Software architecture for the support of context aware applications

Preliminary study by Johan Ensing
February 2002

Delft University of Technology
Prof. Dr. H. Koppelaar
Drs. Dr. L.J.M. Rothkrantz
Prof. Dr. Ir. E.J.H. Kerckhoffs

Philips Research
Ir. M.H. Verberkt

ABSTRACT

Context Awareness is a relatively new research area that uses context information to improve the operation of applications. These applications are called ‘context aware’. However, there is no common understanding what context is, and how context can be used to improve the operation of applications. One of the problems that the developer of a context aware application must deal with, is the problem of obtaining context information. A context architecture should relieve the application developer of this problem: it must obtain context information and present it to applications. The objectives of this study are to define the concepts ‘context‘ and ‘context aware‘, and to gain insight in how context aware applications use context, in order to determine the requirements of a context architecture.

This study has investigated research projects that define context, that develop context aware applications, and that develop context architectures. Based upon this research, this literature study defines context as follows: context is any available (sensor) information that is relevant for an application, excluding the explicit input from and output to the user. Only implicit information is considered to exclude conventional applications from the group of context aware applications. A context aware application is defined as an application that uses context to improve its performance or provide relevant information and/or services to the user. Many research groups have developed some kind of context aware application. These applications usually employ the same types of context: location, identity and time. Context information is used to improve the operation of applications. Context is used to add new functionality to a system, to provide (context dependent) information, to improve the interaction with a device by reducing the interaction, to enhance the interaction by providing new interaction modalities, and to create entire new applications.

Context architectures often take care of all issues concerning distributed computing (e.g. resource discovery). However, these issues will already be solved when the context architecture is part of a larger system (e.g. the WWICE[1] system). The WWICE project has developed a system that, of all investigated architectures, fulfils the most of the requirements that a context architecture should comply with. The most important requirements of a context architecture are:

- Context specification. An application must be able to query the context architecture for context information or to subscribe (with a set of conditions) to context information.
- Common interface. The application does not need to know that (multiple) interpretation steps are necessary to achieve context information. Therefore, all components need to have a common interface such that an application can treat them in a similar fashion.
- Support for context descriptions. A common terminology makes it easier to develop context aware applications, since it can be used to define the interface between applications, the architecture and sensors.
- Context interpretation. Multiple applications often perform a similar interpretation of context information (e.g. transforming an identification number into an email address). Therefore, the context architecture must provide interpretation, such that interpretation only needs to be done once.
- Constant availability of context acquisition. The components that acquire context must execute independently of the applications that use them, such that the context information is always available to different applications.
- Context storage. Context history must be stored such that applications, but also the context architecture itself, can use it to establish trends and predict future context values.
- Model of the environment. A model that is available to both the architecture and applications enables both to perform more sophisticated inferencing. An example of a model is the map of a home.

Since the WWICE project already deals with many requirements of a context architecture, the challenge is not to built a context architecture, but to explore ways to use new types of context. A context type that seems to be a promising research topic is the ‘activity’ of a person (e.g. sleeping, having dinner, and watching television).

[1] Section 4.15 explains the WWICE system in detail.

CONTENTS

1 INTRODUCTION

This chapter defines the subject of this preliminary study. Section 1.1 explains the problem that this study deals with. Section 1.2 explains the relevance of this problem in some more detail. Section 1.3 states the objectives of this study. The scope of this study is explained in section 1.4. Finally, section 1.5 explains the approach to this preliminary study and section 1.6 concludes with the outline of this study.

1.1 PROBLEM DESCRIPTION

Context Awareness is a relatively new research area that uses context information to improve the operation of applications. These applications are called 'context aware'. However, there is no common understanding what context is, and how context can be used to improve the operation of applications. One of the problems that the developer of a context aware application must deal with, is the problem of obtaining context information. A context architecture should relieve the application developer of this problem: it must obtain context information and present it to applications. Philips Research develops context aware applications as part of the Ambient Intelligence research[2]. Therefore, Philips Research wants to gain more insight in how context aware applications use context, and how a context architecture can support the development of context aware applications. Figure 1 illustrates the place of a context architecture in a software system.

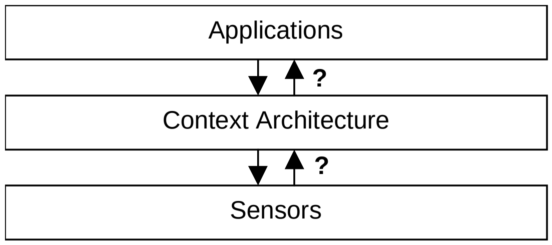

Figure 1 place of a context architecture in a system

1.2 RELEVANCE

With Moore's law still predicting that the number of transistors that can be put on a chip doubles every eighteen months, the embedded computing power in our homes looks set to continue increasing exponentially [1]. But the way in which we experience that computing power is about to change. Greater computing power should be used to provide greater user control and operating convenience, but today it generally manifests itself in products with such a variety of buttons and menu options that many people find them hard to use.

Ambient Intelligence will put an end to the proliferation of buttons and menu options, replacing them with intelligent systems that respond to and even anticipate our needs according to our tone of voice, gestures and expressions [2]. Ambient Intelligence is one of the visions that guide the research at Philips Research. Ambient Intelligence is characterised by ubiquity, transparency and intelligence. Ubiquity because the user is surrounded by a multitude of interconnected embedded systems. Transparency because these systems are invisible and moved into the background of the user's surroundings. Intelligence because the system is able to recognise the inhabitants, adapt itself to them, learn from their behaviour, and even

[2] Section 1.2 explains Ambient Intelligence research in more detail.

shows emotion. Figure 2 shows an example of ambient intelligence: a child interacting with her environment, using only gestures and possibly speech.

Ambient systems need to be aware of the context they are being used in. This research area, Context Awareness, is currently a hot topic and is likely to remain so. There are many recent publications of research groups that created some kind of context aware application. Some examples are the Tour guide [17], the Conference assistant [24], and the Smart Floor [42]. As is mentioned before, Philips Research is also developing context aware applications. Therefore, Philips Research want to gain more insight in how context aware applications use context, in order to determine the requirements of a context architecture. A context architecture is a software architecture that supports the development of context aware applications.

Figure 2 Ambient Intelligence [2]

1.3 OBJECTIVES

The aim of this preliminary study is to gain more insight in how context aware applications use context, and to determine the requirements of a context architecture. More specific, the objectives of this study are:

1. Definition of the concepts ‘context’ and ‘context awareness’.
2. Analysis of context aware applications. What types of context do context aware application use, and to what purpose?
3. Analysis of context architectures. What are the requirements of a context architecture?
4. Based upon the survey of context architectures and the analysis of the requirements of a context architecture, a choice for an architecture that can be used as a basis to develop a context architecture as part of this graduation project.

1.4 SCOPE

This section explains the scope of this literature study.

The interpretation of raw sensor data is not in the scope
An important input channel for context aware applications is the sensor. The range of sensors that can be used and how raw sensor data is to be interpreted, is not in de scope of this study since this depends on the specific application. The context architecture has to deal with the presentation of interpreted sensor information (context) to applications. Buil and Destura [12] have written a thorough report that describes different sensor input technologies.

Environment: in-home network
Philips Research develops context aware applications for the consumer electronics market. The main focus is on context aware applications in an in-home network. Therefore, the context architecture is also intended to work in an in-home network (e.g. WWICE).

Investigation of context aware research: focus on context architectures.
There are many research groups that are engaged in context aware computing. Dealing with all these research projects is not feasible in a three months’ investigation. It is important to get a feel for what researchers are trying to achieve with context aware applications, but the focus of the study has to be on research that develops context architectures.

1.5 APPROACH

The aim of this preliminary study is to gain more insight in how context aware applications use context, and to determine the requirements of a context architecture. First, I have investigated what research groups mean with ‘context’ and ‘context aware’. Next, I have investigated what research groups are trying to achieve with context aware applications, and what types of context are used for the development of context aware applications. Subsequently, I have investigated the context architectures that have already been developed by other research groups. I have determined the requirements of a context architecture, based upon the properties of these context architectures. Next, I have chosen which context architecture is suitable to be used as a basis in the implementation part of the graduation project. This choice is based upon which architecture complies with the most requirements, and the availability of the (implementation) details of this architecture.

1.6 OUTLINE

Chapter 2 gives an introduction to context aware computing. This chapter explains how different research groups define the concepts context, context aware, and context architecture. This chapter concludes with the definitions are used in this study. Chapter 3 explains what researchers are trying to achieve with context aware applications. This chapter explains what types of context are used, and the benefits of using context in applications. Chapter 4 is the most important chapter of this study: it gives an overview of the current research concerning context architectures both in- as outside Philips Research. Furthermore, it presents the requirements of a context architecture. The last section of this chapter compares the different architectures and presents the architecture that is most suitable to use as a basis for further research. Chapter 5 summarises the results of this study. Finally, chapter 6 presents the conclusions of this study.

2 INTRODUCTION TO CONTEXT AWARE COMPUTING

Many research groups are investigating context-aware applications. However, there is no common understanding what ‘context’ and ‘context awareness’ means. The purpose of this chapter is to define the concepts ‘context’, ‘context aware’, and ‘context architecture’. Section 2.1 describes what context is about and ends with the definition of context that is used in this . Section 2.2 deals with the different types of context aware applications and ends with a definition of a context aware application. Section 2.3 explains what a context architecture is and states the definition of a context architecture that is used in this .

2.1 CONTEXT

The last decade, sensors have become cheaper, smaller, more diverse and easier to use. Therefore, more and more research groups are using sensors to improve the operation of applications. As stated in the 1997 ten-year Forecast of the Institute of the Future [31]: “One can build an accelerometer on a single chip for a couple of dollars, creating a device that is not only cheaper than today’s sensors, but also smarter and more reliable”. Figure 3 shows that sensors are seen as the enabling technology of the next decade, according to the Institute of the Future.

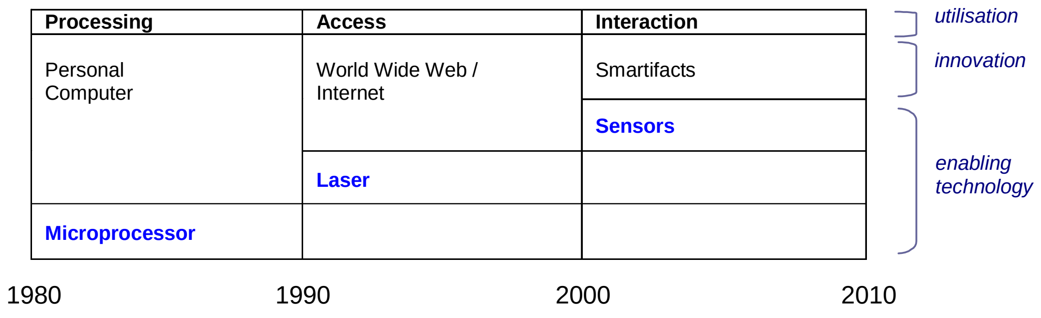

Figure 3 Sensors: the Next wave of Infotech Innovation [31]

Schilit and Theimer invented the concept of “context” in their first work on context awareness [48]. This work defines context as location, identities of nearby people and objects, and changes to those objects. Their work has inspired many research groups to work on ‘Context Awareness’. Most of these research groups have come up with their own definition of context and context awareness. Some of the definitions of context are stated below to give an impression how research groups are trying to handle this vague notion. According to Mari Korkea-aho, context means situational information [32]. Context information is almost any information available at the time of an interaction, for example:

- *identity*
- *spatial information - e.g. location, orientation, speed, and acceleration*
- *temporal information - e.g. time of the day, date, and season of the year*
- *environmental information - e.g. temperature, air quality, and light or noise level*
- *social situation - e.g. who you are with, and people that are nearby*
- *resources that are nearby - e.g. accessible devices, and hosts*
- *availability of resources - e.g. battery, display, network, and bandwidth*
- *physiological measurements - e.g. blood pressure, hart rate, respiration rate, muscle activity, and tone of voice*
- *activity - e.g. talking, reading, walking, and running*
- *schedules and agendas*

According to Guanling Chen and David Kotz, context is the set of environmental states and settings that either determines an application's behaviour (active context) or in which an application event occurs and is interesting to the user (passive context) [14]. They define four categories of context:

- Computing context (e.g. communication bandwidth)
- User context (e.g. profile)
- Physical context (e.g. noise level)
- Time context (e.g. time of the day)
- Context history

Albrecht Schmidt, Michael Beigl, Hans-W.Gellersen have a different approach [51]. They have defined a working model for context that is shown in Figure 4. In this working model, a context describes a situation and the environment a device or user is in. A context is identified by a unique name. For each context, a set of features is relevant. For each relevant feature, a range of values is determined (implicit or explicit) by the context

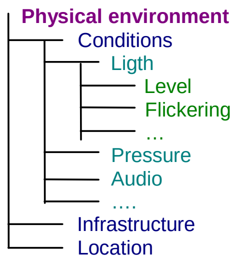

Figure 4 Context Feature space

Henry Lieberman and Ted Selker have another approach [35]. They define context as an implicit input and output to an application, as shown in Figure 5. This means that context is everything that affects the computation except the explicit input and output.

