, serial and I2C connectors. A library that makes the sensors and communication useable without needing to know all implementation details is also provided for the sensor board.

The Berkly Sensor Motes (Conner et. al 2001) provides a platform similar to the smart-its with a slightly stronger focus on the wireless communication and the software platform they call TinyOS which is used on their embedded sensor platforms. In an example application a number of sensor platforms were connected to motion sensors. The sensor data were then captured by an application that provided information regarding free conference rooms.

Both the platforms described above have put a lot of effort into issues of power consumption as well as minimizing the physical size of the platforms. There is also an ambition to make the platforms mobile by using wireless communication. These ambitions however put a lot of requirements on the platform, constraining it in many ways. Since the system in this work is focused on stationary implementations of the platforms we can chose to use a bigger size platform with better processing and memory capabilities, making the platform easier to program, and thus to modify for different purposes, since you can use higher level programming languages.

6.4 Conclusions

This chapter has presented a design of a sensor platform that is supposed to be embedded in the rooms of a building as a shared resource to be used by different kinds of context aware applications. The system provides support for adding new sensors to the board by providing software templates for some standard type of sensors as well as support for distribution of sensor data to applications.

A partial reference implementation of the sensor board based on the TINI board shows some problems related to the hardware platform, which leads to the conclusion that a hardware platform tailored for this purpose would be preferable. The sensor board system is described in some more detail in paper B.

Chapter 7

Future work

This work has addressed some problems concerning the design of highly distributed post desktop computing environments. A design approach for realizing such systems, called Ubiquitous Service Environments, has been introduced. Within this approach two specific problems have been targeted: How to support the collection and distribution of context information and how to support context aware service discovery.

The major contribution with respect to these problems is the Context Shadow infrastructure, which aims to target both issues regarding context aware service discovery and distribution of context information. The system has been implemented and tested with several different applications where it enabled several new and useful functions.

This infrastructure was also extended with a design of a sensor platform that is meant to be built into rooms of a building. The platform provides support for easily adding a range of different sensors that might be required by applications running in the environment.

These systems could of course be improved and extended a great deal. Regarding the Context shadow infrastructure, several improvements could be imagined. One such addition to the system could be to extend the interface to the infrastructure to also handle queries over the HTTP protocol. This would make the infrastructure independent of programming language, allowing services to post XML-based queries to a specific URL, e.g. over the SOAP protocol.

Another extension concerns the creation of new context servers. At the moment new context servers must be created manually by adding information in a property file. A first step would be to create a web-based user interface that would allow users to create new context servers in an easier way. A more radical change would be to create the ability for context servers to be created in an ad-hoc manner. You could imagine sensors that detects when two persons are close to each other but can provide no information about their physical location. In this scenario you would want to create a temporary location representation only to connect the persons. Support for events is also currently lacking in the system. Events might be useful in order to notify a service of immediate changes in the environments or to trig some action when a certain contextual condition is detected.

Regarding the sensor platform, alternative microcontrollers will have to be tried out that might be more suitable for the proposed platform design. In the near future the Berkeley Sensor Motes (Conner et. al 2001) will be examined as a possible replacement.

From a research perspective one could imagine at least two interesting lines of continuation of this work:

A continuation of the USE approach focussing on a closer integration of the physical and computational space. This might include a more extended context model that more explicitly maps to the entities in the USE-model. Such a model could e.g. include information about devices, information and activities.

Another possible line of research could concern how context information managed and created in everyday computing usage by normal users of the system. By using a simple context model and making it clearly visible to the users of a system, as well as creating means to easily contribute and alter the system, might have several benefits. Firstly the users might be more likely to understand the behaviour of a system, something that might be problematic in systems that perform actions based on real world models. Secondly the users might be more interested in the maintenance of the system and also to adapt it to their own needs.

Bibliography

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.

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.