Submitted for Publication in {Applied AI; AI in mobile Systems}. Prepared with the T&F Journal Template.

Context Shadow: An Infrastructure for Context Aware Computing

Martin Jonsson
Department of Computer and Systems Sciences, Stockholm University, Sweden

Patrik Werle
Department of Computer and Systems Sciences, Stockholm University, Sweden

Carl Gustaf Jansson
Department of Computer and Systems Sciences, Stockholm University, Sweden

Address Correspondence to: Martin Jonsson, Dept. of Computer and Systems Sciences Stockholm University/KTH, Forum 100, 164 40 Kista. Email: martinj@dsv.su.se

Abstract. *In ubiquitous computing environments, hardware and software services will create huge and ubiquitous service spaces. These service spaces will need to be organized in meaningful ways so that the services can be easily accessed and utilized by other services and humans. Context Shadow is a system that makes it possible for services to ask questions about a person's current context, and specifically about what services that are relevant to that context. In the system, sensors and services are organized in meaningful collections, creating a searchable topology of context information.*

Introduction

Providing information about users' context to applications could be very useful in several ways. The behavior 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. Under the notion of context aware computing there have been several attempts both to define what context really is, and how to design applications that use it (Schilit et al. 1994; Dey 2001)

In this paper context aware computing will be viewed from a ubiquitous or disappearing computing perspective, a vision firstly described by Mark Weiser (Weiser 1991). More specifically we will examine how context information can be used in order to design a specific type of systems that can be described as *ubiquitous service environments*. The assumption we make is that our everyday environments will become increasingly more readable and controllable. This is due to the ability to embed computers into everyday objects, and to connect them to the Internet. We also see an increasing use of personal mobile computing artifacts. 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 carries several computing devices, and also shares a number of devices with other people. These new stationary and mobile artifacts will provide different kinds of services either to a user directly or it will be accessible to other software services over the Internet. This results in a potentially infinite space of services that are constantly reachable for humans or other services over the Internet.

Will this new situation in any way affect the design of computer systems that are aimed to target some specific needs? So far the manufacturers of novel computing devices (PDAs, projectors, cellulars, etc.) has chosen to design 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 designed similarly to the way systems are being designed for standard 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 is synchronized with the calendar application on a PC.

A parallel and more radical line of development of these computing devices is to see them as potential nodes in a larger system; an approach that makes 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 requires 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. 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 view will impose a number of new problems that has to be dealt with within areas such as security, usability, stability, etc.

Modelling the new environments

In order to achieve a better understanding of these kinds of highly distributed systems, one can choose to describe them in terms of ubiquitous service environments or USEs (Werle et al. 2001) This model describes 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. An instance of a USE is constituted of a number of elements:

- Services: Software entities performing different kinds of tasks.
- Tools: A service or a composition of services providing a user interface with which persons can support an activity.
- Devices: Physically standalone containers of hardware resources and services.
- Persons: Human 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, persons, etc.
- Logical spaces: The subset of services and information being used in an activity, and how they are structured.

Since the new computing environments might consist of a number of persons sharing both devices and software services, one has to find an appropriate way to delimit the model, so that only the relevant entities has to be examined. In the USE model this delimiter is some *human activity* consisting of one or several persons that are engaged in some sort of task. Instead of referring to users of the system the notion of *persons* is used. We believe that the term user 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.
The USE model differentiates between the *physical space* and the *logical space* of activity. 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 reason for the notions of physical and logical spaces is to enable exclusion of entities that is not relevant to the activity in focus, thus defining the boundaries of the USE. This might seem like a trivial task but in a scenario with a broad usage of these kinds of open service environments, each computing device could contain a great number of services, thereby creating a truly ubiquitous service environment. Each specific activity uses 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 particular combination of services and devices.

The computing devices in a USE bridges the physical and logical spaces since an instance of a service always reside on a physical *device*. The devices have different hardware resources that services in the USE might want to utilize, typically some kind of user interface, such as audio or display functions. The devices can be public, thus shared and probably stationary installed in some location, or they can be owned by some person.
The basic building block in a USE is however the actual services. Behind this term hides a number of different types of software components with rather different properties. The common broad definition of a service in a USE reads:

> "A service is a software component that performs a task for a user directly or for 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 perform more complex tasks than the individual services.
Since the web of interconnected services can be hard to comprehend by the users of a USE, the *tool* abstraction is also 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 thereby resembling an ordinary software application.
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.

Service interoperability

A crucial issue when creating open service environments 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. We believe that the most fruitful way resides 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 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 an 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 service and the context in which the service resides. This metadata could concern in which room the service currently resides, and whom it belongs to.
This approach makes it possible to create both simple services that only have to match API's to be able to use another service, as well as more complex services that uses 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.