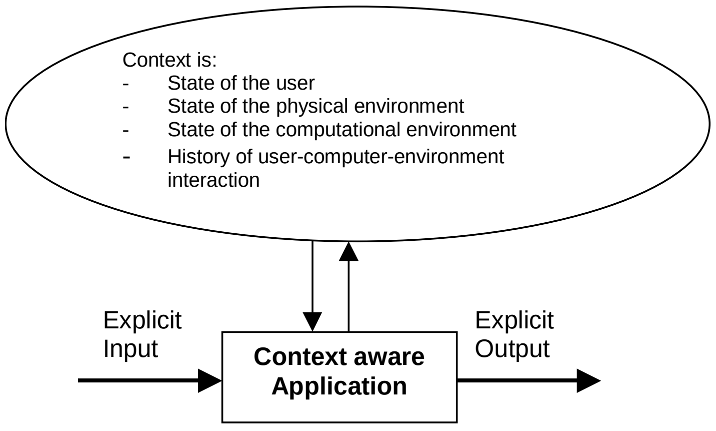
Context is:
- State of the user
- State of the physical environment
- State of the computational environment
- History of user-computer-environment interaction
Explicit Input
Context aware Application
Explicit Output

Figure 5 Context in human – computer interaction [35]

Schilit et al. and states that important aspects of context are where you are, who you are with, and what resources are nearby [47].

Brown *et al.* defines context as location, identities of the people around the user, the time of day, season, temperature, etc. [10].

Ryan *et al.* defines context as user's location, environment, identity and time [44].

Dey [19] states that context is the user's emotional state, focus of attention, location and orientation, date and time, objects, and people in the user's environment.

A.K. Dey and G.D. Abowd define context as any information that can be used to characterise the situation of an entity [22]. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. They define the most important types of context to be *location, identity, activity and time.* These are the *primary* context types. These can be used as indices to find *secondary* context (e.g. the email address) for that same entity as well as primary context for other related entities (e.g. other people in the same location).

§ ***Conclusion***

Many research groups use sensors to improve existing or invent new applications. However, sensor information is not the only information that is used to improve the operation of applications. Therefore these research groups use the concept ‘context’ to indicate the information that is used to improve the operation of their applications. However, there is no common definition for ‘context’. The definition that is used in this is based upon the definition of A.K. Dey and G.D. Abowd [22]:

“ Context is any available (sensor)information that is relevant for an application, excluding the explicit input from and output to the user”

Thus context is all information that an application can use, excluding information that is explicitly provided by the user of the application and the direct output to the user. Only implicit information is considered to exclude conventional applications from the group of context aware applications.

2.2 CONTEXT AWARE

This section explains how different research groups define ‘context aware‘, and what properties a context aware application has according to these research groups. This information is used to determine the definition for a context aware application that is used in this .

Chen and Kotz define Context aware computing as a mobile paradigm in which applications can discover and take advantage of contextual information [14]. Two definitions:

- Active context awareness. An application automatically adapts to discovered context, by changing the applications behaviour.
- Passive context awareness. An application presents the new or updated context to an interested user or makes the context persistent for the user to retrieve later.

Brown *et al.* has identified the following features of context aware applications [8]:

- Ability to capture the current context.
- Ability to represent the current context. In other words, synthesise high-level events from low-level ones.
- Context memory. In other words, the ability to act on context changes and to analyse context history.

Brown defines the following utilisation for context aware applications:

- Proactive triggering. Applications that present relevant information or perform relevant actions based on (context) information.
- Streamlining interaction. Applications that simplify the user interface based on (context) information, e.g. applications that use speech instead of keyboard input when a user is too busy to use a keyboard.
- Memory for past events. Applications that help to remember things based on stored (context) information.
- Reminders for future contexts. Applications that remind you to do something based on some requirements that are fulfilled. For example a reminder that you have to buy milk, when you are near a grocery store.
- Optimising patterns of behaviour. Applications that provide relevant information to simplify a task.
- Sharing experiences. Applications that make it possible to share experiences with other people based upon specific (context) information

Schmidt *et al.* has identified four reasons for applying context [51]:

- To make adaptive applications.
- Improvement of human interaction with devices.
- To give additional meaning to information supplied by user.
- To filter information.

Cheverst *et al.* identifies three features of context aware applications [15]:
- Simplifying/reducing the task specification required from the user. In other words, reducing the input/action of the user.
- Changing the output produced by the system. In other words, reducing the quantity or increasing the quality of information that is presented to the user.
- Reducing the complexity of rules constituting the user's mental model of the system. In other words, a context aware application does some computing on the users' behalf.

Schilit and Theimer were the first to introduce the term 'context-aware' [48]. They define the following types of context aware applications:
- *Proximate selection applications.* These applications make items that are applicable to the users context, easier to choose.
- *Automatic contextual reconfiguration*. These applications automatically retrieve information, based upon context.
- *Contextual command applications*. These applications execute commands *manually* based on context. This means that a user executes a service, but the context makes services available.
- *Context-triggered applications.* These applications execute commands *automatically,* based upon context.

Pascoe defines these features of context aware applications [43]:
- *Contextual sensing.* The ability to detect contextual information and present it to the user.
- *Contextual adaptation*. The ability to execute or modify a service automatically, based upon context.
- *Contextual resource discovery.* This allows context aware applications to locate and exploit resources and services that are relevant to the user's context.

Dey and Abowd define these features of context aware applications [22]:
- *Presentation* of information and services to a user.
- Automatic *execution* of a service.
- *Tagging* of context to information for later retrieval.

Dey and Abowd define a system to be context aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

§ ***Conclusion***

The common idea is that context-aware computing should make interacting with computers easier. A context aware application often uses, among other things, sensors to obtain context information. Context aware applications adapt their behaviour based upon context information. This could mean, for example, that the type of user interface is changed (e.g. speech input instead of keyboard input), the user interface itself is changed (e.g. adjustment of the font size), or the application changes its execution (e.g. less user input required or automatic execution of a program). Some applications do not only improve their performance, but in fact the whole application is about delivering context information or services. Chapter 3 deals with context aware applications in more detail. This uses the following definition for a context aware application, which is mainly based on the definition of Dey and Abowd [22]:

"A system is context aware if it uses context to improve its performance or provide relevant information and/or services, where relevancy depends on the user's task"

Section 2.1 has defined context as any available (sensor) information that is relevant for an application, excluding the explicit input from and output to the user. The definitions of 'context' and 'context aware' deliberately leave open what context information an application can use, how this information can be acquired, and what an application can do with it. This is exactly what Context Awareness research is trying to figure out.

2.3 CONTEXT ARCHITECTURE

Many research groups that develop context aware applications do this to obtain one particular application. There are hardly any general frameworks that can be re-used when developing context aware applications. A general framework for context aware computing would make it a lot easier to develop new or evolve existing context aware applications. A good attempt to build such a general framework is the Context Toolkit [20]. Section 4.1 explains this context architecture in detail. This section states a definition for a 'context architecture'.

A context architecture is a software architecture that deals with context information. Boasson defines a software architecture as 'a system structure that consists of active software modules, a mechanism to allow interaction among these modules, and a set of rules that govern the interaction [7]'. Figure 6 gives an impression of a model for a context architecture.

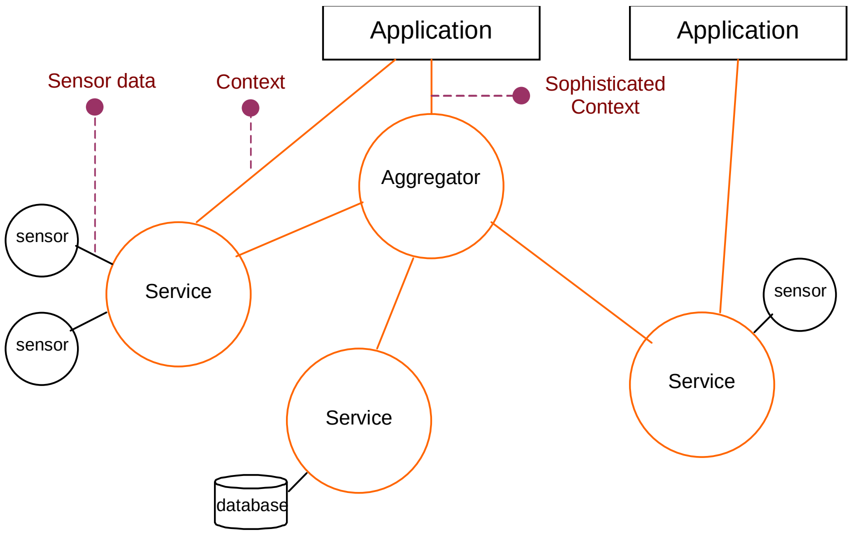

Figure 6 model of a context architecture

The model that is shown Figure 6 must be interpreted as follows. Sensors sense raw sensor data. This data has to be interpreted to be able to use it. The interpreted data is called 'context' and is presented for external use on the network by a so-called 'Service'. For example, a sensor might sense fingerprints and a service might interpret the fingerprint into an identification of a person. Different context information can be aggregated to form more sophisticated context information. For example, one might combine the identity of a person with the weight and blood pressure of a person, to calculate a 'fitness'-value. An application can use (sophisticated) context information to improve its performance and to go into certain actions. Figure 6 shows that context information is not necessarily (interpreted) sensor data. For example, a service might also provide context information that it retrieves from a database.
A context architecture should consist of 'building blocks' that support the development of a context aware application. This architecture should make it easy to combine context information and to present context information to applications. This uses the following definition for a context aware application.

" A context architecture is a software architecture that supports the development of context aware applications "

The subject of this study is how a context architecture can support the development of context aware applications. In other words this study determines the requirements of a context architecture.

3 CONTEXT AWARE APPLICATIONS

This chapter gives an overview of what researchers are trying to achieve with context aware applications. The applications are divided into categories that identify the goal of the application and that highlight a specific property of a context aware application. Every section explains one category and describes one example of a context aware application. The applications that are described in this chapter are a few out of many, but they give a good overview of what researchers are trying to achieve with context aware applications.

3.1 OFFICE AND MEETING TOOLS

Office and Meeting tools are an important research topic in the field of context aware computing. The goal of these tools is to improve office tasks, like forwarding phone calls. A good example is the Active Badge System from the Olivetti Research Lab [54]. This is a system for the location of people in an office environment. This system is often used by other research projects to provide location information.

Members of the staff wear badges that transmit Infra Red (IR) signals providing information about their location to a centralised location service, through a network of sensors. The system is used to locate persons in an office and forward calls to the closest phone. A network of sensors placed around the office building picks up the signals and a central location server polls these sensors. The system also includes commands to obtain the current location of a badge, to find out which other badges are in immediate proximity to a particular badge, to find out which badges are currently near a specified location, to notify when a badge is again traceable, and to obtain information that indicates where the badge has been during an one-hour period.

The type of context that is used in office and meeting tools is mostly location, identity and location-history. This information is used to add new functionality to a system. For example, using the Active Badge system it is possible to see where people are and to forward phone calls to them. Other office and Meeting tools are the Cyberdesk system that is developed at the Georgia Institute of Technology [23], [19], the Office assistant of MIT Media Laboratory [57], and the ParcTab system that is developed at the Xerox Palo Alto Research Centre [49]' [55]. The ParcTab system will be explained in more detail in the chapter four.

3.2 (TOURIST) GUIDES

The goal of (tourist) guides is to provide visitors with location-based information. An example of a tourist guide project is the GUIDE[3] project of the University of Lancaster [17]. This project develops a context sensitive tourist guide for the visitors to the city of Lancaster.

Users have a portable PC that communicates with a Wireless LAN to retrieve information. The used context information is knowledge of the user and knowledge of the environment (including the physical location of the user). Based on the location and user preferences, visitors can obtain information in two ways. All the required information is broadcasted by cell base-stations to portables either as part of a regular schedule (information relevant for the region the station covers) or in response to user requests. The system determines the location of the user by the cell coverage, or the user can specify in which location she is. This approach is the opposite to the approach of the Cyberguide project of the Georgia Institute of Technology. That project has stand-alone portable end-systems that have all the information

[3] http://www.guide.lancs.ac.uk/

they require preinstalled [3]. The Cyberguide receives a locationID that is transmitted by beacons and uses this ID to find locally stored information concerning this location.

The GUIDE project fulfils the following functionality:
- Different visitors have different demands. It must be possible to tailor information to the needs of the visitor: level of information (academic, scholar), duration of tour, class information (history, architecture), etc.
- Information changes continuously, such as the availability of tours, weather forecasts, etc. This information needs to be updated dynamically.
- Users have specific questions or want to use specific services, e.g. booking of accommodation. Therefore, there is a need for a communication link to the portable end-systems.

The context information that is used by most tourist guides is the location of the user and possibly user preferences (that have to be supplied by the user herself). This information is used to present the user with location specific information. Another tourist guide application is the Smart Sight Tourist Assistant that is developed by Carnegie Mellon University [58]. The Virtual Information Towers architecture of the University of Stuttgart also supports tourist guide-applications [34].

3.3 FIELDWORK TOOLS

The goal of a fieldwork tool is to ease the work of an ecologist by decreasing the amount of information that the ecologist has to record. Some information, such as the location and time, is automatically added to observations. The University of Kent has developed a context aware fieldwork tool [44]. This tool has been constructed to explore how context awareness can be used to aid an ecologist's observations of giraffe in a Kenyan game reserve.

