University of Bath

University of Bath

PHD

A context model, design tool and architecture for context-aware systems designs

Kaenampornpan, Manasawee

Award date:
2009

Awarding institution:
University of Bath

[Link to publication](#)

A CONTEXT MODEL, DESIGN TOOL AND ARCHITECTURE FOR CONTEXT-AWARE SYSTEMS DESIGN

Manasawee Kaenampornpan

A thesis submitted for the degree of Doctor of Philosophy

University of Bath

Department of Computer Science

2009

COPYRIGHT

Abstract

The concept of context awareness is widely used in mobile and ubiquitous computing to reduce explicit user input and customization through increased use of implicit input. This requires that the systems take account of *context* in order to infer the user's objective and relevant environmental features. In order to develop systems that support the user in an automatic and appropriate manner, a design process which provides understanding about context and its use is required. Further, an implementation architecture is required which benefits from this design process and supports both system implementers in realising the designs and users in refining and, where necessary, correcting the context sensing and modelling processes at run time.

The aim of this dissertation is to introduce a uniform and systematic, by which we mean consistent and structured, context model and design tool to the design and implementation of context-aware systems. The context model helps to bridge the gaps between designers, developers and users to support shared understandings about context, and it presents a structured understanding of what is taken into account as context. The context model presents a design tool that provides systematic steps and instructions for designing the context-aware system to meet user requirements. It guides the designers to make consistent design choices to meet user requirements rather than adopting a more technology-driven approach.

This dissertation provides 3 main contributions. The first contribution is to introduce a systematic context model based on Activity Theory. The context model describes a uniform set of context elements and relationships between them. We explain why Activity Theory is chosen to help model context. The concept of adding a temporal dimension to extend Activity Theory is proposed. Based on our extension to Activity Theory, the second main contribution is to develop a design tool. Our context model and design tool can be used to model and represent context, evaluate the potential of context resources, indicate situations where a context-aware system is feasible to support a user, and guide the designer in providing functions to support a user without taking control away from the user. In order to support the functionalities that our design too introduces to the context-aware system, as our third main contribution, we present a three layered architecture. In this dissertation, we provide a demonstration of how design choices can be explored, supporting flexible reuse of well structured and discrete context resources, elements and reasoning process.

The use of the context model, design tool and architecture is demonstrated in different scenarios. First the context model and design tool are applied to a simple conference scenario and an initial scenario based on an ethnographic study of the A&E department in a London hospital. The resulting designs for both scenarios are developed in a context-aware system architecture, where the context model and its associated design process are applied to generate design and implementation recommendations. Finally, a prototype architecture is implemented using Java and XML based on the design for patient check-in and check-out scenarios in the hospital A&E department.

Table of Contents

x

List of Figures

List of Tables

Acknowledgments

First and foremost I would like to thank my parents and especially my sister Pornpan Kaenampornpan for providing me with moral support throughout my studies. Thanks for being patient and listening to my complaints through your long distance phone calls everyday. I also wish to thank my supervisor, Eamonn O'Neill for all his support, corrections, and ideas. Thank you for spending your valuable time with me and being the best supervisor I could have asked for. Your perfectionism helps me push myself to get closer to your standard. Furthermore, I wish to thank Nicholas Gorman and his family for helping me get through the difficult times and making me part of their family. Thank you for your warm welcome and making me less homesick. Special thanks are due to Dawn Woodgate for her ethnographic studies. I wish to thank my academic family, Peter Johnson, Hilary Johnson, Leon Watts, Vassilis Kostakos, Andy Warr, Peter Wild, Rachid Hourizi, Iya Solodilova, Anne Bruseberg and Stavros Garzonis. Your help and support has been most appreciated. Finally, I am grateful to the Thai Government for providing me with funding during my studies.

Chapter 1

Introduction

This chapter provides an introduction to mobile and ubiquitous computing and briefly discusses how context awareness has developed as a research area. (Context awareness is discussed in more detail in Chapter 2.) This leads to a discussion of current issues in the context awareness research field. From these issues, the research contributions of this dissertation are summarised at the end of this chapter.

1.1   *Mobile and Ubiquitous Computing*

The majority of computing in recent years has been concerned with *desktop computing*. This is where a computing device is sited at a fixed location, is fairly large and difficult to move around. In order to use such a computing device, the user is required to go to the location where the desktop computer is situated. Typically, the same user uses the same computing device at the same place most of the time, resulting in the working environment of the user and the computing device remaining largely unchanged much of the time. Much of the information

about the user, device and their environment is therefore relatively easily predictable as substantial changes typically happen slowly over time.

Mobile and ubiquitous computing [Weiser, 1991] is a relatively new type of computing. In this type of computing, computing devices and services are available everywhere in the environment and the computing devices and services can be effectively invisible to the user. What might be viewed as part of the move towards ubiquitous computing is the increasing popularity of "laptop" or "notebook" computers which now outsell desktop computers. However, these devices are still quite bulky, hard to use on the move and are effectively just physically smaller desktop computers. The move towards ubiquitous computing includes, amongst their developments, the development of new form factors and interaction techniques beyond the desktop paradigm. For instance, the computing device could be embedded into a user's clothes enabling the user to focus on other things. As these devices are small and/or wearable, the user can carry or wear them wherever she goes. Examples of such devices include: a Personal Digital Assistant (PDA), tablet PCs, smart mobile phones and in-car driver assistance systems. Freed from a fixed location on a desktop, this new type of computing can lead to more rapid changes in information relating to users, devices and the environment. Thus, the context of use is harder to determine, model and predict.

Ubiquitous computing is growing very rapidly. There has been a considerable increase from 27% to 78% in the proportion of households with a mobile phone since 1998-99. In 2004-05, 45% of households in the lowest income group reported owning a mobile phone, compared with 94% in the highest income group [DirectGov, 2005]. Wireless technology, which allows people to roam around small areas while surfing the Web with a laptop, PDA or mobile phone, is gaining popularity in every market around the world. The number of mobile phone users accessing the internet on their handsets is increasing. According to figures announced by the Mobile Data Association (MDA) [Mobile Data Association, 2006], a total of 40.7 million users were recorded as having used their phones for

downloads and browsing the mobile internet in the UK during the third quarter of 2006. The total number of users recorded in July 2006 was 13 million, this increased to 14 million by September 2006. The number of wireless users in 2009 is expected to increase by 77% compared to the 2004 figures [Pyramid Research, 2005]. This large and rapid increase in the number of people using wireless services shows that users are becoming more comfortable and familiar with ubiquitous computing , just as they were with desktop computing. In the near future, it will be increasingly natural for people to use ubiquitous computing in their everyday life.

In ubiquitous computing, users are no longer static and concentrating on one task with one static device. Users are accessing many devices and services such as PDA and mobile phone while they are dealing with multitasking such as find direction on PDA while on the phone and cross the busy road. This raises a new set of questions for researchers in order to improve usability and the user experience. As the number of ubiquitous computing users is increasing, researchers have tried to deal with a new set of human-computer interaction problems. Researchers introduce the concept of context awareness. There are various definitions of context by different researchers and it may be considered broadly as information that has influence on the user in performing an activity. The next section will discuss the concept of context awareness where researchers take advantage of changes in the user's environment to improve the usability of ubiquitous computing.

1.2   *Context Awareness*

In the vision of ubiquitous computing, computing devices and services may be everywhere in the environment. This means that at any time that users need, they can access different services through different types of devices. For example, a user may type in a keyword for what she is looking for on the Internet via a PDA

whilst walking to the nearest shop. Researchers have tried to take advantage of the changing information about users, devices and their environment to improve user interaction by (i) reducing the need for explicit input and (ii) customising the services offered to a user in a given context. From previous example, if user is looking for direction to the nearest shop on her PDA while crossing the road and finding out where she is, the system can reduce the user's explicit input of typing the current address. Instead, the system automatically fills in the current address and shows the direction to the nearest shop for the user. This capability in a computing system is known as context *awareness*.

"The idea behind context awareness is that computational artefacts are enabled to sense the context in which they are being used so that they can adapt their functionality accordingly" [Lueg, 2002]. Context awareness has become a popular topic of research in ubiquitous computing. There are three main reasons for facilitating implicit input rather than, or in addition to, explicit input:

1.   Ubiquitous computing interfaces may be restricted in the interaction functionality offered and their usability. The interfaces to mobile devices have tended to become physically smaller and correspondingly less usable. Even the hype around modern touch screen smart phones cannot hide the fact that conflating user interaction with an increasingly smaller form factor leads to usability problems [Weiser, 1999]. In addition, the interfaces to fixed devices in the environment, such as large public displays [O'Hara et al., 2008; O'Hara et al., 2003], are often by their nature aimed less at individual users and often lack an explicit input device such as a keyboard.

2.   As the available digital services become more transparent and distributed, it becomes difficult for the users to be aware of what devices and services are

available to them at a given place and time as they move through different environments.

3.   To support the user in efficiently carrying out several activities at the same time in a transparent and distributed computing environment. Users may not be concentrating on one task but may be multitasking. For example, a user may be rushing through a crowded space and buying a bus ticket on her mobile phone whilst her mobile device directs her to the bus which is about to leave.

Context awareness takes advantage of technologies that can sense information about a user and her environment. Context awareness processes the sensed information, and typically infers further information, to model the situation of the user. By understanding the situation, it can help the user to become aware of different transparent services and devices in different environments such as available printer in another room. At the same time, it can narrow down the services so that only relevant services are shown to the user in a limited interface at the right time in the right place. For example, instead of showing a town map on the small PDA screen, the system sues the user's current location to rescale the map and only shows the city map that user is situated. Also, the sensed information can be used to reduce the need for the user to explicitly interact with a device, thereby helping a user to efficiently multitask in her everyday life. Users are no longer need to do explicit input of where they are or type of restaurant they want to find. The system automatically uses the user's current location and food preference they provided during system registration to show nearby restaurant on the map on her PDA.

Context awareness has been explored by several researchers in the past but it is still in its infancy. In order to further the field, researchers are exploring several

problems. Current problems in the context awareness field are discussed in the next section.

1.3 *Issues in Context Awareness*

As will be discussed more fully in Chapter 2, the main problems in context awareness can be summarised as follows:

1. The definition of context is broad and still unclear. The boundary of what is and is not context is not properly defined. The question of "What context are we defining?", that we believe is important in understanding context, has not been answered. A clear boundary will guide designers to narrow down the context information to be used in their design of a context-aware system. Context is potentially an infinite set of information so having a boundary helps a designer identify the context information that is necessary to a particular design.