Agents in USEs

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 do, problems that could be tackled using different kind of reasoning strategies. Agent based services could also monitor the behaviors 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 logical 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 entity that would provide a shared view of the USE containing dynamically updated representations of users, places and services.

Service discovery

From the discussion above we identify one crucial problem as 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 very frequently in context aware applications, is location, or more precise: *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 typical example of this is 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.

Another property that we se as central is *personal association*. Many of the services in use have an owner. This might be the software on your personal computing artifacts 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 communicating with each other. Other useful properties that one could use in order to create these meaningful collections could concern social or organizational relations, such as project membership etc.

Providing context information in service environments

One way of creating the kind of support described above is to provide the interesting 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 this setting this would include the sharing 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 the system. In this paper we present the Context Shadow system, an infrastructure which provides means to assemble searchable clusters of context information.

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 the services to acquire important context information such as information concerning local artifacts, services or 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. 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 serve 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.

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 chunks of information in the context servers, but is also embedded in the topology of the surrounding web of context servers.

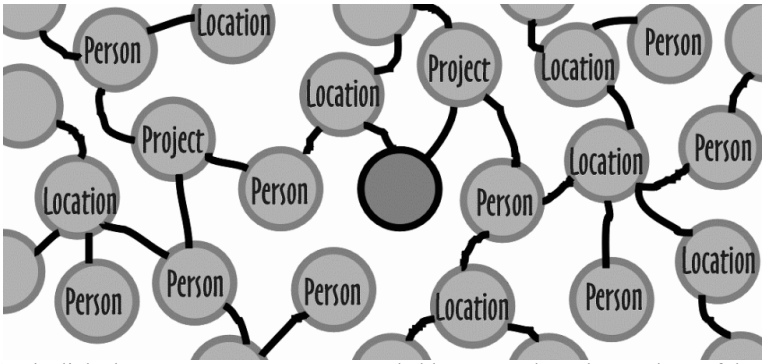

Figure 1. The linked context servers create a searchable space, where the topology of the space is part of the context information.

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 with knowledge about meeting history, documents etc.

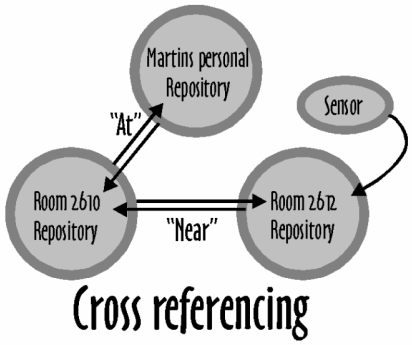

Figure 2. The context servers are linked with references. By following the links it is possible to acquire context information from other context servers than your starting point.

Context sensitive service discovery

The Context Shadow system can be used for different kinds of resource and service discovery tasks. For example, in Fig. 5, 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 a XML description embedded with the speaker service representation, or it can choose to try and match a specific type name describing the service.

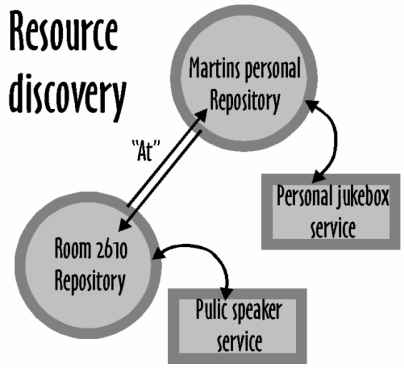

Figure 3. Using Context Shadow, a jukebox service finds a speaker service at the user's current location.

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 context.

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 users 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.

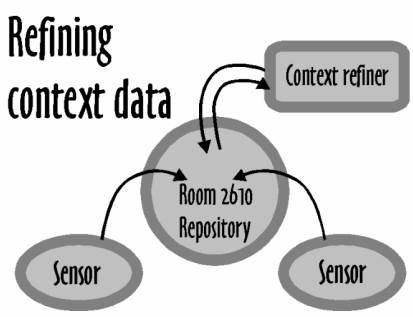

Figure 4. A Context Refiner reads information from a context server, performs some operation or transformation on the data and then posts the result back to the server.

Example applications

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

Messenger service

With this prototype we wanted to examine how public resources can be used to send a message to a person. In the prototype an active messenger service actively tries to find resources near the receiver of the message that can receive the message and display it to the person. The prototype was used in the following scenario:
1. An active messenger service with a message to a person has migrated to that person's laptop. This computer however lacks output resources that the service can use to display the message.
2a. The person enters a room with a public wall display. The messenger service discovers this output resource and chooses to display its message on this resource.
2b. The person enters a room where another person sits with his personal computer. This computer has a public speaker resource. The messenger service discovers the speaker resource and chooses to play an audio version of the message through the speakers.