The application uses context-awareness to influence how data is recorded. The PalmPilot is used as terminal since this is a small, lightweight, robust, device. GPS is used for positioning. The context aware recording device assists the user in making observations because it can `observe' some states of the user's surroundings: the location and the time. The application is based on the *stick-e note* model that is explained in more detail in section 4.9. A user can make notes, and the application automatically adds information. The users can define what information the notes should include, e.g. location, time, date, etc. The application enables users to view the location of stored notes on a map. This is useful to observe patterns in the recorded data. Furthermore, the user can locate and orient herself. The tool has been tested in Kenya in 1997 to 1998. The researchers using the tools have found them valuable. The research group at the University of Kent has recognised the need for a context architecture to provide a common access medium to the current context for any program, service, or user. Therefore, they have developed the Contextual Information Service (CIS) that is explained in section 4.2.

This fieldwork tool uses the context types ‘location‘ and ‘time‘. There are no other research groups that develop fieldwork tools, but since it is an interesting research area it is included in this chapter. Fieldwork tools improve the interaction with a device by reducing the interaction: some information is sensed automatically and does not need to be entered anymore.

3.4 REMEMBRANCE TOOLS

The goal of Remembrance tools is to help people remember things, e.g. to remember things from the past (did I turn off the stove?), or to remember things in the future (a shopping assistant that reminds you to buy coffee when you are near a supermarket). An interesting remembrance tool is the MemoClip of the University of Karlsruhe [5]. This application reminds its’ user of tasks, based upon her location.

The MemoClip is a computer with some sensors, communication capabilities and a LCD display. At the places of interest, location beacons are installed. The MemoClip constantly requests location information. When a beacon receives these requests, it sends location information to the MemoClip. The user can associate task information (a text message) with place descriptions. This task information is stored on the MemoClip. When the clip receives location information from a beacon that corresponds to the location information that is associated with a task, the clip makes a 'beep' sound and displays the text message describing the task. An important issue in this project is presenting the user with an understandable location model. The model that is developed describes location with a semantic description and possibly with relationships between places, for example: "my home" *contains* "my living room".

The MemoClip uses only location information. Other remembrance tools also use other context, such as the event 'starting an application'. For example, when you text editor starts, you might get a message that you still have to write a letter to your grandparents. Remembrance tools are *new* applications dedicated to use context information to help people remember things. Other remembrance tools are the remembrance agent of MIT Media Laboratory [53], the Cybreminder of the Georgia Institute of Technology [21], the shopping assistant of A T&T Bell Laboratories [4], and the Forget-me-not tool from Xerox research centre [33].

3.5 INTERFACE TOOLS

A group of context aware applications that is less visible to the user of a device is the group of interface tools. These context aware applications automatically adjust the interface of a device according to the observations of their sensors. A good example of this category is the research from Microsoft Research that investigates ways to enhance the interaction of a user with a PDA [29], as shown in Figure 7.

This research integrates a set of sensors into a handheld device to demonstrate some new functionality:

- Recording memos when the device is held like a cell phone. This is sensed by a combination of the proximity range sensor and the tilt sensor.
- Switching between portrait and landscape display modes by holding the device in the desired position. This is sensed by a tilt sensor
- Powering up when the user picks up the device. This is sensed by a touch sensor.
- Scrolling the display using tilt. This is sensed by a tilt sensor.

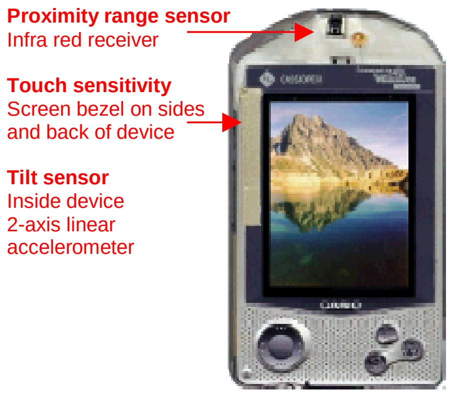
Proximity range sensor
Infra red receiver
Touch sensitivity
Screen bezel on sides and back of device
Tilt sensor
Inside device
2-axis linear accelerometer

Figure 7 Enhanced PDA [29]

Interface tools are context aware applications that enhance the interaction between a user and an application/device by providing new interaction modalities. Other projects that use sensors to develop new interaction concepts are the TEA sensor board of the University of Karlsruhe [50] and the Nomadic Radio project of MIT Media Lab [45].

3.6 CONCLUSIONS

This chapter explains five categories of context aware applications. The type of context information that is mainly used is location. Other important context types are time and identity. Research groups use context information to improve the operation of applications. There are many ways to do this: office and meeting tools use context to add new functionality to a system. Tourist guides use context information to provide (location specific) information. Fieldwork tools improve the interaction with a device by reducing the interaction: some information is sensed automatically and does not need to be entered anymore. Remembrance tools do not improve an existing application: they are *new* applications dedicated to use context information to help people to remember things. Finally, interface tools enhance the interaction between a user and an application by providing new interaction modalities.

4 CONTEXT ARCHITECTURES

This chapter gives an overview of the research concerning software architectures that support the development of context aware applications. The goal of this chapter is to acquire a list of requirements that a context architecture should comply with. Section 4.1 to section 4.11 describe context architectures that are developed outside Philips Research, and section 4.12 to section 4.15 deal with research projects inside Philips Research. Of course, there are more architectures that support the development of context aware applications. Examples are Cooltown[4] (every object has an URL), CALAIS (mainly focuses on presenting location information) [38], the TEA board (uses sensors to enhance interaction with a mobile phone) [50], EKTARA (architecture for wearable computing) [18] and the Human Centred Interaction Architecture (focuses on different input modalities that present their context information on a blackboard) [56]. However, describing more architectures does not add any new requirements to the list of requirements that is presented in section 4.16. Section 4.17 compares the described architectures and shows which architecture is the most suitable to use as a basis for future research.

4.1 CONTEXT TOOLKIT

The Context Toolkit[5] is a project of the Georgia Institute of Technology [20], [25]. The context toolkit is a context architecture that supports the development of context aware applications. An example of an application that is developed using the Context Toolkit is the Intercom application that keeps track of the locations of people and enables people to send messages to other people by just saying “House, I want to speak to…”.

4.1.1 Architecture

The architecture uses an object-oriented approach: it consists of three types of objects: context widgets, context interpreters and context servers.

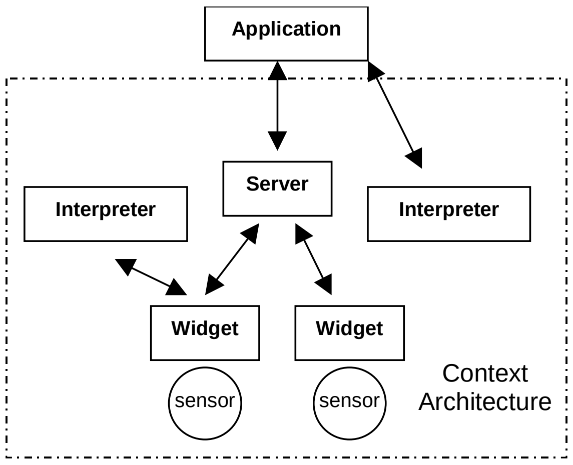

Figure 8 Architecture Context Toolkit [19]

[4] For more information, see http://cooltown.hp.com/

[5] The Context Toolkit is freely available at http://www.cc.gatech.edu/fce/contexttoolkit/

Context widgets
Context widgets provide components (applications, context servers) with context information that is obtained via, for example, sensors. Widgets are reusable and customisable building blocks for context sensing and abstract the details of sensing or acquiring context from using it. A widget has attributes and callbacks. Attributes are the pieces of context that are available to other components. Callbacks are the types of events that the widget can use to notify components that have subscribed with the widget. Components can query a widget for its attributes and callbacks. Components can use polling and notification mechanisms to retrieve current context information. Furthermore, historical context information can be retrieved.

Context interpreters
Context interpreters transform between different types of context such that it becomes useful for an application. For example, it context interpreter can look up the name of a person when her identification number is sensed by a sensor, or it can transform {x,y,z}-coordinates into application specific coordinates (e.g. “living room”).

Context servers
Context servers aggregate related context values to obtain ‘high-level’ context information, e.g. combining blood pressure, weight and temperature to calculate the fitness of a person. Context servers can also be used to collect the entire context about a particular entity (e.g. a person). The context server is responsible for subscribing to every widget of interest and acts as a proxy to the application. Just like a widget, it has attributes and callbacks, it can be subscribed to and polled, and its history can be retrieved.

The context architecture also contains a Discoverer-component that acts as a registry. All components automatically register themselves and their capabilities with the Discoverer. Applications can query and subscribe to the Discoverer for components that interest them. Once the components have been found, the application can create a series of subscriptions and queries to individual widgets. Furthermore, an application can request interpretation from individual interpreters. Components (widgets, interpreters and servers) are instantiated independently of each other and can run on different devices. All components and applications use BaseObject instances to communicate with each other. BaseObject is a (Java) class that takes care of distributed communication. Using BaseObject, none of the individual components need to know how the communication with other components is being implemented. The Java implementation of the Toolkit uses HTTP for communication and XML as message format. However, one can easily change these mechanisms.
When the hosts that runs the component crashes, widgets and aggregators have a subscription log that allows them to re-establish communications with the components and applications that have subscribed to them. Context widgets and servers automatically store the context they acquire (if desired).

4.1.2 Requirements

Dey defines seven requirements that are the basis for his context architecture [20]. This sub-section deals with these requirements and with some additional requirements that are identified during the development of the toolkit.

1. Context specification (subscription / polling mechanism)
There must be a mechanism that allows an application to indicate in what context information it is interested and when. An application must be able to query the context architecture for context information or to subscribe (with a set of conditions) to certain context information.

2. Separation of concerns and context handling (common interface)
Context information that is used by applications might come ‘directly’ from sensors (after interpretation) or there might be several layers of abstraction. For example, to calculate whether or not a meeting is going on in a specific room, there might be a need to calculate how many people are in that room, whether or not a meeting is scheduled and whether or not the people are talking to each other. From an application designer’s perspective, the use of these multiple layers and the use of sensors should be transparent: the application designer

does not need to know that (multiple) interpretation steps are necessary to achieve this context information. Therefore, all components (aggregators, interpreters and widgets) need to have a common external interface such that an application can treat them in a similar fashion.

3. Constant availability of context acquisition
It must be possible that multiple applications request the same or a different set of context. Therefore, components that acquire context must execute independently of the applications that use them, so the context information is always available to different applications. An additional advantage is that the application designer needs not to instantiate, maintain or keep track of components that acquire context.

4. Context interpretation
Multiple applications often perform a similar interpretation of context information (e.g. transforming an identification number into an email address). Therefore, the framework must provide interpretation. Furthermore, it must be possible to use different techniques to interpret context information. Examples of interpretation techniques that might need to be supported are sensor fusion techniques, operators such as OR, XOR and NOT, learning techniques such as neural networks, classification techniques such as Hidden Markov Modelling, and the follow-by operator that lets a programmer specify that one event must follow another event in time.

5. Transparent distributed communications
The fact that communication is distributed should be transparent to both sensors and applications. This relieves the designer of having to build a communications framework.

6. Context storage
Context history must be stored such that applications, but also the context architecture itself, can use it to establish trends and predict future context values.

7. Resource discovery
There should be a resource mechanism (e.g. a registry) that is responsible for finding any applicable components (keeping a list of available components) and for providing the application with ways to access them.

4.1.3 Additional requirements

During the development of the context toolkit, some more requirements of a context architecture are identified. These are not implemented (yet). These requirements are:

a. Controlling access to context
Context information might be private information. Therefore, the access to context must be controlled.

b. Support for Context Descriptions
It is difficult to describe context. There is a need for a structure to develop a context terminology. Issues that need to be defined are the values that context can have, the services that the architecture delivers (e.g. is it possible to request the names of all persons in room X), etc. A common terminology makes it easier to develop context aware applications, since it can be used to define the interface between applications, the architecture and sensors.

c. Prototyping Environment
It must be possible to test applications without using real sensors. Software sensors in the form of a Graphical User Interface (GUI) can collect input explicitly from the user. When these GUIs are treated as sensors, an application can easily be prototyped and tested.

d. Model of the Environment
A model of the environment that is available to both the architecture and applications would enable both to perform more sophisticated inferencing. An example of a model is the map of a home. If this knowledge is placed in the architecture, all context-aware applications could query it for information that is relevant to the application's particular domain.

e. Quality of Service
Context components should indicate their baseline quality of service values for the context they can provide to applications. They must notify subscribers of any changes to these values. Examples of Quality of Service metrics are ambiguity, reliability, coverage, resolution, frequency and timeliness. *Ambiguity* deals with ambiguous context information. *Reliability* deals with sensor failures and uncertainty of sensor information due to 'ageing' (after some time, information is not up-to-date anymore). *Coverage* defines the set of all possible values for a context attribute; *resolution* defines the actual change that is required for the context attribute to change. Frequency and timeliness determine the "real-time" requirements of the application. *Frequency* defines how often the information needs to be updated and *Timeliness* defines the period of time that an application allows between the actual context change and the related notification to the application.

f. Support for multiple platforms
Since the context architecture is meant for distributed computing, the architecture must work on different platforms (Windows, Linux, etc.). The Context Toolkit supports this feature since it is developed using Java.

g. Support for alternative implementations of architecture features
Another useful feature of the toolkit is the possibility to replace the default implementation of features (e.g. communications schemes) at instantiation time. The instantiated component is able to automatically replace the default implementation and use the alternative implementation. This is useful when you want to incorporate a context architecture into an existing project.