2. In attempting to define context, classifications by different researchers have covered many different aspects of context. While not always in agreement, these findings have shown that there are a large number of elements that make up context. However, the implementation of context-aware applications typically has been technology-driven instead of driven by user requirements. This means the developers design the applications according to the available of the technology such as what types of sensors are available to them at the development stage. Therefore often only a subset of context, for example that can be sensed by a particular technology that the researchers have to hand, is used in the implementations. It can cause difficulties when there are new types of sensor available because the application is not prepare to use other types of information which could improve the efficiency of the context-aware system. Therefore the redesign of the application is required before further research can be done.

3.   As context contains a potentially infinite set of information, the process of gathering the context can be expensive or impossible. Therefore we need to identify and analyse the most influential and critical elements of context that have an influence on human activity in ubiquitous computing. This level of analysis has never been carried out and there is confusion surrounding the various elements. Moreover, there is a lack of understanding of the relationships between the elements.

4.   By having no fully identified context elements and uniform relationships between elements, there is a lack of a context model that could provide a systematic tool to help build shared understandings about context amongst designers, developers and users. Without a systematic tool, context can be too complicated for developers to understand and implement, while users have difficulty in understanding the complex reasoning methods behind the context-aware system. This lack of understanding can lead to breakdowns and frustration when the system makes mistakes. The system is using the changes in information about the user and environment to infer about use's current task and therefore be able to provide support to the user at the right time in order to reduce user's tasks overload. Even highly intelligent, human make mistake in inferring what other is trying to do. Therefore it is possible that the system inferring process can make mistake. By having uniform context model, it hopes to provide consistency in the system and as a result users can build a mental model about the system easier.

5.   Finally, how the context-aware system should use the context has not been dealt with comprehensively. There is no uniform method to process the context in order to infer the user's objective. In other words, designers do not have a design tool to uniformly guide them during the design process. A uniform tool can introduce a uniform reasoning process and data storage model into the context aware system's architecture. Hence consistency in context reasoning can be

implemented. With this architecture, the reasoning methods and context data (which can be very expensive to collate) can then be more easily reused.

We summarise our contributions to addressing these challenges in Section 1.4.

1.4   *Contributions*

Researchers have been developing context-aware applications using whatever technology is available to them. However, it is difficult for researchers to reuse applications that are developed by other researchers since the various applications have been developed without a systematic context model, design tool and process. The context gathering process and reasoning process are driven by the particular technology. Thus, there is little consistency across projects or common understanding of what the context model is. As a result, researchers often have to develop new applications from scratch before they can explore other problems in the context awareness field. The main contribution of the research reported here is therefore to produce a common context model and a systematic design tool for context-aware systems that can offer reusability and support for context-aware system design. Furthermore, based on the context model, a context-aware system architecture is produced to support the functionalities that the design tool introduces.

The context model draws on Activity Theory [Kaptelinin and Nardi, 1997] in representing the context elements and relationships amongst them that may have an influence on users in achieving their objectives. The model is based on information drawn from Table 2-1. The model is used in inferring the user's objectives. Researchers can then follow the model and systematic tool during design and implementation. At run-time, the model underpins the context aware architecture and can be called upon to represent the system's context model and reasoning to the user, also allowing the user to correct and refine the model.

This research is divided into the following three parts:

1.   We provide a common context model that provides a conceptual classification system for context. The context classification system in the context model includes key elements of context that have an influence on a user's activity. Moreover, it also includes consistent relationships between each element in the classification so that these relationships can be represented and exploited during the development of a context-aware system.

2.   We provide a systematic design tool based on the context model. This design tool is intended to help designers analyse situations to decide which types of information have an influence on the user in achieving their objectives. The relationships between context elements in the context model are used to separate the context elements from the reasoning methods. The relationships are also used for designers to communicate with implementers in order to produce uniform reasoning methods to infer and support the user's objectives.

3.   Based on the context model, we provide an architecture that supports the separations between identified context elements and the relationships between these elements in the context model. By having a clear separation between context elements, the architecture simplifies processes such as changing types of sensors to acquire context data. Moreover, as a result of supporting uniform relationships between the elements in the context model, the architecture supports the ability to reuse context information and reasoning methods in different applications and even across different domains.

The dissertation demonstrates how the proposed context model can be used as a design tool. The goal for the context model is to provide a generic yet operationalised understanding of context. The context model can then be used to guide the development of a context-aware system architecture. A prototype implementation is described in the dissertation to demonstrate how the architecture offers flexibility and simplicity in changing or adding new types of sensors, reusing context data for new applications and domains, and communicating the underlying context model and reasoning to the user.

1.5   *Outline of Dissertation*

Chapter 2 presents background in the field of context awareness. Previous context definitions, classifications and context-aware projects are reviewed in order to identify challenges in the field. The chapter is concluded with a discussion of requirements for a design tool and architecture for context-aware systems. Based on these requirements, the research question of this dissertation is proposed. The aims and objectives of the dissertation are discussed in order to elaborate on how we set out to answer our research question.

Chapter 3 presents an introduction to the work of this dissertation. It begins describing Activity Theory and the reasons for using it in this work. It then discusses the significance of context history and how Activity Theory and context history are used in our proposed context model. In our context model, a temporal dimension is added to Activity Theory in order to take account of history. Our context model contains nine elements. The definitions of the nine elements are introduced in order to aid designers to come to a shared understanding about the context model in a consistent and structured manner.

In Chapter 4, the use of the context model as a design tool for context-aware system design is discussed. First, we provide an overview of the six systematic steps that designers should consider when using the context model as a design tool. We discuss how our use of Activity Theory brings a uniformly structured design tool to context-aware system design. We discuss how the proposed context model is intended to meet the context-aware system design tool requirements described in Chapter 2.

Chapter 5 provides an overview of the architecture that supports the designs resulting from applying the context model. It begins with a brief summary of the three layers in the architecture. The structured reasoning mechanism and data storage are introduced. The flow of data in the architecture is then discussed in order to show how the architecture supports the separation of context according to its properties. We discuss how the architecture is intended to meet the context-aware system architecture requirements.

In Chapter 6, two scenarios are introduced in order to apply and demonstrate the design tool based on the proposed context model. The first uses a common conference assistant scenario that has been used in previous context-aware projects [Dey, et al., 2001; Dey, et al., 1999; Sumi and Mase, 2001]. The second example is drawn from a complex scenario in the A&E department of a large London hospital. The design tool is applied to these two scenarios. The results are discussed to evaluate how the use of the design tool has met the requirements developed in Chapter 2.

Chapter 7 applies the design outputs provided by our application of the context model and design tool in Chapter 6 to demonstrate the implementation of a prototype for managing patient admissions in the hospital A&E department scenario. It shows how the design output from Chapter 6 assists the developer

during implementation of the system with its consistent structure of databases for sensors, context elements and context model. The consistent structure of the databases provides well separated layers in the architecture to deal with different levels of context information. As a result, with regard to the architecture described in Chapter 5, the implementation of the three layered architecture is described in order to show the potential advantages it introduces to the context-aware system. Then an application of the design outputs from both scenarios in Chapter 6 is discussed in order to demonstrate the use and advantages (e.g. ease of expansion and reusability) of the architecture. The requirements developed in Chapter 2 are then used to evaluate the architecture by investigating how the architecture actually meets these requirements.

Chapter 8 summarises the work of the dissertation, draws conclusions, and indicates directions for future work.

Chapter 2

Context
Awareness

This chapter introduces context awareness. It starts by discussing the problems that ubiquitous computing introduces to traditional desktop computing users. Previous context-aware definitions and classifications are then discussed. Previous context-aware projects are also analysed, highlighting their similarities and differences. The analysis of previous context-aware systems also leads to discussion of the problems that need to be tackled in order to further the field of context awareness. Lastly, a problem solving idea is presented and a research question is raised.

2.1   *Usability Issues in Mobile and Ubiquitous Computing*

As technology develops, the use of computing devices is no longer limited to a single location as in traditional desktop computing. Ubiquitous computing allows users to carry a device with them at all times. A user can have access to information anywhere via different devices or services that are embedded in the environment. There is therefore the possibility of a user having to concentrate on several activities at the same time. Moreover, the interfaces are in many cases

becoming less usable. There are at least two sources of usability problems associated with ubiquitous computing applications.

First, mobile and ubiquitous users access information and services in diverse settings via different devices that are mobile or fixed in the environment and whilst performing other activities. This multitasking in changing environments puts increased cognitive demands on the user. While some research [Schumacher, et al., 2001] suggests that users may become skilled at managing some of these demands, and recent studies show that users can successfully perform relatively simple multitasking, such as running through city streets while avoiding obstacles and glancing intermittently at information on a PDA [Benford, et al., 2003; Flintham, et al., 2003; Jameson and Klöckner, 2005], more cognitively demanding multitasking remains a problem [McCrickard, et al., 2003; Oviatt, et al., 2004.] In particular, usability is likely to suffer when interactive tasks involve explicit input from the user [Oviatt *et al.*, 2004]. Explicit input is input where the user tells the computer directly (e.g. by command-line, direct manipulation using a GUI, gesture or speech input) what he expects the computer to do, whereas implicit input is an action performed by the user that is not primarily aimed at interacting with a computer system but which such a system understands as input [Schmidt, 2000.]. An example of the implicit input is information about accessing a room or objects when user is opening the door or picking up the objects that are embedded with sensors [Antifakos, et al., 2003].

Secondly, in ubiquitous computing , usability is often hindered by the conflation of the physical characteristics of the device with the characteristics of the interface between the user and the services that the device delivers [Kostakos and O'Neill, 2003; O'Neill, et al., 2006]. For example, as mobile devices become smaller, their input and output features become smaller and less usable. At the other end of the size spectrum, fixed ubiquitous devices such as large public displays driven by

14

embedded computers typically do not have the keyboard and mouse that support explicit user input in the desktop environment. Researchers have explored new techniques of interacting with ubiquitous devices such as gesture or speech input [Minker, et al., 2005; O'Neill *et al.*, 2006]. Unfortunately, we have not yet developed interaction devices and techniques for such settings that are as effective for explicit input as those in use in the standard desktop setting. It therefore becomes harder for ubiquitous computing users to perform explicit input compared to desktop users. Researchers have attempted to improve user interaction by taking advantage of the changes in information relating to users, devices and environments. This concept is known as *context awareness*. Context awareness may be exploited to overcome the usability challenges of explicit input. The goal of this research is to use context to improve usability in ubiquitous computing by reducing the requirement for explicit input. This may be achieved by increasing the use of implicit input. The reduction in explicit input that users have to perform should improve usability both by reducing the user's cognitive load and by reducing the user's reliance on poorly usable interaction techniques and devices, thereby addressing both of the sources of usability problems described above.