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

Tools for local collaboration

There tend to be more and more computer artifacts present at meetings, both personal mobile devices such as laptops and PDA's as well as stationary devices such as projectors. These artifacts 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 2002), a small button-like computer memory that the persons actively have to press into a receptor in the room to announce their presence. The button actually contains the URL reference to its owner's Context Server. What happens when the button is placed within the receptor is that the reference is being put into the Context Server for the room, and a cross-reference is automatically posted into the person's personal Context Server.

Active Documents

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

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

Related work

The problem of creating an infrastructure of meaningful assemblies of services was earlier taken on in the Cooltown project by HP labs (Caswell and Debaty 2000), where places are provided with a web page. Services that exist in these places can then be accessed via that web page. The system provides several interesting ways to automatically assemble the services at a location and then make them accessible through the web page. The notion of tying services to a location also exists in Context Shadow, with the difference that in Cooltown the ambition is to make services easily accessible directly by users, while Context Shadow has the ambition to provide services mainly to other services.

Another support system, which targets the problem of how to support the design of context aware applications, is the Context Toolkit system from Georgia Tech (Dey 2001). This is a middleware system with which you can incorporate sensor data in your applications. The system is built with a widget approach making it possible for applications to incorporate context data about the same way as you incorporate a GUI component. This system does not have an explicit infrastructure approach but rather supports the creation of standalone applications. The Context Toolkit provides much of the same functionality as the Context Shadow, such as context queries and refinement of context data. The major difference from the Context Shadow system is that Context

Shadow includes other applications as being part of the context, thus providing support for collaboration between services.
The TEA project presents a system architecture as well as a method to support the design of context aware systems (Schmidt and Laerhoven 2001). The system architecture consists of several layers where the bottom layer consists of cues that represent an abstraction of the sensor-data. The cues are then combined into contexts, which can be seen as a high level description of the current situation. These context descriptions can then be fetched by the applications from a tuplespace. The provided method gives step-by-step support for the choice and assembly of sensors as well as for the application development. The architecture from TEA is similar to the Context Toolkit approach in the sense that they both support the development of standalone applications and that none of them uses shared context servers to represent real world entities.
The Interactive Workspace project at Stanford University (Fox et al. 2000) uses a system they call an event heap to enable services in a room to communicate on an event level. The event heap could be compared with the context servers representing locations in the Context Shadow system, with the difference that the context servers are not used for communication between services to the same extent as the event heap. The system also has no support for combining several event heaps into an infrastructure, nor will the event heap communicate with services that have other properties than being in the room.

Conclusions

In this paper we have presented the Context Shadow system, which allows applications to make queries about users' context. In the system, services and sensors are tied to context servers representing static entities such as persons or locations. These context servers are then connected to each other in a dynamic way that provides the context information with a semantic structure. This novel approach of dealing with context information provides a number of benefits.
Firstly, the system creates shared representations of persons and places that can be used by software services. These abstractions are very useful in highly distributed settings, where services might be distributed over several machines. Choosing an infrastructure solution like Context Shadow will have the effect that not every service itself has to collect and maintain all context information that is available. The system simplifies the creation of context sensitive applications, by taking care of issues such as discovery, transport and refinement of context data. Yet, Context Shadow is *not* a centralized system with all the drawbacks that will bring about. In addition to the advantage of robustness etc., the distributed design of Context Shadow makes it possible for a person to have his or hers context server running locally on a laptop and when moving between different contexts, to establish links from that server to other servers relevant to the person's situation. But, if desirable, it is also possible to have all the persons' context servers running on a single server machine. In both cases, the Shadow Context system facilitates a shared and updated view of the environment. This web-like system structure also results in that the system scales well when new persons, places and sensors are added to the system.
In the implemented prototypes Context Shadow has been explored for different purposes. We have for example described how Context Shadow facilitates sharing of services in different ways. One way is the creation of links between context servers, which make it possible for Context Shadow to support the propagation of queries to other related context servers whereby services for example can find other services on other person's devices in the same room. We have also showed how not only persons can modify the structure of the infrastructure (i.e. by putting one's iButton into a receptor), but also how active services could do it to announce themselves for persons and services inside a room. Services can also contribute to the context information by adding new information that could be based on more low-level information or own observations.

Future work

One future addition to the system will 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.

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 will 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. This context server would disappear after being used. Finally support for events has been asked for, that could for example notify a service when a user changes room.

References

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.

Dallas Semiconductor. 2002. *iButton home page* [On-line]. Available:http://www.ibutton.com/

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

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.

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

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

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,.

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.

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

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.

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., and Van Laerhoven, K.. 2001. How to Build Smart Appliances? *IEEE Personal Communications* 8(4), 66-71.

Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000, Available from World Wide Web: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

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.

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>