4.2 CONTEXTUAL INFORMATION SERVICE (CIS)

The Contextual Information Service (CIS) is developed at the University of Kent [43]. The goal of the CIS is to maintain a model of contextual information that makes it easier to obtain contextual data. An example of an application that is built using the CIS is a wearable field assistant designed to aid an ecologist's observations of giraffe. This application automatically senses the ecologist's location so that the ecologist does not need to enter this information anymore. Unfortunately, there is little information available concerning the CIS and it is not possible to download this architecture.

4.2.1 Architecture

The CIS model consists of a world of artefacts. Artefacts are objects that have a name, type, and set of contextual states, e.g. a 'person'-artefact can have a 'location' state. One can add a new artefact by selecting a template from the artefact catalogue. The CIS contains a state dialog that lists the various generic states that can be used to construct artefacts. Each state has one or more formats and types, methods to translate between them, and a set of operations. CIS clients (applications) can access any state of any artefact. Figure 9 illustrates the CIS model.

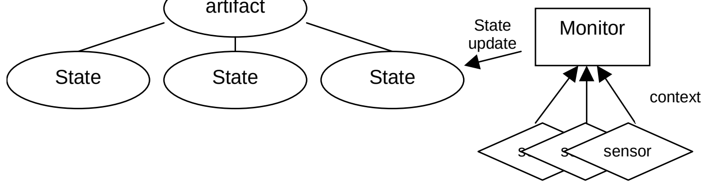

Figure 9 CIS model

A so-called ‘monitor’ is associated with a single artefact state and searches for ‘sensors’ and ‘synthesisers’ that can provide the information that it needs, based upon a Quality of Service directive (e.g. location must be accurate to 30 meters). ‘Sensors’ are programs that extract contextual information from connected or embedded devices and ‘synthesisers’ are programs that extract information from the state of other artefacts. The monitor decides when to obtain / generate values and perform simulations.

4.2.2 Requirements

CIS implements some of the requirements that are already mentioned in the Context Toolkit section: ‘Support for Context Descriptions’ is partly taken care of by using catalogues and the monitors use ‘Quality of Service’ directives to choose the sensors that they need to obtain context information. Furthermore, the CIS has a ‘Model of the Environment’. The CIS architecture does not introduce new requirements.

4.3 EASYLIVING

Easyliving is a project of Microsoft Research [37], [11]. Its’ goal is the development of an architecture that allows for dynamic aggregation of diverse I/O devices, within one single room. This research focuses on middle ware, geometric modelling, perception and service description. One of the proposed applications is a light-controller that provides light for the user as she moves around at night. Unfortunately, it is not possible to download (parts of) the Easyliving project.

4.3.1 Architecture

The most important aspects of the Easyliving project are the middleware, geometric modelling, perception, and service description.

Middleware
The middleware part facilitates distributed computing. Instead of using Corba, RMI or another existing middleware component, the EasyLiving project has developed InConcert. This extends the existing middleware solutions with a naming and lookup service and asynchronous message passing.

Geometric modelling
This part models geometric knowledge, i.e. information about the physical relationships between people, devices, places and things. A model is developed that can be used to model the relations (e.g. ‘close to’, ‘in line of sight’) of objects in one single room. The model can use multiple perception technologies and abstracts the application away from any particular sensor modality. Measurements can have an uncertainty associated with them. How the information for this model can be obtained, is still a research topic.

Perception
A stereo computer vision system is developed to distinguish people in a room. Up to three people are distinguished quite well.

Service description
There is a separation between hardware device control, internal logic, and user interface presentation. Abstract service descriptions allow each service to expose a set of attributes or commands so that other services may interact with it automatically. A general communication mechanism is developed using XML-messages.

4.3.2 Requirements

This architecture does not provide any new requirements with respect to a context architecture. However, it extends the 'Model of the Environment' requirement.

Model of the Environment - extension
A location models must fulfil the following requirements:

- It must contain a static model (issues are granularity, which objects to represent, which relationships to represent, etc.)
- It must be scalable. The geometric representation should be dynamically updated when people, devices or locations are added or deleted from the model, e.g. it should be easy to add a new location to the map of a house.
- The model must deal with contradictionary facts. Often multiple sensors are combined to obtain information. This means the system must be able to deal wit contradictionary input.

4.4 HIVE

Hive[6] is a platform for distributed, decentralised software agents, developed by the MIT Media Lab [39]. The Hive architecture enables application developers to easily build applications and reconfigure systems. Hive is not intended for the support of context aware applications, but it can be used for this purpose. An example of a Hive application is a table that senses poker chips lying on the table. These poker chips correspond to a particular MP3-fie. The nearby computer plays the corresponding MP3 when a poker chip is tossed on the table.

4.4.1 Architecture

The Hive architecture consists of three components: cells, shadows and agents. A Hive cell is a program that runs on a specific computer with a published network address that represents a device. Each cell has a "server list agent" that contacts a registry to maintain membership in a Hive network. Each cell has local resources, called shadows, such as a sensor, a display or a digital camera. A shadow can be seen as an API to the resource and takes care of the security and the resource control policy. Each cell can host multiple agents that use the shadows. An application consists of the communications and actions of agents. Agents communicate using Java Remote Method Invocation (RMI) and can only access each other's remote interfaces. The most important aspects of agents are:

- Agents are autonomous
- Agents are proactive: they encapsulate computational activity
- Agents are self-describing: an ontology of agent capabilities can be used to describe and discover available services. The ontology is based on the Java type system (syntax) and the Resource Description Framework (semantics), which is a W3C standard.
- Agents can be mobile: agents can move around the network, although in practice this aspect is not used very often.

4.4.2 Requirements

This architecture provides the following new requirements:

1. Controlling access to context
Context information might be private information thus the access to context must be controlled.

2. Resource management
When multiple context aware applications use the same resources (e.g. displays or speakers), resource conflicts might occur. There has to be a resource management mechanism that deals with these conflicts.

[6] Hive is freely available at http://www.hivecell.net/

4.5 METAGLUE

MetaGlue is agent-based system that is developed at the MIT Artificial Intelligence lab [16]. The Metaglue architecture is an extension to the Java programming language and is used to build software agent systems for controlling Intelligent Environments. Intelligent Environments are highly dynamic, distributed, computational systems. An example of an application in the intelligent environment is the MeetingManager, a multi-user, multi-modal collaboration tool for planning, facilitating, and browsing structured meetings. Unfortunately, it is not possible to download the Metaglue system.

4.5.1 Architecture

MetaGlue introduces an agent-class. Software developers that want to develop their own agent can use this Java class. Agents are used to represent local resources (e.g. sensors) and to interact with those resources. Agents run on a MetaGlue Virtual Machine, which means that they run autonomously from individual applications so they are always available to service multiple applications.

Metaglue has the following capabilities:

1. *Configuration management.* Metaglue manages information about agent's modifiable parameters, stores the agent's state, and gives database access.
2. *Establish and maintain the configuration that each agent specifies.* Agents can specify requirements that the system must fulfil for the agents to run, e.g. necessary hardware.
3. *Establish communication channels between agents.* Metaglue establishes communication paths between agents, so an agent creator does not need to worry about communications. In order to find agents, Metaglue uses an internal Catalogue that stores the location and capabilities of agents. When an agent is not running but is needed by an application or agent, the application or agent can automatically start the agent.
4. *Maintain agent state*. The state of an agent can be stored and retrieved from Metaglue's internal SQL database.
5. *Introduce and modify agents in a running system.* Metaglue handles what to do when agents (temporarily) stop. Metaglue tries to restart the agent while meeting the agent's required configuration, or it might temporarily switch to other agents.
6. *Manage shared resources.* Metaglue uses a resource manager to know what resources exist and are available. A mechanism is developed to allocate and de-allocate resources.
7. *Event broadcasting*. An agent or groups of agents can register to be notified of certain events, such as somebody entering a room.
8. *Support for debugging.* Metaglue has a graphical interface called the catalogue monitor to examine a running system of agents.

4.5.2 Requirements

The following new requirements can be extracted from the Metaglue-project.

1. Recovery support

When a part of the system goes down, it must be possible to automatically restart any unfinished jobs when the system works again. Furthermore, the loss of a part of the system should have a minimal impact on the rest of the system: the system should be robust.

2. Debugging

A useful feature of a system is support for debugging. A powerful way to support debugging is the graphical presentation of state information.

4.6 OPEN AGENT ARCHITECTURE

The Open Agent architecture (OAA)[7] is an agent-based system developed by SRI International [36]. The primary goal is to integrate heterogeneous applications in a distributed infrastructure. It provides a framework for the construction of distributed software systems, which facilitates co-operative task completion by flexible, dynamic configurations of autonomous agents. It is not developed for context aware computing, but it can be used as a context architecture. An example of a OAA-application is a phone agent that can plan a trip from one city to another by processing spoken sentences and using special strategies for producing responses.

4.6.1 Architecture

The OAA consists of multiple agents that contribute services to an agent community that is managed by a Facilitator This is illustrated in Figure 10. The key concepts of the OAA are:
- An agents informs its parent *Facilitator agent* of the services it can provide.
- The interpretation and execution of a task is a distributed process that is coordinated by the *Facilitator agent*.
- A single request can produce cooperation and flexible communication among many agents, written in different programming languages and spread across multiple machines.

Agents are initiated by applications. The agent library is a common infrastructure for constructing agents. Three main types of agents are distinguished:
- Application Agents that provide particular services such as speech recognition, data mining, and travel planning.
- Meta Agents that assist the Facilitator agent in co-ordinating the activities of other agents.
- User Interface Agents that monitor different input modalities (e.g. speech, pen gestures, etc.).

The Facilitator takes care of dividing a request into sub-problems and deciding which agents are available and capable of handling sub-parts of the request. Furthermore it takes care of all agent interactions required to handle the complex query. All communication is done using the Interagent Communication Language (ICL). Larger systems can be assembled from multiple facilitator/client groups, each having the structure that is shown in Figure 10. The Facilitator can also be used as a global datastore (a so-called blackboard). Sensor abstraction can be achieved by using an agent to put sensor-information on this blackboard.

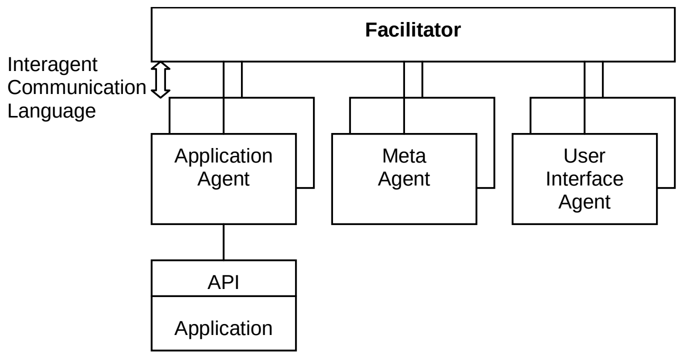

Figure 10 OAA system structure [36]

There is a trigger mechanism that enables agents to request that some action is taken when some set of conditions is met. An agent can install triggers locally (on itself), or remotely (on its facilitator or peer agents).

[7] The OAA is freely available at http://www.ai.sri.com/~oaa/main.html

There are four types of triggers:
- Communication triggers that allow incoming or outgoing events (from the Facilitator) to be monitored.
- Data triggers that monitor the state of a data repository (act upon addition or deletion of a fact).
- Task triggers that contain conditions that are tested after the processing of each event.
- Time triggers that monitor time conditions.

4.6.2 Requirements

The Open Agent Architecture does not pose any new requirements to a context architecture.

4.7 SCHILIT'S SYSTEM ARCHITECTURE

In 1995, Schilit presented in his Ph.D. thesis a system architecture that supported context-aware mobile computing [46]. This thesis was the first work on context aware computing and inspired many other research groups to work on this topic. The thesis is based on hands-on experience with the ParcTab system that is developed at the Xerox Palo Alto Research Centre [49]' [55]. The ParcTab system is based on palm-sized, wireless ParcTab computers and an infrared (IR) communication system that links them to each other and to desktop computers through a Local Area Network. Because of the communication with the IR-sensors in every room, the system is aware of the location of the different ParcTabs. Unfortunately, it is not possible to download the ParcTab system. The most popular applications of the ParcTab are:
- Electronic mail reader.
- Weather program.
- File browser, providing access to text and command files.
- Tab loader, allows users to store information in the tab's local memory.
- Note taking.

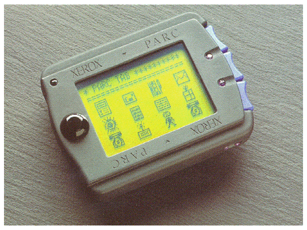
Figure 11 the ParcTab

4.7.1 Architecture

Schilit's architecture is an agent-based architecture that supports the gathering of context information about devices and users. It has three main components:
- Device agents that maintain the status and the capabilities of devices.
- Active maps that maintain the location information of devices and users.
- User agents that maintain user preferences.

For each ParcTab there is exactly one device-agent that acts like a switchboard to connect applications with a ParcTab via IR-senders/receivers. An agent performs four functions:
- It receives requests from applications to deliver packets to the ParcTab that it serves.
- In the reverse direction, it forwards messages (along with location identifiers) from its tab to the current application.
- It provides a source of tab location information for applications.
- It manages application communication channels.

The ParcTab system uses active maps to continuously know where each ParcTab is. An active map is a map of the building that is continuously updated. For example, it shows which ParcTabs are in the same room and notifies applications of location changes. The user agent keeps track of personal user information that might include, among other things, the preferences of that user, the characteristics of the devices that are currently in-use by that user, and the time and location of the most recent interaction. The user's own applications, as well as applications run by others, can request personal user information from the user's agent.