Previous research in context awareness is discussed in the next section. It presents previous context definitions and classifications proposed by different researchers. The analysis of previous context-aware projects is then discussed.

2.2   *Previous Research in Context Awareness*

A large number of researchers have explored the field of context awareness in the past few years. Early works [Abowd, et al., 1996; Brown, 1996; Schilit and Theimer, 1994] considered context to be related to the location of users. Technology has developed rapidly in the area of computing and sensing devices. This means that devices may soon be placed in more and more locations in the environment, sensing vast amounts of increasingly diverse information.

Researchers hope to be able to make use of this sensed information through context awareness to improve the usability of ubiquitous computing. Researchers have attempted to define context in order to have a general view on the diversity of context information.

2.2.1 Context Definition

A number of definitions of context awareness have been developed for various applications. Researchers have defined context to better understand the theories behind their implementations. Some of these different definitions are presented and discussed here.

The first set of definitions offers a very broad definition of context. For example, Capra et al. [Capra, et al., 2001] defined context as "everything that can influence the behaviour of an application". Lieberman and Selker similarly considered context to be "everything that affects the computation". However, they specified that explicit input and output are not considered as part of context [Lieberman and Selker, 2000]. These definitions are too vague to be used as theory behind an implementation as it is very hard to define for implementation purposes what exactly "everything" refers to.

The second set of definitions attempts to define context more precisely. For example, Chen and Kotz [Chen and Kotz, 2000] provide a definition where context is a set of environmental states and settings that are of interest to the user or ones that trigger application events. Similarly, Benerecetti, Bouquest and Bonifacio argue that context can be thought of as a subjective representation of the environment that an agent uses to solve a particular problem [Benerecetti, et al., 2001]. The context, in this case, is not all the states and settings but is limited to ones that are of interest to the user or to solve a particular problem. Although these definitions have attempted to define context more precisely, it is still unclear exactly what the states of a particular environment actually are.

The last set of definitions of context is again more precise but is not limited to the environmental states and settings that are of immediate interest to the user. "Ward, Jones and Hopper defined context as a state of the computer or application's surroundings" [Hopper, et al., 1997]. Ryan, Pascoe and Morse [Morse, et al., 1997] similarly defined context as the information about a computer's environment. Schilit and Theimer [Schilit and Theimer, 1994] defined context as information about the world around the users. Schmidt, et al [Schmidt, et al., 1999a] define context as more than just a state of either the application's surroundings or world around the users. They defined context to be knowledge about both the user's and device's state, including their surroundings, situation and to a lesser extent, location. Here they have specified that context is not everything that influences the application, but can be grouped into knowledge of both the user's and the device's state. More specifically, Schilit, Adams and Want [Schilit, et al., 1994] defined context as the user's physical and computing environment that is changing over time. Dey and Abowd [Dey and Abowd, 2001] provide a similar definition but cover more than just a user's and device's state. "They defined context as any information that can be used to characterise the situation of an entity where 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 the application themselves." [Dey and Abowd, 2001]

Bucur [Bucur, et al., 2005] attempted to extend the definition from Dey and Abowd by defining context as the factors that influence a certain decision. The context may therefore be described as a set of attributes and finality. The finality is the goal for which the context is used at a given moment, the focus of the activity at hand.

Although these definitions attempt to provide further detail, the boundaries of context and the relationships between the user, device and environment are still unclear. From different definitions, the root question that should be raised is: "*What context are we defining?*" To date, nobody has answered this fully and this has lead to vagueness in the definition of context. To design or develop a context-aware system based on these unclear definitions is difficult.

Researchers have tended to use context definitions to give them a general idea about context. Based on such definitions, some researchers have derived classifications of context for a context management perspective.

2.2.2 Context Classification

Before researchers use context in applications, they need to have an understanding of what they should take into account as context. The scope of context is potentially infinite, encompassing everything that may in one way or another influence the user. Clearly, a way of reducing this infinite set to something more manageable is needed. A first step is to identify the elements of context that are likely to be most relevant to the user's needs and actions. (The ambiguity in this claim illustrates the need for adaptability of our definition and representation of context at each stage from analysis, through design and implementation, to use.) Having reduced the set of elements of context that we must consider, to complete a model of context we need to capture the relationships amongst these elements.

Several researchers have tried to develop better understandings of context by producing context definitions and classifications of the key elements of context.

Table 2-1 summarises this work. The columns in Table 2-1 are derived from elements that researchers have identified as relevant parts of context. In the first

row of Table 2-1, Benerecetti, Bouquest and Bonifacio [Benerecetti *et al.*, 2001] have classified context into Physical Context and Cultural Context. Physical Context is a set of features of the environment while Cultural Context includes user information, the social environment and beliefs.

Schilit et al [Schilit and Theimer, 1994] similarly have included Physical Context and Cultural Context, which is called the User Environment. However, Schilit et al have paid attention to the Computing Environment as well.

Schmidt et al [Schmidt, et al., 1999b] on the other hand have extended the classification into three dimensions: Physical Environment, Human Factors and Time. Human Factors cover the same features as Cultural Context. Physical Environment combines Physical Context and Computing Environment. They have added time to reflect the importance of context history, which has an influence on modelling the user's past, current, and future actions.

Lieberman and Selker [Lieberman and Selker, 2000] have ignored Time and classified context to include the Physical Environment, the User Environment and the Computing Environment. In this case, the User Environment includes the user's location and is treated separately from the Physical Environment. Lieberman and Selker treat the Computing Environment as a separate entity here because they believe that information such as network availability can be of interest to the user and related computing devices. Hull et al [Hull, et al., 1997], Lucas [Lucas, 2001] and Chalmers and Sloman [Chalmers and Sloman, 1999] argue that characteristics of the device itself, such as screen size and input device, are also of interest to the user and system. They have therefore included Device Characteristics as one element of their context classification. Chalmers and Sloman have also added user activity into their context classification. However,

they do not consider Time and other user characteristics, which may be important elements of context.

Based on Dey and Abowd's definition of context [Dey and Abowd, 1999], they have provided a top-level classification system which includes four types of context: Location, Identity, Time and Activity. They claim that these are primary types of context that can be used to refer to other secondary context. Becker and Nicklas [Becker and Nicklas, 2004] used the concept of Identity from Dey and Abowd's classification of context [Dey and Abowd, 1999]. They divided context information into three criteria: the Identity of the entities, the Location of entities and Time. Because of the important roles of identity, location and time to the organisation of context models, they refer to these as primary context. Lee and Meier [Lee and Meier, 2007] extended Becker and Nicklas' classification by including Quality of Service context in the primary context. However, with these three classifications, there is no clear separation between device and user. The computing device and user should be treated differently because they have different features and they affect user behaviour differently.

Similar to Dey and Abowd, Korpipää et al [Korpipää, et al., 2003] provided a top-level classification system with categories including Location, Time, Environment, User and Device. This provides a clearer separation between User and Device compared to Dey and Abowd's classification. Korpipää et al separated Location from Environment and defined User Activity as a subcategory of User.

Thomson et al. [Thomson, et al., 2005] present a classification including Location, Tools, Time and People. The People context includes information about the user, her actions, other people around the user and their social relationships. The Tools context includes information about Device Characteristics and the Computing Environment. The representation of People and Tools illustrates that

this classification has clear separation between User and Device similar to the classification by Korpipää et al. On the other hand, it has combined information about Location and Environment in Korpipää et al. into Location context. Similar to both Dey and Abowd and Korpipää et al., it includes Time as part of context.

Oh et al [Oh, et al., 2006] use the well-known 4W1H concepts of knowledge representation to classify preliminary contexts into 5 types: Who, What, Where, When, and How.

Dix et al. [Dix, et al., 2000] have classified context into 4 types by considering the nature of the context in which interaction with mobile and ubiquitous applications takes place. First, Infrastructure Context is concerned with information such as variability of service, user awareness of service and "liveness" of data. Secondly, System Context deals with information about other devices, applications, and users. Thirdly, Domain Context is concerned with information on application domain, style of use and identification of user. Lastly, Physical Context is concerned with the physical nature of the device, environment and location.

These classification systems are typically intended to be context models defining what elements of context should be used to understand the user, in order to have a better understanding of the user's interactions and intentions. Chen and Kotz [Chen and Kotz, 2000] have introduced a classification system with a different aim, where context is classified depending on how it is used in the application. They have classified context very broadly into two types: Active and Passive, where Active Context is that which influences the behaviours of an application, and Passive Context is that which is relevant but not critical to an application.

	Location	Conditions	Infrastructure (Computing Environment)	Information on User	Social	User Activity	Time	Device Characteristics
[Benerecetti et al.'01]	Physical Environment			Cultural Context				
[Schilit et al. '94]	Physical Environment		X	User Environment				
[Schmidt et al.'99]	Physical Environment			Human Factor			X	
[Lieberman and Selker'00]	User Environment	Physical Environment	X	User Environment				
[Hull et al.'97]		Physical Environment		X				X
[Chalmers and Sloman'99]	X		X		X	X		X
[Lucas'01]	Physical Environment		Information Context					X
[Abowd and Dey'99]	X			Identity		X	X	Identity
[Korpipaa et al.'03]	X	Environment		User		Subset of Information on User	X	Device
[Becker and Nicklas'04]	X			Identity			X	Identity
[Lee and Meier' 07]	X		Quality of Service	Identity			X	Identity
[Oh et al.'04]	Where		What	Who	What	How	When	What
[Thomson et al.'05]	Location		Tools	People			X	Tools
[Dix et al. '00]	Physical		Infrastructure	Domain/System				System
[Chen and Kotz'00]	Active/Passive							

Table 2-1 Context classification systems.

From Table 2-1 apart from Chen and Kotz's classification, we can see that each approach covers different elements of context for understanding human behaviour. Some groups in different classification systems are overlapping. Moreover, some groups cover the same elements but are labelled differently such as *Cultural Context* in [Benerecetti *et al.*, 2001] and *User Environment* in [Schilit and Theimer, 1994]. Together these groups cover key elements of context that have an influence on user behaviour. From these classification systems, we identify 5 high

level categories of elements that should be taken into account in modelling context for design. These 5 high level categories are:

User. This is information about the user (for example, identification, habit, and preference) and the user's current actions.

Physical Environment. This is information about the physical environment such as the physical location of user and devices and condition of the environment (for example level of noise, light, traffic etc.). It is separate from the computing environment because it is different in its features, and the ways in which these features are captured and reasoned about will be different.

Tools. This groups all information about the tools, including both non-computing tools and the computing environment such as notice board, network availability, printer queue status etc.

Social. This is separate from **User** because it represents information about the relationship between a user and other users that will be captured and processed differently from information about the user himself.

Time. This is time-based information such as time of day, date etc.

It is important that the context classification represents key elements as it will be used during implementation. When the context classification is not complete or too complicated, the developers may have difficulty in implementing the system [Paganelli and Giuli, 2007]. Moreover, researchers may face difficulties in reusing and expanding the system.

2.2.3   **Previous Context-Awareness Projects**

In this section we group previous context-awareness projects according to the approach they took to modelling context.

2.2.3.1   **Location Based Systems**

As Location Based Systems deal with only one particular type of context (i.e. location) and only with one or a couple of sensors, system architecture can therefore involve simple direct sensor access. The designers mainly concentrate on how to acquire or gather the sensor data, represent the sensor data and how to improve the accuracy of data from the sensors. The context model, which is used in these systems, only deals with one type of context (i.e. location). It can however gather information from different types of sensor such as GPS, Active Badges, Active Bats, Smart Floor, etc.

For example, **Location-Aware Web System** (LAWS) [Haghighat, et al., 2004] allows users to see web pages on their roaming device's interface that are dynamically generated based on their location from their own in-door positioning system. So the users know where they are in the physical space and are able to locate items or places of interest that they are looking for, either through a map that is shown on the roaming device or through a reference point to the item's location. The positioning system represents location in the form of X-Y coordinates.

Sotto Voce electronic guidebook [Aoki, et al., 2002] provides content about exhibits on a user's device according to a user's location (The user can click on the photo of the item in the room on his device to obtain more information on it).

ImogI system [Luyten and Coninx, 2004] uses Bluetooth to establish communication between the PDAs and the exhibits and reflects the closest exhibits to the location of the user. **Active Map** [Schilit and Theimer, 1994] detects a user's current location via active badges and shows it on the map so that it allows users to be located quickly.

Location-aware city guides [Davies, et al., 2001] use location information from GPS or network-based location beacons to present information relevant to a user's location and provide route guidance.

SmartCampus Location-Aware Community System [Kim, et al., 2007] uses WiFi access points to determine the location of users. It runs applications that link "people-to-people-to-place", or P3-systems. For instance, the applications allow a user to see the location of her 'buddies'. However, by using just location, it limits the functionality of the applications. The users are therefore left with some concerns such as privacy control, the validity of the data (e.g., will applications be used to make verbal attacks on others?), and interruptions or overload with information, which may be disruptive.

Most Location Based Systems were designed to be used for a particular scenario, as they concentrate on a technology and its capability to get one type of context. Therefore this type of system typically does not separate the sensor code from system code. Thus it is impossible for researchers to reuse the systems with different types of sensors or domains. Moreover, by limiting the context to just one type of information, it may also limit the functionality of context-aware systems.

Researchers have explored different types of context beyond location in order to improve the functionality of the context-aware system. These context-aware systems are discussed in the next section.

2.2.3.2   Context Aware Systems

Location Based Systems do not take account of other information about the user or her environment. For example, Sotto Voce's electronic guidebook does not take into account whether the user is with a companion or not. As a result, visitors frequently complain that audio tours with headphones isolate them from their companions, and visitors have few opportunities to interact effectively with each other while an audio tour is played to them.

Instead of using one type of context, Context Aware Systems combine different types of context (e.g. location, user's environment, society and time) in order to improve the understanding of a user's current task or objective [Baldauf, et al., 2006]. This increases the ability to adapt to the user's needs and become a more useful and usable system. However, it can only be used for a particular scenario and particular types of sensor. These systems do not support other applications and the sensors cannot be reused.

For example, **SenSay** [Siewiorek, 2003] is a mobile phone that adapts to changing user states by manipulating ring volume, vibration, and phone alerts for incoming calls according to context information. Context information such as the user's activity and the user's environment is collated from a number of wearable sensors including accelerometers, light and microphones mounted on the user's body.

26

SmartRestaurant [Lukkari, et al., 2004] is a web service for mobile users that has been designed to enhance a restaurant's production and delivery process. The SmartRestaurant actors are categorised into customers and employees.

The customers (also referred to as end-users) are normal customers except that they use the SmartRestaurant to order and pay for their lunch before they reach the restaurant. SmartRestaurant takes account of customers' current context (time, location) to schedule the delivery time for their order so that the food will be hot and fresh when they enter the restaurant. The employees of the restaurant configure the service and prepare the ordered meals. SmartRestaurant allows the restaurant to automatically adjust sales in line with production capacity (a maximum of 10 orders can be sold per delivery period of 15 minutes). SmartRestaurant also provides the restaurant with prior knowledge of upcoming orders and reduces the time consuming process of completing payment.

Ubiquitous Multimedia Information Delivering Service (U-MIDS) for smart homes [Hsu, et al., 2007] uses Radio Frequency Identification (RFID) to detect user's locations and behaviours. U-MIDS uses this information together with users' preferences to automatically deliver multimedia information, such as MP3 music, Internet radio, spoken online news and personal spoken messages to the users in a smart home. The U-MIDS gateway can control the network media players to play desired spoken information or music according to users' preferences, locations and situations. The users can therefore be free and relaxed to gather the ubiquitous multimedia information around their home all the time.

Chalmers et al [Chalmers, et al., 2004] introduce a **framework for contextual mediation** concentrated on managed system resources. The context elements that they take into account include the computing context and user context such as screen size, network type and user's current task. A context-aware map

application is used as an example. Aspects of the contexts are used to select the most appropriate profiles which specify the required mediation rather than trying to cater to all possible context variations. The use of context as arguments and the ability to compose sub-profiles give some flexibility.

Chisel [Keeney and Cahill, 2003] is an open framework for dynamic adaptation of services in a context-aware manner based on a policy-driven approach. Chisel adapts the behaviours of service according to the changes of environment, user context and application context. The adaptation is driven by a human-readable declarative adaptation policy script.

Another example is the **context-aware mobile communication in hospitals** [Muñoz, et al., 2003]. It is comprised of context information in hospitals, which includes location of a worker, device or artefact state, time and person's role, and allows users to send messages and access hospital services when and where they choose. The system extends the instant messaging paradigm to add context awareness as part of the message. By using this system, users can utilise their own personal device to write messages that set circumstances when the message should be sent. For example, the sender can ask that a patient's lab results be delivered to the first doctor to enter room 124 after 9am. The system architecture consists of a context-aware client, an instant messaging server and several agents. Each agent contains three modules: 1. Perception module gathers information sources (sensors, users, other agents, the server) 2. Reasoning module governs the agent's action 3. Action module triggers a user specified event. All messages between agents are XML encoded.

Previous Context Aware Systems projects have advanced the field of context awareness. However, the often monolithic systems developed typically do not

lend themselves to reuse for different situations or sensors. Context Aware Frameworks provide attempts at a more abstract approach.

2.2.3.3   Context Aware Frameworks

Even though the Context Aware Systems can be optimised for the situations they are used in, they do not have to be flexible and extensible. In order to ease the development of context-aware applications, an abstract framework is needed. The framework provides a generic infrastructure that not only provides the client with access to retrieve context data, but also permits the simple registration of new distributed heterogeneous data sources [Baldauf *et al.*, 2006]. This means the researchers do not have to invest time and resources to repeatedly develop new Context Aware Systems. Examples of past Context Aware Frameworks are discussed below:

Gaia [Román, et al., 2002]

Gaia extends typical operating system concepts to include context awareness. Its aim is to support the development and execution of portable applications for active spaces. Gaia is a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space. Figure 2-1 shows the three major building blocks of Gaia:

- Gaia Kernel contains a management and deployment system for distributed objects and an interrelated set of basic services that are used by all applications. The *Component Management Core* dynamically loads, unloads, transfers, creates, and destroys all the components and applications of Gaia. Gaia's five basic services are:

  o *Event manager service* is responsible for event distribution in the active space and implements a decoupled communication model based on suppliers, consumers and channels.

o   *Presence service* is responsible for detecting digital (e.g. service and application) and physical entities (e.g. furniture and people) present in an active space. It defines four basic types of entities: Application, Service, Device, and Person.

o   *Context service* helps the applications to query and register for particular context information and high level context objects.

o   *Space repository service* stores information about all software and hardware entities contained in the space (e.g., name, type, and owner) and provides functionality to browse and retrieve entities based on specific attributes.

o   *Context file system* makes personal storage automatically available in the user's present location. It constructs a virtual directory hierarchy to represent context as directories where path components represent context types and values.

The Gaia Application Framework provides mechanisms to construct or run applications or to adapt existing applications to active spaces. The framework is composed of a distributed component-based infrastructure, a mapping mechanism, and a group of policies to customise different aspects of the applications. *The Applications* are the applications available in an active space.
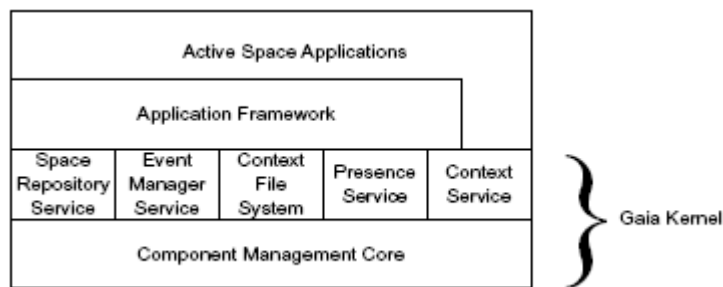
Figure 2-1 Gaia system architecture

The context model in Gaia is represented in a 4-ary predicate which is based on first order logic and Boolean algebra An atomic context predicate is defined in the following way: Context(*<ContextType>*, *<Subject>*, *<Relater>*, *<Object>*).   It is written in DAML+OIL [Connolly, et al., 2001]. The *Context Type* refers to the type of context the predicate is describing, the *Subject* is the person, place or thing with which the context is concerned, and the *Object* is a value associated with the Subject.   The *Relater* relates the Subject and the Object, using a comparison operator (=, >, or <), a verb, or preposition. These rules may be a combination of lower level context information. This model provides a simple way to write a predefined rule about context. It is, however, very specific for different situations and can be difficult to reuse or extend. The implementation of each application requires subscribing for different context information and high level context objects.   There is no consistency in context model between applications.   To develop a subscription part for every new application can be a time consuming process in itself.   Moreover, by subscribing a combination of lower level of information can lead to the difficulties in reusing the context model. For example in different application where the same sensor is not available or new type of sensor is introduced to the system. Then the context model (rules about context) has to be changed for each set of rule.