The ParcTab project is mostly about mobile computing. The focus is on location context, which is obtained by the IR-senders/receivers that have to be installed in every room. There is no support for the use of other sensor information.

4.7.2 Requirements

Schilit's architecture is an interesting architecture, but it does not provide any new requirements.

4.8 SITUATED COMPUTING SERVICE

The Situated Computing Service is a context architecture that is developed at Hewlett Packard Laboratories [30]. An example of an application that uses the Situated Computing Service is a handheld tourist guide that uses the knowledge of the tourist's current location to present relevant information. There is little information available regarding this context architecture and it is not possible to download the architecture.

4.8.1 Architecture

The Situated Computing Service interprets sensor data and makes it available to applications through a standard API. The main responsibilities of this service are:
- Combine and interpret sensor data and make it available for context aware applications.
- Send events to interested applications when context changes.
- Applications can query the Situated Computing Service for context information.

One experimental Situated Computing Service is developed with an emphasis on a common interface for sensors and constant availability of context acquisition. Furthermore some Quality of Service issues are emphasised (e.g. low latency).

4.8.2 Requirements

There is little information available concerning this architecture. Therefore, this architecture does not provide new requirements for a context architecture.

4.9 STICK-E NOTES

The stick-e notes architecture[8] is a general framework for supporting context-aware applications and is developed at the university of Kent [10], [9]. It is not intended for a distributed environment. It provides general support for specifying in what context an application programmer is interested. Data can be represented in a simple portable form, that can readily be exchanged and published, and can easily be converted to other forms (SGML). It can be kept separate from the application(s) that use it. An example of a Stick-e notes application is a reminder service that for example reminds the user of an upcoming appointment when she meets someone.

4.9.1 Architecture

The stick-e note is the electronic equivalent of a Post-it note. A stick-e note exists of two parts: a note portion that specifies the context that must be fulfilled and a body portion that

[8] Freely available at http://www.cs.ukc.ac.uk/projects/mobicomp/Fieldwork/Software/

specifies what action is to be taken when the context of the note is fulfilled. The syntax of a stick-e note is controlled by a specification, called a DTD, that defines the SGML tags that can be used. An example of a stick-e note that can be used in a tourist guide application is:

```
<note>
<required>
      <at> (1,4)..(3,5)
      <facing> 150..210
      <during> december
<body>
The large floodlit building
At the bottom of the hill
Is the cathedral.
```

The stick-e note architecture consists of three component types:

- SeTrigger: the triggering component that matches the user's present context with the context of loaded stick-e notes. When the context of a note matches, the note is triggered
- SeShow: triggered notes are passed to the execution component (that can be any existing program), so the execution component can execute the body of the note.
- SeSensor: the set of sensor components that periodically feed information to the triggering module. These components can be distributed over a network.

4.9.2 Requirements

The stick-e note is an interesting architecture, but it does not provide any functionality that the architectures described in the previous sections do not offer. Therefore, it does not provide new requirements.

4.10 SULAWESI

The Sulawesi[9] framework is developed at the University of Essex. The goal of the Sulawesi framework is to support multi-modal interaction on a wearable device [40], [41]. The Sulawesi framework uses agents to abstract the gathering of data from the different input sources, like mouse, pen-input and GPS receivers. The wearable system uses context (e.g. the user is walking) to determine the best way to deliver information. The idea behind the Sulawesi framework is to provide a "common" integration platform that is flexible enough to encompass a wide variety of input devices, separating the service (agent) development from the input mechanism. An example of a function of the Sulawesi architecture is to change from speech-output to a different output-modality (e.g. display) when the architecture detects that the user is talking.

4.10.1 Architecture

The Sulawesi framework consists of three parts:

- An input stage that gathers raw sensor data.
- A core stage that contains a natural language processing module and service agents to process information from the input stage.
- An output stage that decides how to present the results from the core stage.

[9] The Sulawesi architecture is freely available at http://wearables.essex.ac.uk/sulawesi/

Figure 12 illustrates the Sulawesi architecture.

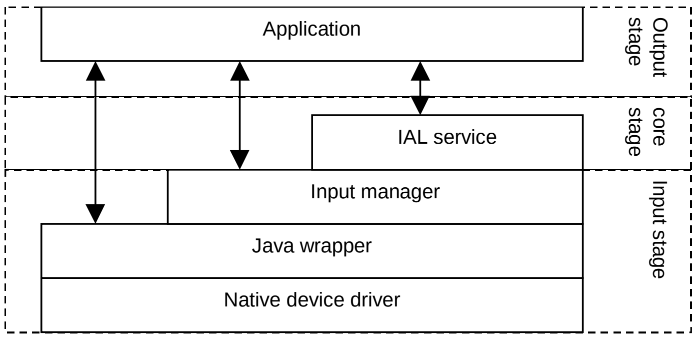

Figure 12 Sulawesi architecture [40]

The native device driver is wrapped in Java. The input manager agent translates the information provided by the sensor into intermediate information that is send to the Information Abstraction Layer (IAL) service agent. The IAL agent takes care of resource discovery. This agent combines the information from the input manager agent with information from other input manager agents (e.g. location information from GPS and from RF beacons) and offers the information to the application. The application still has access to the detailed sensor information, but will probably only use the information that is presented by the IAL service agent. Sulawesi provides a non-volatile system for task completion. This means that even if the batteries fail, the system knows what it was doing and with a small agent it is possible to automatically restart any unfinished jobs when the power is restored.

4.10.2 Requirements

The Sulawesi framework does not provide any new functionality compared to the architectures that are described in the previous sections. Therefore, the Sulawesi framework does not provide new requirements.

4.11 VIRTUAL INFORMATION TOWERS

The University of Stuttgart has developed a location-aware information system called Virtual Information Towers (VITs) [34]. This system is developed to present and access location-aware information with mobile clients. There is not much information available concerning the VIT project.

4.11.1 Architecture

Each VIT contains information specific to a certain geographical area. A VIT can contain a hierarchy of information objects called posters. These posters can be web pages, audio, video, etc. The contents of a VIT are summarised by a profile that can be used to search for VITs that contain information about a certain topic. After a while, information may not be relevant anymore. The VITs solve this problem by tagging information with a ‘time of last modification’ and a ‘time of expiration’. Furthermore, a VIT has an owner and access rights.

The University of Stuttgart has developed a prototype. This prototype consists of a VIT Client that is based on a web browser and allows its user to view the content of a VIT. Furthermore it notifies, if desired, its’ user (or other applications) of changes in the context. VITs can be found by using the global VIT Directory, which is a distributed service comparable to the Domain Name Server (DNS). The VIT Directory stores per VIT the name, the area of visibility, some keywords, and the address. The last component of the prototype is the VIT manager that stores and manages a number of VITs. Content that is stored on a VIT can be added, updated or deleted, using the VIT manager.

The only context that is used by the VIT project is location. The follow-up of the VIT project is the Nexus project [28]. This project is also only concerned with location aware applications. The focus has been put on the development of a Java library that allows the access of the functionality that the geographical information system provides. Furthermore there is a focus on how different location models can be used together.

4.11.2 Requirements

Since there is not much information available concerning the VIT system, it is not possible to extract new requirements from this architecture.

4.12 HEALTH COACH

This section deals with the Health Coach project that is being developed at Philips Research. This project develops a context aware application. However, since it is a Philips project and it is important to know which requirements of a context architecture are important to Philips Research, it is dealt with in this chapter. Health Coach is a system for people who are interested in maintaining or improving their physical condition. It informs about the current health situation and it advises on nutrition and activity based on (trends in) physical measurements.

4.12.1 Architecture

Health Coach provides the user with the following information topics: General health, Fitness, and Nutrition. It uses five sensors to calculate these three topics:
- Temperature
- Weight
- Heartbeat
- Blood pressure
- Activity

The current software architecture takes into account that one might add different sensors and different that types of data (e.g. energy balance based on the combination of weight and activity) might need to be calculated. A ‘proof of concept’ is developed that works as follows: a computer reads the raw sensor data from the sensor; interprets the raw sensor data by using the knowledge it has about the sensor; and stores the result together with a timestamp in a database. When the user wants to have information on measured or calculated data, the program uses all relevant information that is currently stored in the database to calculate the desired information. For some information (e.g. energy balance) a special mathematical formula is used.
A ‘software’ sensor contains the physical sensor, shows first order feedback to the user (e.g. the temperature), and reminds the user when another measurement is needed. Furthermore, the architecture contains a base station. The Health Coach application can ask the base station for information, e.g. the Fitness of a person. The base station takes care of all calculations. It has the following properties:
- RF capability. This is used for wireless communication with the sensor.
- Database. All sensor data is stored in a central database
- Knowledge. This part calculates the desired data by transforming data or by combining data from several sensors.
- Presentation. The presentation part shows the information on a display.
- Home Network. The base station is linked to a home network. This means that the Health Coach application can run on any device that is connected to the home network (e.g. Personal Digital Assistant, TV or Set-Top Box).

The interpretation of raw sensor data is done at the computer that runs the base station. This does not cause any traffic on the home network since a RF connection is used and the RF-receiver is directly connected to the base station. Of course, if the home network is (partially) a RF network, Health Coach *does* consume some bandwidth of this RF network. Further research will extend the application such that it will not only show the statistics of the users' health, but will also recommend what actions the user can take to improve her health.

4.12.2 Requirements

The Health Coach project is a context aware application, but it is not a context architecture (and it is not intended to). The requirements of a context architecture that are identified as the most important to the Health Coach project, are the following:

1. Privacy / security
Some information is private to people. It must be possible to indicate whether or not data is stored, how it is stored, how long it is stored and who is allowed to view the data.

2. Resource discovery
Now and then the sensor comes in range of the base station. There must be a resource discovery mechanism to identify when the sensor is in range and to notify the base station of this event.

3. *Possibility for low-level data handling*
When new sensors are added, the algorithms that use the sensor data require rather low level access to data from all sensors. It must be possible to obtain the raw sensor data.

4. Multiple interpretation of sensor data
It must be possible to interpret raw sensor data in different ways. One application might for example use an accelerometer to measure whether a user is moving or not; another application might want to use this sensor to measure whether a user is biking, walking or sitting.

5. Quality of Service
A QoS statement for data can be useful. Examples of QoS-metrics are accuracy, reliability, frequency and timeliness.

4.13 PERSONAL REMOTE CONTROL

The Personal Remote Control (PRC) project is another project at Philips Research that develops a context aware application [13]. The PRC application is a carrier application to demonstrate how the context aware abilities of the device can make the device more socially acceptable, and how the context aware abilities can decrease the need for explicit user input.

The PRC gives access to the users' content, both when she is at home and when she is away, in the form factor of a PDA-like device. Using this handheld device, the user can access an overview of the TV programmes for the next coming days, and of those that are recommended by the system. The user can view background information on programmes and can directly switch to the programs that are currently broadcasted (watch them on the TV). The user can set reminders to be notified before a programme starts. The user can also send and receive suggestions for programs to and from other users via SMS. Reminders and incoming SMS suggestions are presented to the user in the form that is suitable to the current use. Contextual cues obtained via a number of sensors determine when and how these signals should be presented to the user. The system is still being developed, but the system architecture is already coarsely defined.

4.13.1 Architecture

A desktop PC collects Electronic Programming Guide (EPG) information from the Internet and matches this with a user profile to generate recommendations. The EPG data and the recommendations are downloaded onto the PDA device. On the PDA, the user can interact with the EPG data and the recommendations. One of the functions that the PRC offers is to switch the Television to the channel where the recommended program is currently playing. Besides controlling the Television, the user can also communicate with other PRC users by sending SMS suggestions for Television programs.
The PRC reacts to the environmental dynamics and available resources. For example, it adapts the way messages and alerts are presented to the user by adjusting the visual interface. The PDA collects context information via a variety of sensors. These sensors are connected to a data-acquisition unit (DAQ), which essentially converts the analogue sensor data into digital signals that can be read by the PDA. Figure 13 shows that there are four conceptual stages between receiving sensor input and adjusting the interface.

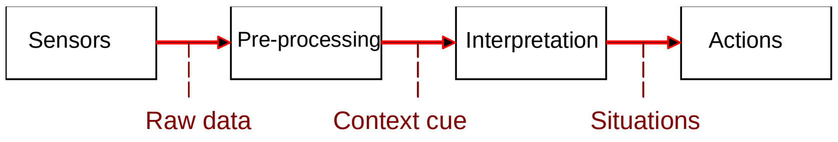

Figure 13 Flow of (context) information

The first stage exists of sensors that provide raw data to the system. This data is pre-processed in he second stage to generate context cues, for example ‘light’ vs. ‘dark’, or ‘noisy’ vs. ‘quiet’. Context cues are stored by a system component called the ‘Environment info manager’. Other system components can indicate to this manager in what context information they are interested. An example of such a component is the ‘Reasoner’, which takes care of stage three: the interpretation of the collection of cues into a higher level of context (situations), for example ‘inside pocket or bag’. Table 1 illustrates how different sensor input can lead to different situations.