CASS (Context-Awareness Sub-Structure) [Fahy and Clarke, 2004]

CASS is centralised server based middleware intended to support context-aware applications on hand-held and other small mobile computers. Figure 2-2 illustrates that the middleware contains:

- *Interpreter*

- *Context retriever* is responsible for retrieving stored context data. It may use services of an interpreter.

- *Rule engine* has 3 subclasses that correspond to the categories of context awareness application features identified in [Dey and Abowd, 1999].

- *SensorListener* listens for updates from sensors which are located on distributed computers called sensor nodes. It may then use the services of an interpreter before storing the gathered data in the database.

An inference engine works in conjunction with a knowledge base and uses the rules contained in the knowledge base to solve problems. The rules are stored in a database separate from the interpreter. The components are therefore not required to recompile when the rules change.
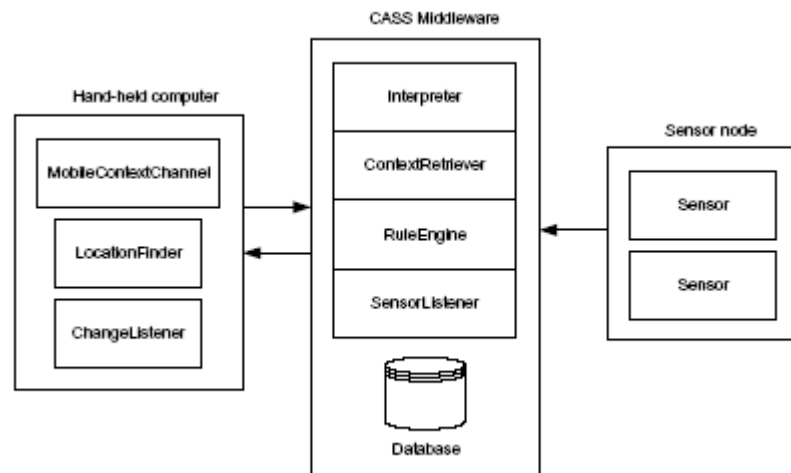
Figure 2-2 CASS system architecture

Rain	Brightness	Temp	Goal
wet	dull	cold	indoor

Table 2-2 Example of rule database entry

Table 2-2 shows a weather state for a tour-guide application. It might use such a rule to allow it to display hyperlinks to indoor activities or can be used in a further rule. There is no standard structured way in creating the rules. Normally, the rules are specific to a particular domain which makes that extension and reuse in different domains very difficult.

Middleware Enabling Context-awareness for Smart Environment (MidCASE) [Bai, et al., 2007]

Similar to CASS, MidCASE is based on a layered middleware. Its architecture aims to provide a service oriented middleware to bridge the gap between the programmable application layer consisting of different scenarios and the hardware layer consisting of heterogeneous devices. In this process, the middleware utilises a service-oriented, distributed, extensible architecture to achieve the service in each awareness service domain. The services are deployed in accordance with the form of "One scenario, one service, one reasoning and awareness process". The awareness process is achieved by applying rule-based reasoning. The context model (*Context Tuple Space*) in MidCASE uses combinations of entities to represent the physical world in the domain. For each context service domain, the selection of entities and their attributes, and the selection of methods, are critical in the model building process.

Figure 2-3 shows the architecture comprising of five layers and two cross-layer modules:

- *Hardware Abstract Layer* treats hardware devices as generic common objects to obtain all kinds of context data. This layer makes the sensors transparent to the upper layer.

- *Service Registry Layer* provides the mechanism of registration and realises the communication among services through remote process calls.

- *Context Model Layer* consists of the entity and context containers. The entity container is used to model the environment. Each entity models the object in the real world such as a nurse and a monitor. The status and capability of the objects refer to the attributes and methods of the entities. The context container is used to connect the context data taken from the hardware abstract layer. To facilitate the process of awareness and reasoning, as shown in Figure 2-3, this layer combines context agent and context queue in order to work as a connector to a rule engine. The context agents bridge up the entity in context-awareness service and devices in physical world. The context agent could gather data from different sensor devices where the data becomes part of information about entity. The context agent also could get different accessing objects from variable entities such as nurse entity. In order to model the entity in the real world, the context of entity constitutes a different context queue. The context agents keep accessing the data from sensor devices and compare it with the previous data that has been stored a moment ago. The differences between the two groups of data will originate context event, which means the changing of scenario. The attribute of the relative entities could be changed through context event and be input into rule engine as facts through the context queue, which is loaded into a fact base and rule engine.

- *Awareness and Reason Layer* provides a rule engine which is embedded in the middleware. The fact and rule loaders are provided

in this layer so that the facts and rules in scenarios from entities can be loaded. This layer checks whether the current context of entity "facts" satisfy some rules. It then sends the result of reasoning to the application presentation layer.

- *Application Presentation Layer* shows how to use the result of reasoning in the physical world.

- *Energy Management Module* is implemented with cross-layer cooperation. Combining the functions of module and rule engine, it can control network energy assumptions by assuring normal running on the fewest required nodes.

- *Security Module* refers to the hardware authentication of context acquiring and access priority control of context data.
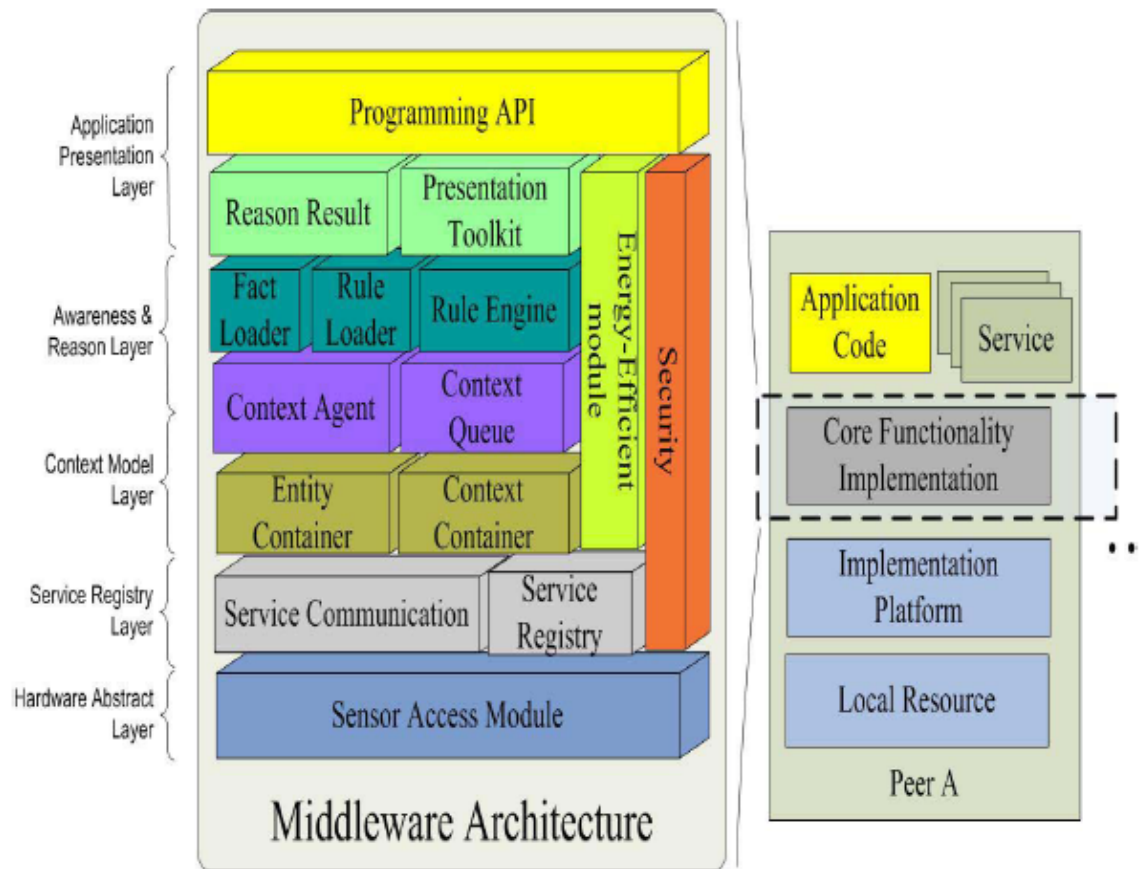
Figure 2-3 Architecture of MidCASE

Context Toolkit [Dey, 2000; Dey *et al.*, 2001]

This Toolkit was one of the first projects that considered separating the acquisition and representation of context from the delivery and reaction to context, facilitating easier building of the context aware application. As shown in Figure 2-4, the components in the Context Toolkit architecture are:

- ▪ *Widgets* send particular context attribute information to subscribers and store them in MySQL. For example, a Presence Widget that senses the presence of people in a room. or a Meeting Widget that detects new meeting information either from a user's schedule or built on top of a Presence Widget which would show that there are

two or more people in the room. The context widgets separate the applications from sensors.

- *Interpreters* convert data to meaningful or useful information. Interpreters help the process of raising the level of abstraction of a piece of context. For example, location may be expressed at a low level of abstraction, such as geographical coordinates or at higher levels such as street names. An example of combining data is as follows: if a room contains several occupants and the sound level in the room is high; one can guess that a meeting is going on by combining these two pieces of context. The interpreters hide the context translation process from the applications. Therefore they can be reusable by multiple applications.

- *Aggregators* gather logically related information about a context entity that is relevant for applications and make it available within a single piece of software. For example, Attendee Aggregator is used to collect information about a user such as location from Presence Widget and a user's note from Memo Widget.

- *Discoverers* are responsible for maintaining a registry of what capabilities exist in the framework. This includes knowing what widgets, interpreters, aggregators and services are currently available for use by applications.

- *Services* are components in the framework that execute actions on behalf of applications. Examples of services include sending an e-mail to a user or sending a message to a user on a two-way pager containing a number of possible message responses.

The peer to peer architecture with centralised discoverer supports multiple simultaneous applications and querying or storage of context. The Context Toolkit

considers context broadly as information about the relevant entities (people, places, and objects) in the environment.