Table 1, Situation model

Situation \ Sensor	Light sensor front	Light sensor back	Touch sensor	IR sensor	Microphone	Cricket system
In_pocket_or_bag	dark	dark	not touched	-	-	-
Outside_pocket_or_bag	(back=dark) -> ligh	(front=dark) -> light	-	-	-	-
Held_by_user	-	-	touched	user detected	-	-
Not_held_by_user	-	-	not touched	-	-	-
User_active_nearby	-	-	-	activity detected	-	-
No_user_active_nearby	-	-	not touched	no activity detected	-	-
Close_to_tv	-	-	-	-	-	in range
Far_from_tv	-	-	-	-	-	not in range
Noisy_environment	-	-	-	-	high noise level	-
Quiet_environment	-	-	-	-	low noise level	-
Tv_is_on	-	-	-	-	-	tv on
Tv_is_off	-	-	-	-	-	tv off

System components can subscribe to certain events (e.g. the situation ‘inside pocket or bag’ occurs). These components take care of the last stage that decides which combinations of situations and events should lead to actions that generate responsive behaviour. At this moment, context information such as context cues and situations is not available for other applications. Furthermore, the context acquisition is intended to work ‘stand-alone’. It does not use sensor information from any external system.

4.13.2 Requirements

The Personal Remote Control is a context aware application, not a context architecture (and it is not intended to). The requirements of a context architecture that are identified by the team of this project are:

1. Handling uncertainties
It is difficult to interpret sensor data into situations. There must be a mechanism to deal with uncertainty in the observed context information.

2. Extensibility
It must be easy to add new sensors, new ways to interpret sensor data into cues, and new ways to interpret cues into situations.

4.14 PHENOM

The Phenom project is a Philips project that exists of four PhD projects. Together, these projects build a prototype of a context aware application: the Memory browser [6]' [26] The Memory Browser is a portable photo browser. People can store their digital photos in a home-network and, when they are in the home, they can use multiple devices (e.g. a portable screen, a television) to view their pictures. It is also possible to make and view personalised associations between photos and ornaments within the home. This means that when such an ornament comes close to the portable browser device, the photos that are associated with that ornament and the current user, are displayed in the photo browser.

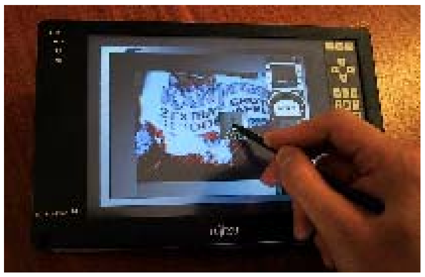
Figure 14 Memory Browser

4.14.1 Architecture

Figure 17 shows the architecture of the Memory browser. The main component is the 'Castle' that runs on the Java Virtual Machine (JVM). One castle can have multiple 'Servants'. Empires, Castles and Servants are explained below.

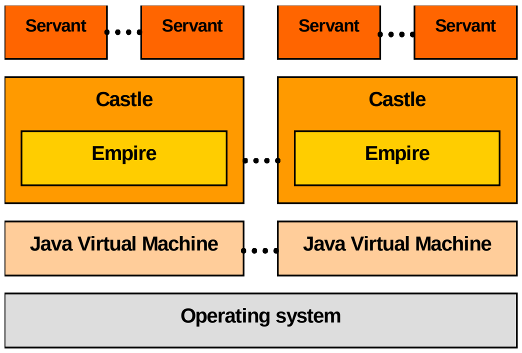

Figure 15 Physical architecture

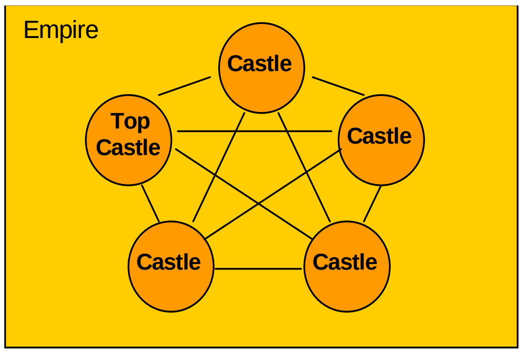

Figure 16 Logical architecture

Empire
Figure 16 shows the logical view of an empire. An empire is a logical network of castles (or 'hosts') that are connected with each other through socket connections. The empire is managed by the ‘Top Castle’. Each castle can be a top castle, since every castle has empire capabilities. The top castle takes care of network management. It has a list of castles and takes care of adding and deleting castles to this list. Furthermore, it takes care of joining empires.

Castle
A castle is a software component that maintains a list of servants and that takes care of communication issues. A castle runs a number of servant threads. A castle maintains a list of its’ own servants and the servants of the other castles in the empire. Another function of the castle is transparent communication between servants: a castle has socket connections with all other castles in an empire and takes care of all communication issues.

Servants
A servant is an autonomous software component that may provide computation, storage, access to a hardware device such as a sensor, etc. A servant object is a normal thread that has the added functionality to communicate with any other servants in the empire. If a servant needs to communicate, it asks its’ castle for a specific servant or for a specific type of servant. The castle returns the (proxy to the) servant. The servant can communicate with this servant without knowing whether or not that servant is running on a different machine. A servant has a logging functionality that can be used for debugging. Furthermore, servants can subscribe with other servants to be notified of certain events.

The Empire/Castle/Agent architecture has the following properties:
- Resource discovery. Castles can easily join the network (empire) and keep a list of all the servants in a network.
- Transparent communication. Servants can communicate with each other without knowing that a certain servant might be on a different machine.
- Subscription/ notification mechanism. Servants can poll and subscribe with other servants to be informed of certain events.

Servants present context information for external use and take care of interpreting sensor data. The contexts that play an important role are the location and identity of the users of the handheld device, the location of other people in the room, the location of ornaments (tagged objects), and the location of the portable browser device itself. A tracking system that is installed in one single room, keeps track of whether persons and objects are inside or outside the room by means of detection at the door. Persons and objects are tagged with Radio Frequency (RF) tags. A tag reader is attached to the table in the room, since mounting the tag reader to the portable device costs too much battery power. The portable browser device is informed of tracking events through its wireless link. The location of all other active objects (e.g. a screen, scanner or digital camera) is detected using network discovery services.

Figure 17 shows how the Phenom project deals with context information. The *TagReader* servant looks for tags and translates tag IDs to IDs of persons/objects. Another way of obtaining context information is by using a graphical user interface. The person/object ID is forwarded with its location to the *Tracker* that keeps track of current person-, object-, and device locations. The Tracker is not aware of the specific tracking hardware being used. It is only interested in events that describe the presence or absence of people, objects and devices. Servants or applications can register with the Tracker servant to get informed about the location changes of people or objects.

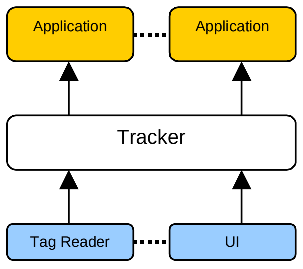

Figure 17 Dealing with context

Future research will integrate an ultra-sound positioning system to provide (more) location information. Furthermore, speech in- and output will be added to the system.

4.14.2 Requirements

The Empire-framework that is used by the Memory Browser is a sort of context architecture. The Phenom team identifies the following requirements of a context architecture:

1. *Transparent communication*

A functionality that is important for every distributed application is transparent communication. The context architecture could deal with this. However, it must be possible to use different communication mechanisms (e.g. RMI and Corba).

2. *Resource discovery*

Resource discovery is another standard functionality in a distributed application. New services and resources should be detected and their capabilities should be available for other services. This function is typically a function of the middle ware layer. A context architecture could include this functionality, but it must always be possible to work together with different resource discovery mechanisms.

3. *Subscription/ polling mechanism*

The context architecture must enable applications (and services) to subscribe with services to be informed of certain events, or to poll a service to get the current information that a service can provide.

4. *Availability of context information 'everywhere'*

It must be possible to store context information both local (e.g. on a PDA) and central (e.g. on a home network) at the same time.

5. *Speed*

The propagation of context information from the sensors to the application that needs the information, might need to be fast. It depends on the application whether or not speed is an important requirement. However, the architecture must be able to deliver context information within a specific period of time.

6. *Recovery Support*

When a part of the context architecture does not function anymore this should have a minimal effect on the rest of the system.

7. *Debugging support*

Logging is used for debugging. The Memory Browser has a logging mechanism that allows users to view different levels of logging information (different periods, different levels of detail).

4.15 WWICE

This section describes the WWICE project [27], [52]. The WWICE project is a project at Philips Research that explores new applications in the electronic market that offer more entertainment, comfort, and flexibility in a networked home environment. It focuses on new application concepts, the corresponding interaction concepts for easy access and control, and the system architecture needed to support these concepts. In particular, the system architecture has to deal with issues concerning distributed computing. In fact, the project that is described is the WWICE 2 project (which is, obviously, the follow-up of WWICE 1). From this point forward, the WWICE-2 project is called the WWICE project.

The WWICE system takes care of many problems that arise due to the distributed nature of the WWICE system. For example, it takes care of resource discovery, URN resolving, and remote procedure calls. The WWICE system contains (among other things) services. A service is an entity of the WWICE system that can provide a service to a client. Services are (remote) software objects with a type and an associated well-defined API that can be invoked by clients. An example of a service is a resource. A 'resource' is a software object that offers an interface to the functionality of a platform resource. A platform resource is either hardware (e.g. hard disks, MPEG-encoders, and sensors) or platform software (e.g. low level drivers and operating system). The WWICE system contains a subsystem, the UI Shell, that takes care of resource management with respect to resources that provide UI-functionality.

4.15.1 Context in WWICE

The WWICE system contains two services that deal with context information: the Presence Monitor and the State monitor. The Presence monitor can provide the physical location and orientation of persons and devices. There can be multiple Presence monitors in one WWICE network (however, one would do). A client can subscribe with a Presence monitor to be notified of changes in the location of persons and devices. The Presence Monitor does not know the location with 100 % certainty: it appoints a certainty factor to every location. The Presence Monitor subscribes with services that can provide information about the physical locations or the orientation of devices and persons (e.g. resources that represent sensors). Whenever one of these services has some new information, it reports this to the Presence Monitor. The Presence Monitor adds this observation as a fact to its knowledge database, called the Presence Fact Base. After each addition of a new fact, the Presence Monitor uses an inference mechanism to look at the total content of the Presence Fact Base to draw new conclusions. Next, it sends a notification to its clients (if necessary).

The State monitor monitors the state of ‘entities’. An entity can be a person or a device, but also an application. For example, the State monitor might monitor whether an application starts or stops. There can be multiple State monitors in one WWICE network. A client can subscribe at a State monitor with a (event, {conditions})-tupel. The ‘event’ is the event that the State monitor must use to determine when to evaluate the set of conditions. The set {conditions} is the set of conditions that should hold in order to cause the State monitor to notify the client. The State monitor subscribes with services that provide the information that it needs to determine when events occur, and to evaluate the conditions. For example, a client could subscribe with the event ‘Homer enters the kitchen’, and the condition ‘the time must be between 8:00h and 8:30h’. In this situation, the State monitor subscribes with the Presence monitor to be notified of changes in the location of the person ‘Homer’. If the location of the person ‘Homer’ becomes ‘Kitchen’, the State monitor evaluates the condition and sends a notification (if necessary).

The monitors are not aware of sensors being connected to the system. A resource is the interface between the sensor and the WWICE network. Sensor resources take care of the translation of the sensor specific measurements into measurements in WWICE units (coordinates, user IDs, etc.). Sensor measurements that are sent to the presence monitor are considered as facts. The certainty of such a fact decreases in time when measurements are not continuous. For sensors it is assumed that:

- They can provide a validation (‘enter’) or an invalidation message (‘leave’).
- They can provide persistent (information valid until invalidated) or one shot (instantaneous) information.
- They provide accurate and reliable information. No special measures are taken with relation to handling of unreliable / inaccurate information.
- They filter sensor data before presenting it to the WWICE system.

At this moment, the WWICE system uses the following context:

- Location of devices within certain areas (e.g. ‘near television’, ‘in living room’).
- Location of persons.
- Proximity to stationary screens. Location of persons within certain areas.
- Proximity to portable screens. Location of persons relative to (portable) devices.
- Absolute location of persons.
- Identity of persons.

A context type that seems to be a promising research topic is the ‘activity’ of a person (e.g. sleeping, having dinner, and watching television). An application area where the context ‘activity’ can be used to improve applications is reachability management. The reachability of a person indicates what types of communication can be used to communicate with a person and to what extent a person is willing to start a particular type of communication (e.g. telephone communication, video communication, or email). If an application knows in what activities a user is involved, it can determine, based upon the user profile, whether or not the user wants to be disturbed by communication. The WWICE system contains a model to define area knowledge.

4.15.2 Requirements

The WWICE system does not introduce new requirements of a context architecture. This sub-section explains which properties of a context architecture are important according to the WWICE team.

1. *Support for context descriptions*

At this moment, the only context information used is location and identity. The implications of adding different sorts of context information to the architecture are not known. Especially the context type 'activity' might be useful to many context aware applications.

2. *Quality of Service*

The certainty of information is identified as an important QoS-metric. Sometimes sensors are failing or need calibration and thus do not give correct information. At this moment this issue is ignored. Furthermore, some time after an event causes the conclusion of a certain 'situation' (e.g. a user identified herself using the bedroom television and thus it is concluded that this person is in the bedroom), the probability that this situation is still true diminishes. Currently, the certainty exponentially decreases in time. A more sophisticated mechanism might be needed.

4.16 REQUIREMENTS CONTEXT ARCHITECTURE

Section 4.1 to section 4.15 have described many different types of context architectures. Every section has extracted new requirements from these context architectures (if any). These requirements are divided into three classes: requirements that are common to all distributed applications, requirements that are specific for context aware applications, and 'requirements' that are merely useful extensions to a context architecture.