The context model is represented in simple attribute value tuples which are encoded using XML for transmission. Based on the broad definition of context, the context modelling in this project is domain oriented modelling. The context design only supports context in the same domain. When the domain is changed, the designers have to reconsider the required aggregators, widgets and interpreters, which can be a time consuming process. Even for a new application, the designers have to reconsider the aggregators if the existing ones cannot be reused.
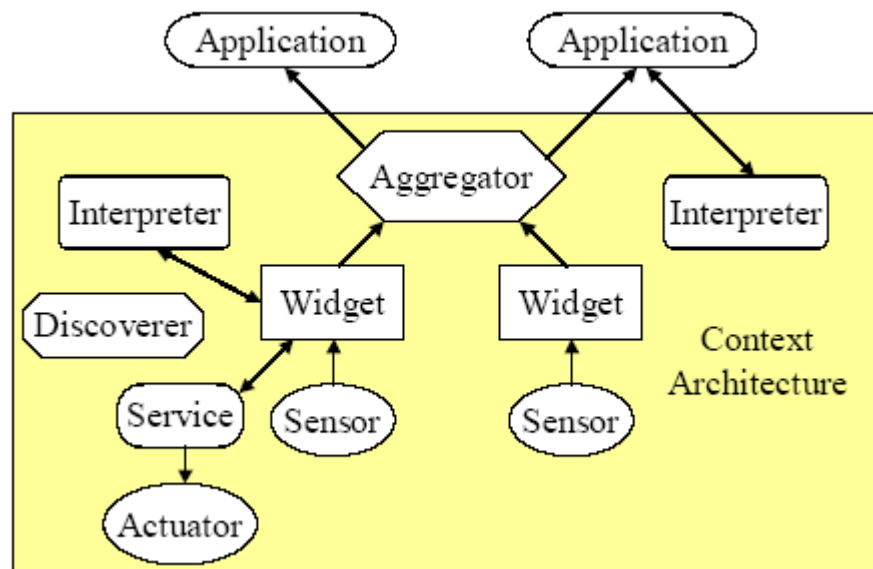
Figure 2-4 Components in Context Toolkit Architecture

Hydrogen [Hofer, et al., 2003]

The Hydrogen framework is based on a layered architecture. Figure 2-5 shows an architecture comprised of three layers. Similar to Context Toolkit, Hydrogen's architecture aims to separate the concerns of interacting with the physical sensors,

storing and maintaining the context, and the application itself. The communication between layers is based on an XML protocol. These layers are:

- *Adaptor Layer* is responsible for getting information from sensors and possibly enriching this information with logical context information. The information is then sent to the Management Layer. This avoids multiple applications reading data from the same sensor.

- *Management Layer* has a *ContextServer*, which stores all contextual information about the current environment of the device, embedded to provide simple methods for the applications to retrieve and subscribe to a context. *ContextServer* provides the possibility of sharing context information with other devices via peer to peer communication. It offers two ways for the applications to refer to context – asynchronous and synchronous methods. The asynchronous method allows the applications to query a specific context from the server in a pull-based manner whereas the synchronous method informs the applications about the changes or the invalidation of the subscribed context.

- *Application Layer* holds context-aware applications. Each application subscribes to a different context via a *ContextClient* or directly via an XML protocol to react to specific context changes reported by the context manager.

Unlike Context Toolkit and many other context frameworks, Hydrogen introduces an architecture that is located on the same device in order to cope robustly with mobile network disconnections. As the applications only deal with a local server with limited storage space, they have to do without storing a vast amount of context history.

.
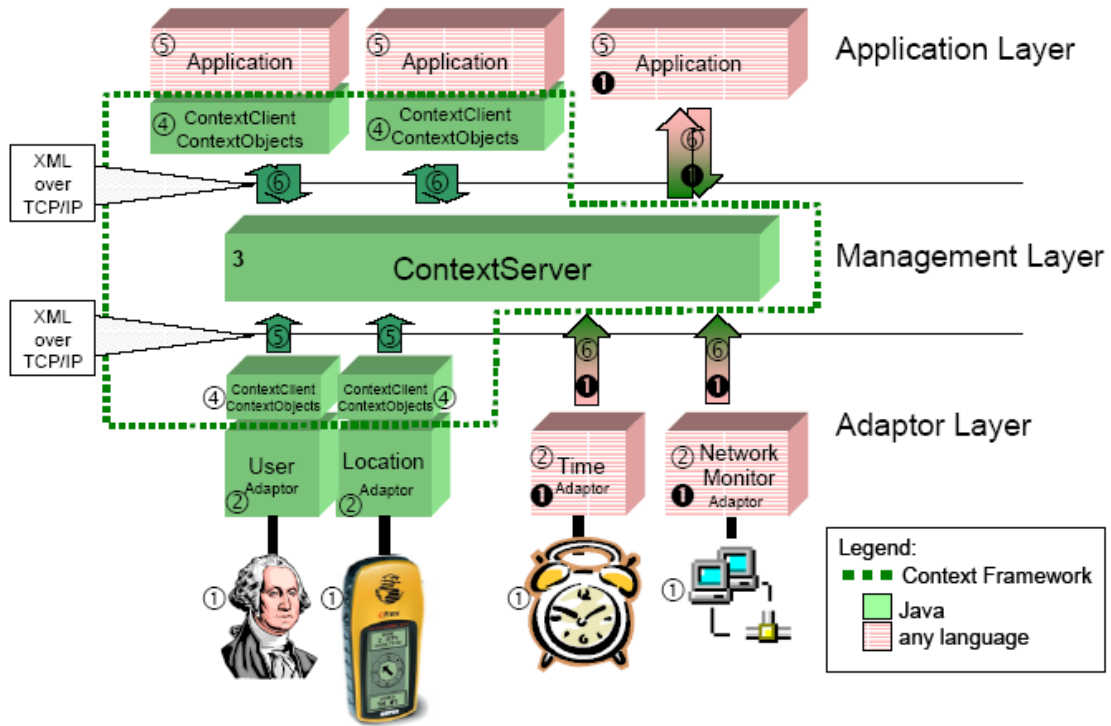
Figure 2-5 Architecture of Hydrogen Project

Hydrogen distinguishes between remote and local context as shown in Figure 2-6. *Local context* contains several *ContextObjects*, which is information that our own device is aware of as provided by any attached sensors. *Remote context* is information other devices know about, and is accessible over the network such as WLAN or Bluetooth. The current context model comprises of five types of context in *ContextObjects*.

- Time - is the current time as provided by the system clock of the used device.

- Location represents the current physical position of the device.

- Device consists of a unique identifier and a device type.

- User contains information about the current user of the device.

- Network contains information about the available network connection types of the device.
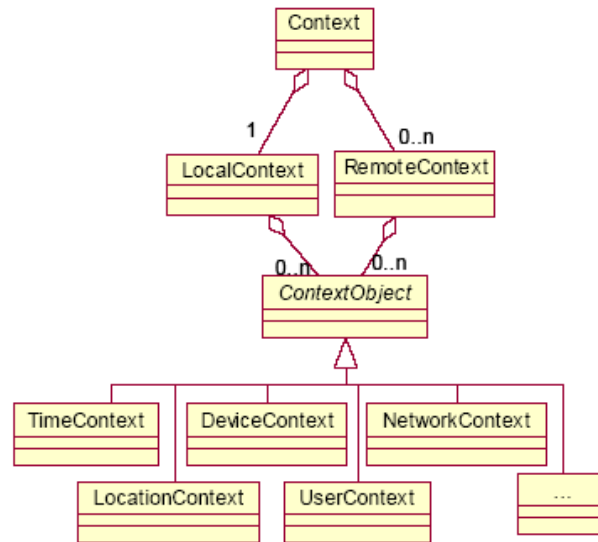
Figure 2-6 Hydrogen's Object Oriented Approach

More specialised types of context can be added to the framework by specialising *ContextObjects* class, which is a base for all context objects.

As mentioned, the application subscribes context according to what it needs. Moreover, applications have their own interpretation of context that they are subscribed to. Therefore if the applications have not the same interpretation, the code has to be rewritten.

Furthermore, the context model only supports context in the same domain and for a particular application in a similar way to the Context Toolkit. When the domain

is changed, the designers have to reconsider what should be included and how to model the *ContextObject*.

CORTEX [Biegel and Cahill, 2004]

CORTEX system uses a context-aware middleware approach. The architecture is based on the Sentient Object Model which was designed for the development of context-aware applications in an ad-hoc mobile environment. The sentient object model incorporates the STEAM event service [Meier and Cahill, 2003] to provide communication among components of the model including sensors, which produce software events and actuators, which consume software events.

Figure 2-7 illustrates that a sentient object which consists of 3 main parts can be both producer and consumer of another sentient object:

- *Sensory capture* performs sensor fusion in order to manage uncertainty of sensor data and derive higher level context information from multi-modal data sources. A probabilistic sensor fusion scheme is employed, based upon Bayesian networks, which provides a powerful mechanism for measuring the effectiveness of derivations of context from noisy sensor data.

- *Context hierarchy* holds and handles the set of contexts. The overall context of a sentient object is made up of a set of discrete environmental facts and data. These multi-modal context fragments are fused by the sensory capture component to determine higher level contexts. The set of contexts in which an object may exist is represented as a hierarchy, based upon the Context-Based Reasoning (CxBR) paradigm [Gonzalez and Ahlers, 1999].

- *Inference engine* is responsible for changing application behaviour according to context and leverages the existing capabilities of the

CLIPS (C Language Integrated Production System) production system language [Giarratano, et al., 2004]. Sentient objects are made context-aware by using conditional rules to specify application behaviour in different contexts; in other words the objects follow an Event-Condition-Action execution model [Ipiña, 2001].
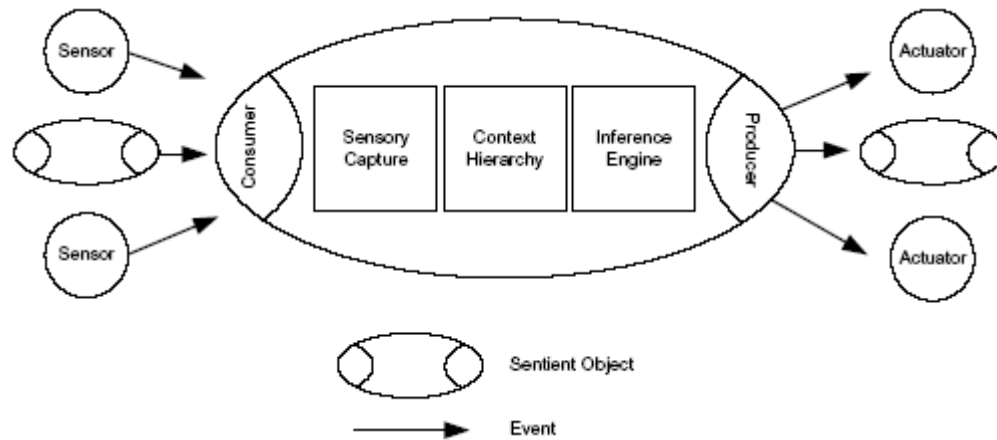
Figure 2-7 Sentient Object Model

Context Managing Framework [Korpipää *et al.*, 2003]

Figure 2-8 represents the CMF context framework that contains 4 major components:

- *Context manager* represents a centralised server managing a blackboard while other entities (except security) act as clients. It stores context data and provides this information to the client applications.