1.8.1 Requirements concerning distributed computing

Some of the requirements that are stated in the previous sections are not specific to a context architecture, but are common to all distributed applications. Since the context architecture is supposed to work in an in-home environment, it has to deal with these requirements. However, these issues will already be solved when the context architecture is part of a larger system (e.g. the WWICE[10] system).

§ ***Resource discovery***

There should be a resource mechanism (e.g. a registry) that is responsible for finding components (keeping a list of available components) and for providing an application with ways to access them.

§ ***Resource management***

When multiple context aware applications use the same resources (e.g. displays or speakers), resource conflicts might occur. There has to be a resource management mechanism that deals with these resource conflicts.

[10] Section 4.15 explains the WWICE system in detail.

§ ***Transparent distributed communications***
The fact that communication is distributed should be transparent to both sensors and applications. This relieves the designer of having to build a communications framework.

§ ***Different platforms***
Since the context architecture is meant for distributed computing, the architecture must work on different platforms (Windows, Linux, etc.)

4.16.2 Requirements specific for context aware applications

This subsection deals with the most important requirements of a context architecture. A context architecture must deal with all these requirements.

§ ***Context specification (subscription / polling mechanism)***
An application must be able to query the context architecture for context information or to subscribe (with a set of conditions) to certain context information.

§ ***Context interpretation***
Multiple applications often perform a similar interpretation of context information (e.g. transforming an identification number into an email address). Therefore, the context architecture must provide interpretation such that interpretation only needs to be done once. It must be possible to use different techniques to interpret context information. Examples of interpretation techniques that might need to be supported are operators such as OR, XOR and NOT, learning techniques such as neural networks, classification techniques such as Hidden Markov Modelling, and the follow-by operator that lets a programmer specify that one event must follow another event in time.

§ ***Constant availability of context acquisition***
It is possible that several applications request the same set of context. The components that acquire context must execute independently of the applications that use them such that the context information is always available to different applications. An additional advantage is that the application designer needs not to instantiate, maintain or keep track of components that acquire context.

§ ***Common interface***
Context information that is used by applications might come ‘directly’ from sensors (after interpretation) or there might be several layers of abstraction. For example, to calculate whether or not a meeting is going on in a specific room, there might be a need to calculate how many people are in that room, whether or not a meeting is scheduled, and whether or not the people are talking to each other. The application designer does not need to know that (multiple) interpretation steps are necessary to achieve this context information. Therefore, all components need to have a common external interface such that an application can treat them in a similar fashion.

§ ***Context storage***
Context history must be stored such that applications, but also the context architecture itself, can use it to establish trends and predict future context values.

§ ***Support for context descriptions***
It is difficult to describe context. There is a need for a structure to develop a context terminology. Issues that need to be defined are the values that context can have, the services that the architecture delivers (e.g. is it possible to request the names of all persons in room X), etc. A common terminology makes it easier to develop context aware applications, since it can be used to define the interface between applications, the architecture, and sensors.

§ ***Model of the Environment***
A model of the environment that is available to both the architecture and applications, enables both to perform more sophisticated inferencing. An example of a model is the map of a home. If this knowledge is placed in the architecture, all context-aware applications can query it for

information relevant to the application's particular domain. Important issues that have to be dealt with when one develops a model, are:

- It must contain a static model (issues are granularity, which objects to represent, which relationships to represent, etc.)
- It must be scalable. The static model must be dynamically updated when people, devices or locations are added or deleted from the model, e.g. it should be easy to add a new location to the map of a house.
- Transformations: it must be possible to transform one model into another model. For example transforming a location model that presents location as an (x,y,z)-coordinate into a model that presents location in a room-based granularity.
- The model must deal with contradictionary facts. Often multiple sensors are combined to obtain information. This means the system must be able to deal with contradictionary input.

4.16.3 Extensions

This subsection states 'requirements' that are merely extensions that can be useful when developing applications with a context architecture.

§ ***Controlling access to context***
Context information might be private information thus the access to context must be controlled.

§ ***Recovery support***
When a part of the system fails, it must be possible to automatically restart any unfinished jobs when the system works again. Furthermore, the loss of a part of the system should have a minimal impact on the rest of the system.

§ ***Quality of Service***
Context components should indicate Quality of Service values for the context they can provide to applications and notify subscribers of any changes to these values. Examples of Quality of Service metrics are ambiguity, reliability, coverage, resolution, frequency and timeliness. *Ambiguity* deals with ambiguous context information. *Reliability* deals with sensor failures and uncertainty of sensor information due to 'ageing' (after some time, information is not up-to-date anymore). *Coverage* defines the set of all possible values for a context attribute. *Resolution* defines the actual change that is required for the context attribute to change. Frequency and timeliness determine the "real-time" requirements of the application. *Frequency* defines how often the information needs to be updated and *Timeliness* defines the time the application allows between the actual context change and the related notification to the application.

§ ***Debugging***
A useful feature of a system is support for debugging. A powerful way to support debugging is the graphical presentation of state information.

§ ***Support for alternative implementations of architecture features***
A useful feature of a context architecture is the possibility to replace the default implementation of features (e.g. communications schemes) at instantiation time. The instantiated component should be able to automatically replace the default implementation and use the alternative implementation. This is useful if you want to incorporate a context architecture into an existing project.

§ ***Prototyping Environment***
It must be possible to test applications without using real sensors. Software sensors in the form of GUIs can collect input explicitly from the user. When these GUIs are treated as sensors, one can easily prototype an application.

4.17 COMPARISON CONTEXT ARCHITECTURES

Table 2 indicates to what extent the architectures that are described in the previous sections, comply with the requirements that are summarised in section 4.16.

Table 2, comparison of context architectures

√ = covered
½ = partly covered

		REQUIREMENTS																
		Distributed				Context							Extensions					
		Resource discovery	Resource Management	Transparent distributed communications	Different platforms	Common interface	Context specification (subscription / polling mechanism)	Context interpretation	Constant availability of context acquisition	Context storage	Model of the Environment	Support for Context Descriptions	Controlling access to context	Recovery support	Quality of Service	Support for alternative implementations of architecture features	Debugging	Prototyping Environment
ARCHITECTURES	Context Toolkit	√		√	√	½*	√	√	√	√						√		
	Contextual Information Service					½*	½	√	√		√	√			½			
	Easyliving	√		√	√	√	½	½	√		½				½			
	Hive	√	√	√	√	√			√				√					
	Metaglue	√	√	√	√	½*	½		√					√			√	
	Open Agent Architecture	√	½	√	√	√	√	√										
	Schilit's system architecture	½		√		½*	½		√		√							
	Situated computing service					½*	√	½	√									
	Stick-e notes	½				½*	½		√									
	Sulawesi	½	√		√	½*	½		√		√							
	Virtual Information Towers	√				½*			√	√	√	√	√		½			
	Health Coach**							½		½								
	Personal Remote Control**					½	√	½			½							√
	Phenom	√		√	√	½	√		√	½	½			√			√	½
	WWICE	√	√	√	√	½	√	√	√	√	√						½	√

* *These architecture have a way to 'hide' the fact that sensors or multiple interpretation layers are used. However, they do not have a common interface defined for all components of the architecture.*
****** *These systems are context aware applications and not context architectures (and not intended to).*

The most important property of the architecture to use as a basis for future research is that it must be possible to obtain enough information about the (implementation) details of this architecture. The architectures outside Philips Research that provide detailed information are the Context Toolkit, Hive, the Open Agent Architecture, and the Sulawesi architecture. Obviously, there is enough information available concerning Health Coach, Personal Remote Control, Phenom and WWICE since these are all projects within Philips Research. However, since the Health Coach- and the Personal Remote Control system are context aware applications and not context architectures, these systems are not suitable to use as a basis for future research. Another important condition is that the context architecture is supposed to work in a distributed environment. Therefore, the Sulawesi architecture is not suitable as a basis for future research.
The Context Toolkit has the useful property that the way communication between objects is handled can be changed. This might make it easier to incorporate this architecture in, for example, the WWICE architecture. However, apart from this useful feature, the Context

Toolkit does not offer any functionality that the WWICE system does not offer. Therefore, this architecture is, compared to the WWICE system, less suitable to use as a basis for future research.
Hive and the Open Agent Architecture (OAA) are both agent-based systems. It might be interesting to investigate to what extent agent systems are useful to use as a context architecture. Hive agents use Java RMI for distributed communication and the Resource Description Framework (RDF) to represent the semantics of the services of agents. The RDF specification provides a lightweight ontology system to support the exchange of knowledge on the Web. Philips Research is one of the companies involved in developing the RDF specification. The OAA uses its own communication language (instead of RDF) and uses a Facilitator to take care of dividing an agent-request into sub-problems and deciding which agents are available and capable of handling sub-parts of the request. Since Hive uses RDF this architecture is, with respect to the agent-based context architectures, the most suitable to use as a basis for future research.
However, since all the details of the WWICE and the Phenom system are available and it is easy to consult the developers of these systems, these are more suitable to use as a basis for future research. Furthermore, of all architectures the WWICE system complies with the most requirements. Therefore, the WWICE system is the most suitable to use as a basis for future research.

The rest of this chapter explains the interesting features of the remaining architectures. Easyliving has given a lot of thought on how to develop a good location model (for a room). The Metaglue system is an interesting system with respect to managing agents, especially concerning the graceful degradation of an agent system. However, it does not have especially interesting features with regard to context architectures. The Schilit architecture is mainly a location-based architecture and does not fulfil a lot of the requirements of a context architecture. The other architectures (Situated Computing Service, Stick-e notes, and the Virtual Information Tower project) do not bring in any interesting features to incorporate in future research since there is not enough information available concerning these projects to extract new research topics.

5 RESULTS

This chapter presents the results of the objectives that are stated in section 1.3. These objectives are:

1. Definition of the concepts ‘context’ and ‘context awareness’.
2. Analysis of context aware applications. What types of context do context aware application use, and to what purpose?
3. Analysis of context architectures. What are the requirements of a context architecture?
4. Based upon the survey of context architectures and the analysis of the requirements of a context architecture, a choice for an architecture that can be used as a basis to develop a context architecture as part of this graduation project.

1. Definitions

Chapter 2 explains the definitions of the concepts ‘context‘, ‘context aware‘ and ‘context architecture‘. These concepts are defined as follows:
“ Context is any available (sensor) information that is relevant for an application, excluding the explicit input from and output to the user ”
" A system is context aware if it uses context to improve its performance or provide relevant information and/or services, where relevancy depends on the user’s task "
“ A context architecture is a software architecture that supports the development of context aware applications “

2. Analysis context aware applications

Section 2.2 and chapter 3 analyse context aware applications. Most context aware applications use the context types ‘location‘, ‘identity‘, and ‘time‘. Chapter 3 presents a rough classification of how context is used to improve the operation of applications: context can be used to add new functionality to a system, to provide (context dependent) information, to improve the interaction with a device by reducing the interaction, to enhance the interaction by providing new interaction modalities, and to create entire new applications.

3. Analysis context architectures

Chapter 4 gives an overview of the research in context architectures, and extracts the requirements of a context architecture. These requirements are summarized in section 4.16. The requirements are divided into three classes: requirements that are common to all distributed applications, requirements that are specific for context aware applications, and ‘requirements’ that are merely useful extensions to a context architecture. The most important requirements are the requirements that are specific for context aware applications:

- Context specification (subscription / polling mechanism). An application must be able to query the context architecture for context information or to subscribe (with a set of conditions) to certain context information.
- Common interface. The application designer does not need to know that (multiple) interpretation steps are necessary to achieve context information. Therefore, all components need to have a common external interface such that an application can treat them in a similar fashion.
- Support for context descriptions. A common terminology makes it easier to develop context aware applications, since it can be used to define the interface between applications, the architecture and sensors.
- Context interpretation. Multiple applications often perform a similar interpretation of context information (e.g. transforming an identification number into an email address). Therefore, the context architecture must provide interpretation such that interpretation only needs to be done once.
- Constant availability of context acquisition. The components that acquire context must execute independently of the applications that use them such that the context information is always available to different applications.
- Context storage. Context history must be stored such that applications, but also the context architecture itself, can use it to establish trends and predict future context values.

- Model of the environment. A model that is available to both the architecture and applications enables both to perform more sophisticated inferencing. An example of a model is the map of a home.

4. Choice for a context architecture

The most important property of an architecture that can be used as a basis for future research, is that it must be possible to obtain enough information about the (implementation) details of this architecture. The architectures outside Philips Research that provide detailed information are the Context Toolkit, Hive, the Open Agent Architecture, and the Sulawesi architecture. Another important condition is that the context architecture is supposed to work in a distributed environment. Therefore, the Sulawesi architecture is not suitable as a basis for future research. Section 4.17 shows that the WWICE system complies with the most of the requirements of a context architecture. Furthermore, the WWICE system is developed within Philips research. This makes it is easy to consult the developers of the system, and all documentation of the system is available. Therefore, the WWICE system is the most suitable to use as a basis for future research.

6 CONCLUSIONS

This section states the conclusions of this literature study.

§ ***Most context aware applications use only location, identity and time***
Context aware computing is a relatively new research area. Many research groups have developed some kind of context aware application, though these applications usually employ the same types of context: location, identity and time. A context type that seems to be a promising research topic is the ‘activity’ of a person (e.g. sleeping, having dinner, and watching television).