- *Resource servers* connect to any context data source and post context information to the context manager's blackboard, which further processes the data if needed and delivers it to the clients according to their subscriptions.

- *Context recognition service* stores recognition service table registers. The resource server and recognition service convert an unstructured

raw data flow into a representation defined in the context ontology shown in Table 2-3 by using a fuzzy logic. It permits serving the human-interpretable context information for the applications.

- Application can operate by using the high-level contexts without needing to know about the underlying process.

The context ontology provides 5 main categories: location, time, environment, user and device. The framework lets applications subscribe to the required context information in an event based manner. This can be a time consuming process if the application requires several context types in the ontology because the user has to go through different types of the ontology which contains 5 main categories and subscribe the required context. Moreover, the process of selecting the context is required for different applications; the process can be burdensome to users. For example, if different applications require the same set of context, the user still has to redo the process for the new application. It shows that different context categories can be reused but the reasoning of the context in a situation (high-level interpretation) is not reusable as it has no formal structure; each application has its own subscription of context. Formal structure in this case means it provide a consistent context model and context reasoning process. Application should be able to access the context model through an interface so it interacts with system in the plug and play manner. It should not have to subscribe different information from sensor devices or high-level context for every new application. The changes in application or sensor technology should have minimal effect on the context model.
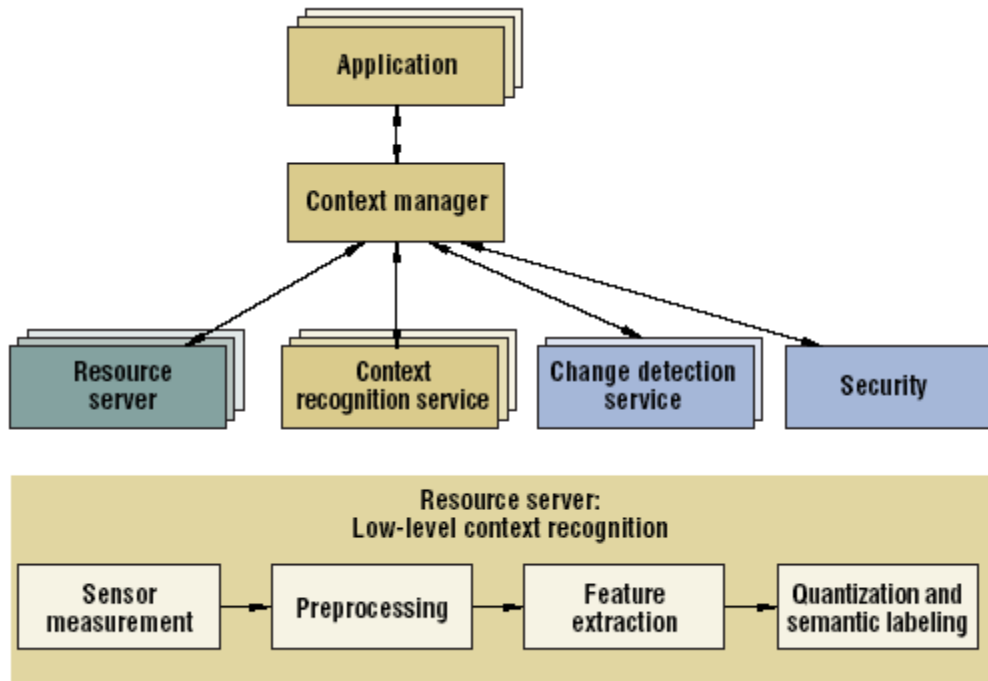
Figure 2-8 Context Managing Framework Architecture

Context type	Context value
Environment:Sound:Intensity	{ Silent, Moderate, Loud }
Environment:Light:Intensity	{ Dark, Normal, Bright }
Environment:Light:Type	{ Artificial, Natural }
Environment:Light:SourceFrequency	{ 50Hz, 60Hz, NotAvailable }
Environment:Temperature	{ Cold, Normal, Hot }
Environment:Humidity	{ Dry, Normal, Humid }
User:Activity:PeriodicMovement	{ FrequencyOfWalking, FrequencyOfRunning, NotAvailable }
Device:Activity:Stability	{ Unstable, Stable }
Device:Activity:Placement	{ AtHand, NotAtHand }
Device:Activity:Position	{ DisplayDown, DisplayUp, AntennaDown, AntennaUp, SidewaysRight, SidewaysLeft }
Context type (higher-level)	**Context value**
Environment:Location:Building	{ Indoors, Outdoors }

Table 2-3 Example of Sensor-based Context Ontology

Service-Oriented Context-Aware Middleware (SOCAM) [Gu, et al., 2004]

SOCAM (see Figure 2-9) uses a central server (Context interpreter) which gains context data through distributed context providers and offers it in mostly processed form to the clients.

It consists of:

- *Context providers* abstract useful context data from internal physical sensors or external virtual sensors. It converts the low-level context sensing to the high-level context in OWL [Smith, et al., 2003] representations so that the context can be shared and reused by other services components.

- *Context interpreter* acts as a context provider as it provides high-level contexts by interpreting low-level contexts using logic reasoning services. It consists of a context reasoner and a context KB.

The context reasoner has the functionality of providing deduced contexts based on direct contexts, resolving context conflicts and maintaining the consistency of the context KB.

The Context KB provides a set of APIs for other service components to query, add, delete or modify context knowledge. The Context KB contains context ontologies in a sub-domain and their instances.

- Context Database Service stores a context ontology and past contexts for a sub-domain. There is one logic context database in each domain.

- *Location service* allows users, agents and applications to locate different context providers – it acts as resource discovery.

- *Context-aware mobile services* are applications and services that make use of different levels of contexts and adapt the way they behave according to the current contexts.

46
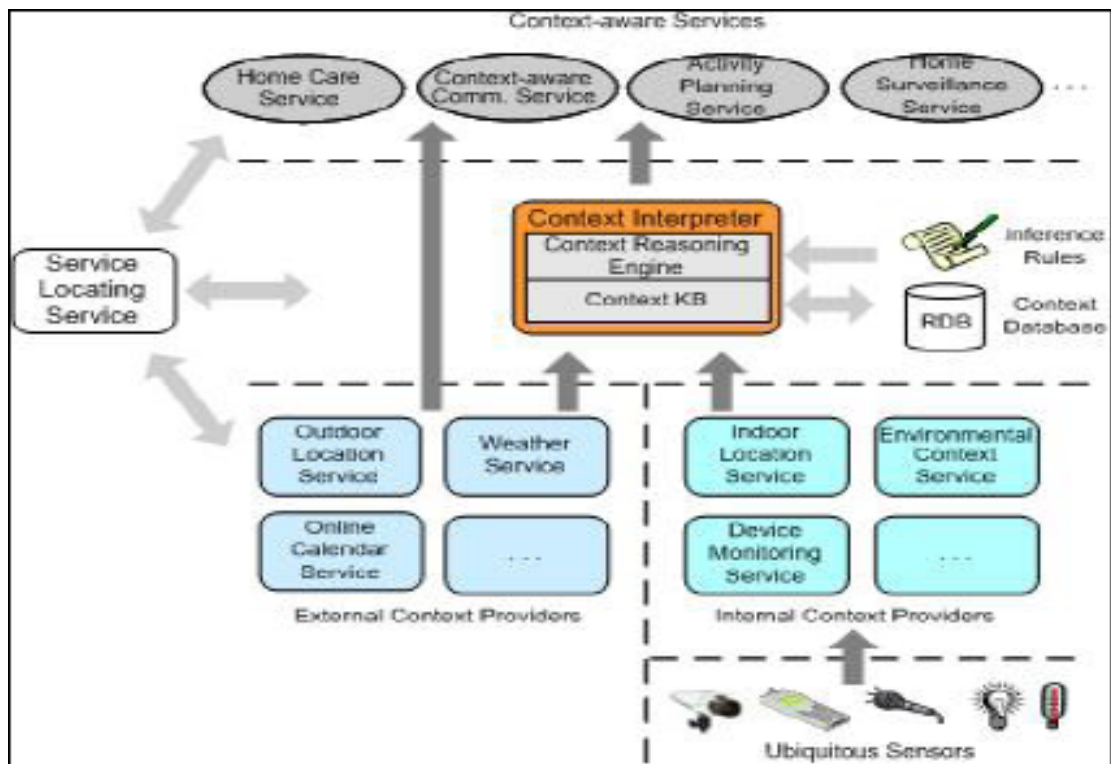
Figure 2-9 SOCAM architecture

The SOCAM architecture presents a formal context model based on an ontology. Contexts are represented as predicates written in OWL. The benefit of the ontology-based approach is that context knowledge can be shared among different entities and reasoning about context becomes possible. However, the logic context database is required in each domain as there is no uniform separation of context categories and the reasoning process that can be reused in different domains.

Context Broker Architecture (CoBrA) [Chen, et al., 2003]

CoBrA is an agent based architecture (see Figure 2-10) for supporting context-aware systems in smart spaces. The heart of the CoBrA is the intelligent context broker. The broker's main responsibility is to maintain and manage a centralised

model of context that can be shared by all devices, services and agents in the space and provides privacy protection for the users in the space by enforcing the policy rules that they define. The broker uses rule based logical inference for context reasoning and knowledge maintenance. The context broker contains the following:

- *Context knowledge base* provides persistent storage of the context knowledge.

- *Context Reasoning engine* determines contextual information that is stored in the context knowledge base that cannot be directly acquired from sensors (e.g. intentions, roles, temporal and spatial relations).

- *Context acquisition* acquires contextual information from sources that are unreachable by the resource-limited devices.