§ ***The WWICE system is the most suitable to use as a basis for future research***
Some research groups have identified the need for a context architecture that helps to develop context aware applications. The WWICE project has developed a system that, of all investigated architectures, fulfils the most of the requirements that a context architecture should comply with. Furthermore, all details of the WWICE system are available and it is easy to consult the developers of this system. Therefore, the WWICE system is the most suitable to use as a basis for future research

§ ***Further research should explore ways to use new types of context***
Since the WWICE system already deals with many requirements of a context architecture, the challenge is not to built a context architecture, but to explore ways to use new types of context. A context type that seems to be a promising research topic is the ‘activity’ of a person (e.g. sleeping, having dinner, and watching television). An application area where the context ‘activity’ can be used to improve applications is reachability management. The reachability of a person indicates what types of communication can be used to communicate with a person and to what extent a person is willing to start a particular type of communication (e.g. telephone communication, video communication, or email). If an application knows what activities a user is involved in, it can determine, based upon the user profile, whether or not the user wants to be disturbed by communication.

BIBLIOGRAPHY

[1] Aarts E.H.L., Ambient Intelligence: calming, enriching and empowering our lives. *Philips Research Password*, 8, 2001, http://www.research.philips.com/InformationCenter/Global/.

[2] Aarts E.H.L. and Harwig H.A., Ambient intelligence: a new user experience. *Cebit 2000*, 2000.

[3] Abowd G.D., Atkeson C.G., Hong J., Long S., Kooper R., and Pinkerton M., "CyberGuide: A Mobile Context-Aware Tour Guide". *ACM Wireless Networks*, 3, No. 5, pp. 421-433, 1997.

[4] Asthana, A., Cravatts, M., and Krzyzanowski, P., "An Indoor Wireless System for Personalized Shopping Assistance". *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994, http://citeseer.nj.nec.com/asthana94indoor.html.

[5] Beigl M., "MemoClip: A location-based remembrance appliance". *Personal Technologies*, 4, No. 4, 2000, http://citeseer.nj.nec.com/beigl00memoclip.html.

[6] Bijsterveld M., Eggen B., Faihe Y., Loo S.van, Tedd D., and Udink R., *Phenom project description*, 2000.

[7] Boasson M., "The Artistry of Software Architecture". *IEEE Software*, 12, No. 6, 1995.

[8] Brown P., Burleson W., Lamming M., Rahlff O., Romano G., Scholtz J., and Snowdon D., "Context awareness: some compelling applications". *2nd International Symposium on Handheld and Ubiquitous Computing*, 2000, http://www.dcs.ex.ac.uk/~pjbrown/papers/acm.html.

[9] Brown P.J., "Some Lessons for Location-Aware Applications". *Proceedings of first workshop on HCI for mobile devices*, Glasgow University, 1998, http://www.dcs.gla.ac.uk/~johnson/papers/mobile/HCIMD1.html#_Toc420818982.

[10] Brown P.J., Bovey J.D., and Chen X., "Context-Aware Applications: From the Laboratory to the Marketplace". *IEEE Personal Communications*, 4, pp. 58-64, 1997, http://www.cs.ukc.ac.uk/people/staff/pjb/papers/personal_comms.html.

[11] Brumitt B., Krumm J., Meyers B., and Shafer S., "Easyliving: ubiquitous computing & the role of geometry". *IEEE Personal Communications*, 2000, http://research.microsoft.com/easyliving/publications.htm.

[12] Buil V.P. and Destura G.J., *Sensor and Input Technologies - only available within Philips Research*, Philips Electronics N.V., 2001.

[13] Buil V.P., Lashina T., Vignoli F., Wijdeven S.v.d., Leeuwen M.van, Haryono Y., Kortenoeven R., and Regt M.de., *Concepts for a context aware personal remote control - only available within Philips Research*, 2002.

[14] Chen G and Kotz D., *A Survey of Context-Aware Mobile Computing Research*, Technical Report TR2000-381, Dartmouth College, Dept. of Computer Science, 2000.

[15] Cheverst K., Davies N, Mitchell K., and Efstratiou C., "Using Context as a Crystal Ball: Rewards and Pitfalls". *Personal Technologies*, 5, No. 1, pp. 8-11, 2001.

[16] Coen M.H., Phillips B., Warshawshy N., Weisman L. , Peters S., and Finin P., "Meeting the computational needs of intelligent environments: the MetaGlue environment". *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99*, Dublin, Ireland, 1999, http://www.ai.mit.edu/people/mhcoen/metaglue.pdf.

[17] Davies N., Mitchell K., Cheverst K., and Blair G., Developing a context sensitive tourist guide. 1998, http://citeseer.nj.nec.com/davies98developing.html.

[18] DeVaul R W. and Pentland A., The Ektara architecture: The right framework for context-aware wearable and ubiquitous computing applications. 2000, http://www.media.mit.edu/~rich/DPiswc00.pdf.

[19] Dey A.K., "Context-Aware Computing: The CyberDesk Project". *AAAI 1998 Spring Symposium on Intelligent Environments*, 1998, http://www.cc.gatech.edu/fce/cyberdesk/pubs/AAAI98/AAAI98.html.

[20] Dey A.K., *Providing Architectural Support for Building Context-Aware Applications*. Thesis for the degree of 2000, http://www.cc.gatech.edu/fce/contexttoolkit/.

[21] Dey A.K. and Abowd G.D., "CybreMinder: A Context-Aware System for Supporting Reminders". *Proceedings of Second International Symposium on Handheld and Ubiquitous Computing*, Springer Verlag, Bristol, UK, 2000.

[22] Dey A.K. and Abowd G.D., "Towards a Better Understanding of Context and Context-Awareness". *Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000)*, The Hague, The Netherlands, 2000.

[23] Dey A.K., Abowd G.D., and Wood A., "CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services". *Proceedings of the 1998 Intelligent User Interfaces Conference*, San Francisco, CA, 1998.

[24] Dey A.K., Futakawa M., Salber D., and Abowd G.D. , "The Conference Assistant: Combining context-awareness with wearable computing". *Proceedings of the 3rd International Symposium on Wearable Computers (ISWC'99)*, 1999, http://www.cc.gatech.edu/fce/ctk/pubs/ISWC99.pdf.

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

[26] Dijk E., Hoven E.v.d., Loenen E.V., Qian Y., Tedd D., and Teizeira D., *Phenom prototype design - only available within Philips Research*, 2000.

[27] Eggenhuizen H.H., *WWICE: Window on the World of Information, Communication and Entertainment*, Philips Research Password, 2000.

[28] Fritsch D., Klinec D., and Volz S., "NeXus - Positioning and Data Management Concepts for Location Aware Applications". *Proceedings of the 2nd International Symposium on Telegeoprocessing*, Nice-Sophia-Antipolis, France, 2000, http://www.nexus.uni-stuttgart.de/.

[29] Hinckley K., Pierce J., Sinclair M., and Horwitz E., "Sensing Techniques for Mobile Interaction". *ACM UIST 2000 Symposium on User Interface Software and Technology, CHI Letters*, 2, 2, 2000.

[30] Hull R., Neaves P., and Bedford-Roberts J., "Towards Situated Computing". *1st International Symposium on Wearable Computers*, 1997.

[31] Institute of the Future, "Sensors: The Next Wave of Infotech Innovation". *1997 Ten-Year Forecast*, 1997, http://www.iftf.org/html/iftflibrary/technology/sensors.pdf.

[32] Korkea-aho M., "Context-Aware Applications Survey". *Internetworking Seminar, spring 2000*, 2000, http://www.hut.fi/~mkorkeaa/doc/context-aware.html.

[33] Lamming M. and Flynn M., ""Forget-me-not" Intimate Computing in Support of Human Memory". *Proceedings of FRIEND21, '94 International Symposium on Next Generation Human Interfaces*, 1994.

[34] Leonhardi A., Kubach U., Fritz A., and Rothermel K., "Virtual information towers - a metaphor for intuitive, location-aware information access in a mobile". *3rd International Symposium on Wearable Computers*, San Francisco, California, 1999, http://citeseer.nj.nec.com/218437.html.

[35] Lieberman H. and Selker T., "Out of Context: Computer Systems that Adapt to, and Learn from, Context". *IBM Systems Journal*, 39, No. 3 & 4, pp. 617-632, 2001, http://www.research.ibm.com/journal/sj/393/part1/lieberman.html.

[36] Martin D., Cheyer A., and Moran D., "The Open Agent Architecture: a framework for building distributed software systems". *Applied Artificial Intelligence*, 13, No. 1999, pp. 91-128, 1999, http://citeseer.nj.nec.com/126894.html.

[37] Meyers B., Brumitt B., Krumm J., Kern A., and Shafer S., "EasyLiving: Technologies for Intelligent Environments". *Handheld and Ubiquitous Computing*, 2000, http://research.microsoft.com/easyliving/publications.htm.

[38] Nelson G.J., *Context-Aware and Location Systems*. Thesis for the degree of University of Cambridge, United Kingdom, 1998, http://www.acm.org/sigmobile/MC2R/theses.html.

[39] Nelson M., Gray M., Roup O., Krikorian R., and Maes P., "Hive: Distributed agents for networking things". *IEEE Concurrency*, 8, pp. 24-33, 2000, http://nelson.www.media.mit.edu/people/nelson/research/hive-asama99/hive-asama99.ps.gz.

[40] Newmann N.J., "Sulawesi: A wearable application integration framework". *Proceedings ofthe 3rd International Symposium on Wearable Computers (ISWC '99),* San Fransisco, 1999.

[41] Newmann N.J. and Clark A.F., "An intelligent User Interface Framework for Ubiquitous mobile computing". *Proceedings of CHI '99*, 1999.

[42] Orr R.J. and Abowd G.D., "The Smart Floor: A Mechanism for Natural User Identification and Tracking". The Hague, Netherlands, 2000.

[43] Pascoe J., "Adding generic Contextual Capabilities to Wearable Computers". *2nd International Symposium on Wearable Computers*, 1998, http://www.computer.org/proceedings/iswc/9074/90740092abs.htm?SMSESSION=NO.

[44] Ryan N., Pascoe J., and Morse D., "Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant". *Computer Applications in Archaeology*, 1997, http://www.cs.ukc.ac.uk/projects/mobicomp/Fieldwork/Papers/CAA97/ERFldwk.html.

[45] Sawhney N. and Schmandt C., "Nomadic Radio: Speech & Audio Interaction for Contextual Messaging in Nomadic Environments". *ACM Transactions on Computer Human Interaction (ToCHI), Human Computer Interaction and Mobile Systems*, 7, No. 3, pp. 353-383, 2000.

[46] Schilit B., *System architecture for context-aware mobile computing*. Thesis for the degree of Columbia University, New York., 1995, http://www.fxpal.com/people/schilit/schilit-thesis.pdf.

[47] Schilit B., Adams N., and Want R., "Context-Aware Computing Applications". *IEEE Workshop on Mobile Computing Systems and Applications*, 1994, http://www.fxpal.com/people/schilit/wmc-94-schilit.pdf.

[48] Schilit B. and Theimer M., "Disseminating Active Map Information to Mobile Hosts". *IEEE Network*, 8, No. 5, pp. 22-32, 1994.

[49] Schilit B., Want R., Adams N., Gold R., Petersen K., Goldberg D., Ellis J.R., and Weiser M., "An overview of the PARCTAB ubiquitous computing experiment.". *IEEE Personal Communications*, 1995, http://www.fxpal.com/people/schilit/parctab-pcs-jan96.pdf.

[50] Schmidt A., Aidoo K.A., Takaluoma A., Tuomela U., Laerhoven K.van, and Velde W.van de, "Advanced Interaction in Context". *Handheld and Ubiquitous Computing,* Gellersen, H.-W. (ed.), (Springer-Verlag Heidelberg, 1999), pp. 89-101. Lecture Notes in Computer Science No. 1707

[51] Schmidt A., Beigl M., and Gellersen H.-W., "There is more to Context than Location". *Computers & Graphics Journal*, 23, No. 6, pp. 893-902, 1999.

[52] Simons D. and Reuzel J., *WWICE Software architecture - only available within Philips Research*, CRE 1999, 1999.

[53] Thad S., Mann S., Rhodes B., Levine J., Healey J., Kirsch D., Picard R., and Pentland A., "Augmented Reality through Wearable Computing". *Augmented Reality through Wearable Computing*, pp. 386-398, 1997, http://citeseer.nj.nec.com/starner97augmented.html.

[54] Want R., Hopper A., Falcão V., and Gibbons J., "The Active Badge Location System". *ACM Transactions on Information Systems*, 10, No. 1, pp. 91-102, 1992.

[55] Want R., Schilit B., Adams N., Gold R., Petersen K., Goldberg D., Ellis J.R., and Weiser M., "The PARCTAB Ubiquitous Computing Experiment". *Mobile Computing*, 1996, http://www.fxpal.com/people/schilit/csl9501.pdf.

[56] Winograd T, Interaction spaces for 21st century computing. 2001, http://hci.stanford.edu/~winograd/papers/21st/ .

[57] Yan H. and Selker T., "Context-aware office assistant". *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, ACM Press, 2000, http://lieber.www.media.mit.edu/people/lieber/IUI/Yan/Yan.pdf.

[58] Yang J., Yang W., Denecke M., and Waibel A., "Smart sight: a tourist assistant system". *3rd International Symposium on Wearable Computers*, San Francisco, California, 1999, http://citeseer.nj.nec.com/240978.html.

Additional reading

[59] Kuypers G., *ABC van een onderzoekopzet*, (Coutinho, Muiderberg, 1986).

[60] Verschuren P. J. M., *De probleemstelling van een onderzoek*, (Het Spectrum, Utrecht, 1992).