- *Privacy management* protects user privacy by enforcing policies that the users have defined to control the sharing and the use of their contextual information.
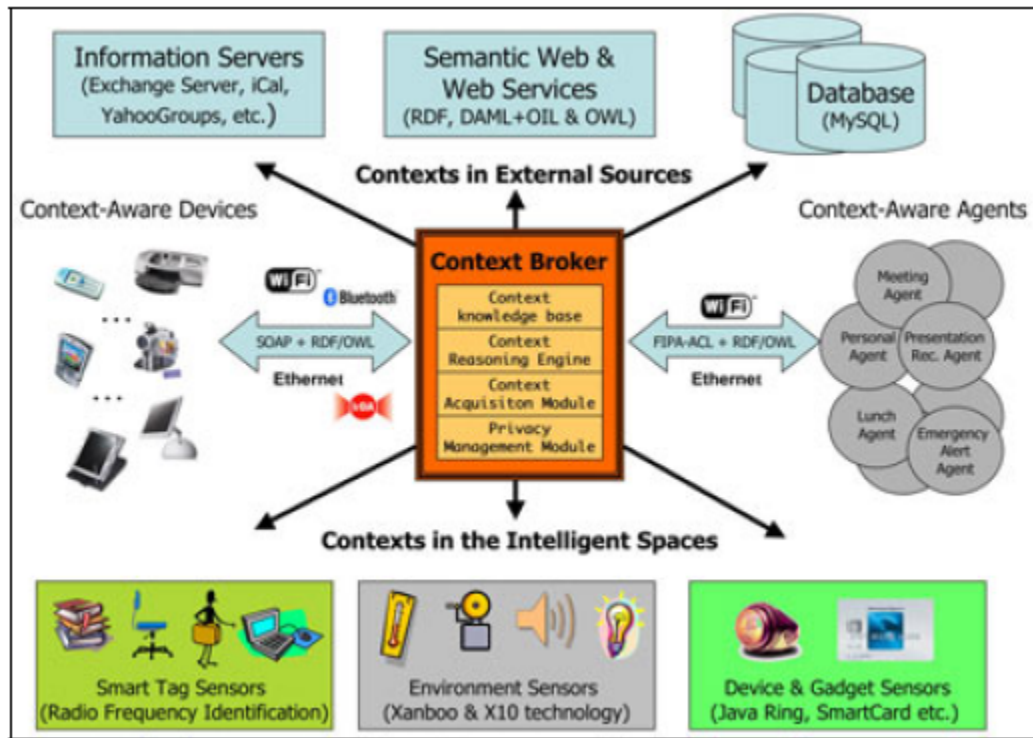
Figure 2-10 CoBrA architecture

CoBrA uses the Web Ontology Language OWL to define ontologies for context representation and modelling, defines rule-based logical inference for context reasoning and knowledge maintenance, and provides a policy language for users to control the sharing of their private information. Their ontology is categorised into four distinctive but related themes:

- Ontologies for physical places

- Ontologies for agents (both human and software agents)

- Ontologies for the location context of the agents

- Ontologies for the activities context of the agents

As seen in Figure 2-11, the role is predefined as part of the information about the agent. As seen in the previous context clsssification in Table 2-1, user is influence by the society and has a social status. As a result, user has a role in the society in different situations. The role of the user can be inferred from the user's current location, people around the user (community) or time. Moreover, the context broker defines different rules for a rule-based logical inference for context reasoning. For the context broker to be able to provide support to the agents with a context-aware ability, the defined rules are created in different manners for the system to detect the situations. For example the rules are referring to a different part of the ontology or different sets of ontologies. It does not provide developers with a uniform method of high level interpretation. The uniform method allows the applications to access context model through an interface rather than directly predefined or subscription of different context for each application. Therefore when a new domain is introduced, with the predefined or subscription context method, the predefined role and the rules need to be redefined and rewritten for each application.

CoBrA Ontology Classes		CoBrA Ontology Properties	
"Place" Related	Agents' Location Context	"Place" Related	Agent's Location Context
Place AtomicPlace CompoundPlace Campus Building AtomicPlaceInBuilding AtomicPlaceNotInBuilding Room Hallway Stairway OtherPlaceInBuilding Restroom Gender LadiesRoom MensRoom ParkingLot	ThingInBuilding SoftwareAgentInBuilding PersonInBuilding ThingNotInBuilding SoftwareAgentNotInBuilding PersonNotInBuilding	latitude longitude hasPrettyName isSpatiallySubsumedBy spatiallySubsumes accessRestricted- ToGender lotNumber	locatedIn locatedInAtomicPlace locatedInRoom locatedInRestroom locatedInParkingLot locatedinCompoundPlace locatedInBuilding locatedInCampus
"Agent" Related		"Agent" Related	
	Agent's Activity Context		Agent's Activity Context
Agent Person SoftwareAgent Role SpeakerRole AudienceRole IntentionalAction ActionFoundInPresntation	PresentationSchedule Event EventHappeningNow PresentationHappeningNow RoomHasPresentationHappeningNow ParticipantOfPresentation- HappeningNow SpeakerOfPresentationHappeningNow AudienceOfPresentationHappeningNow PersonFillsRoleInPresentation PersonFillsSpeakerRole PersonFillsAudienceRole	hasContactInformation hasFullName hasEmail hasHomePage hasAgentAddress fillsRole isFilledBy intendsToPerform desiresSomeone- ToAchieve	participatesIn startTime endTime Location hasEvent hasEventHappeningNow invitedSpeaker expectedAudience presentationTitle presentationAbstract presentation eventDescription eventSchedule

Figure 2-11 List of Classes and Properties in COBRA-ONT v0.2

STU21 [Conway, 2006]

Stu21 is a distributed agent-based framework, and uses ontology to define context. This framework provides standard interfaces between components, publish-subscribe functionality and a directory lookup service which acts like "The Yellow Pages". To search for a service, a client agent can search a yellow- and white-pages directory. Once the client locates the service it can send standard messages or subscribe to receive published information.

In the STU21 model, the primary actors are:

- *PersonAgent*. This is a subclass of the Context aware Agent that acts on behalf of an individual.

- *RoomAgent*. This is a subclass of the Context aware Agent that acts on behalf of a smart space.

- *A myriad of other agents* like table agent, chair agent, desk agent, projector agent, and light agent. In fact, an agent corresponding to any object/entity can be added to the system. This object can either contribute to context by providing information or use the context information to carry out autonomous productive work.

- *Context Broker* acts like the CoBrA context broker that models a set of spaces in concert with subsidiary RoomAgents.

- *SensorAgent*. This is an agent that wraps a sensor in the environment, creating a conduit for the input of arbitrary context into the framework.

- *Various associated agents*. In Figure 2-12, there are several types of agents. These include:

  - *ResourceAgent* – this is an agent that can represent a resource available within a smart space, in the way that a SensorAgent represents a generic sensor.

  - *IntermediaryAgent* – this is an agent that works autonomously on behalf of a person, monitoring, searching, or negotiating to achieve some aim on behalf of the individual.

  - *Context Monitoring/Gathering agent* – this is an agent that monitors context for some purpose.
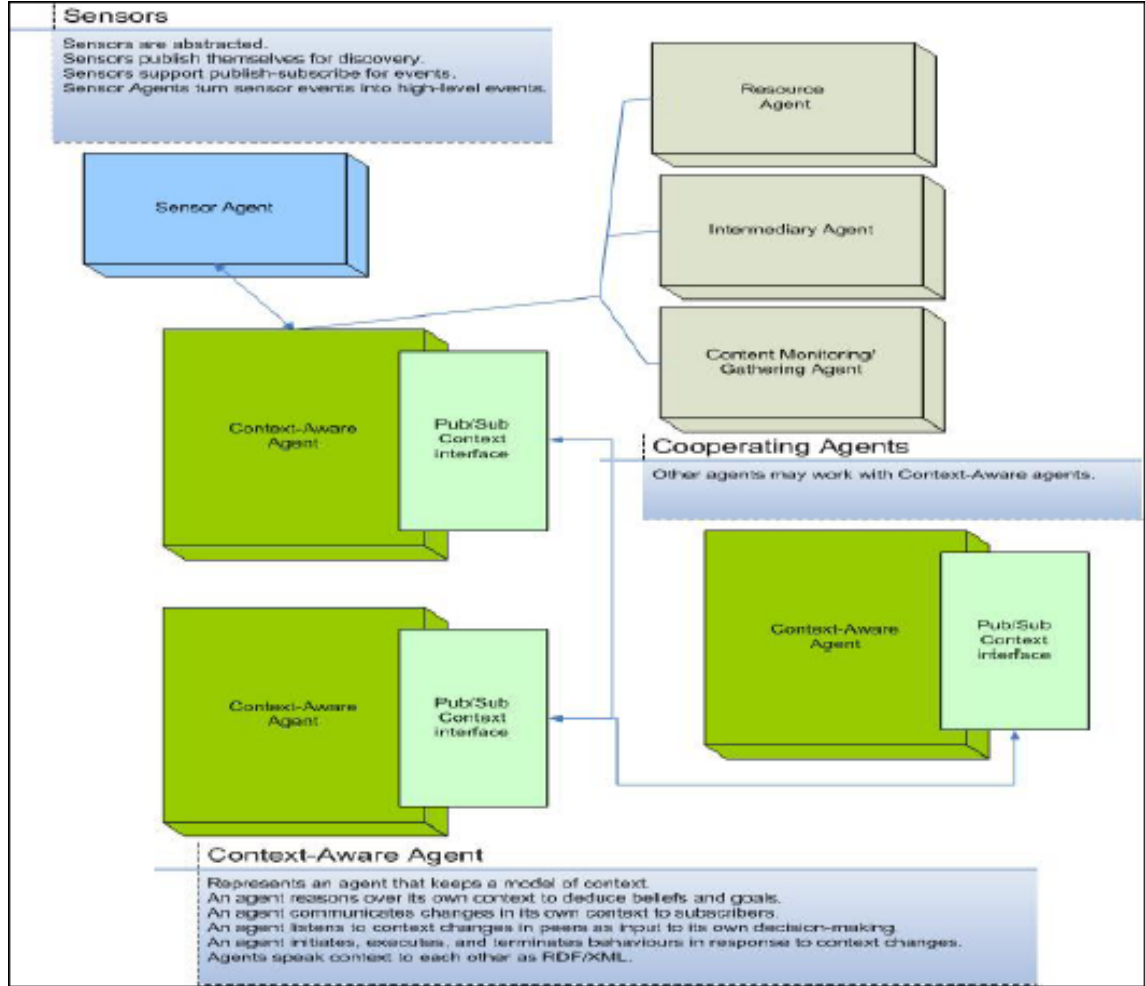
Figure 2-12 STU21 architecture

Stu21 can be considered as an extension over CoBrA. The rules and inference part are kept separate from the ontology, thus allowing different rule representations and inference engines to coexist. Though this makes it possible to switch rule base and inference engine without affecting the rest of the system, it has the big disadvantage of representing rules based on semantic meaning of entities independent of the semantic representation of the entities. Thus, making changes in the ontology would require encoding new rules in the code to account for these changes.

The rules may frequently be a very large and dynamic set. In the current setup, the rules are hidden from the users and are dependent upon programmers and even small changes can lead to broken systems. Thus, this setup is not suitable for an extensible and efficient system. This problem could be solved when the system provides a uniform structure of high-level context elements and the relationships between them rather than inconsistency embedded rules involving low level information. Therefore, by having rules in a simple standard form that is independent of a particular implementation, through its consistency, the users can easily build understanding about the system and be able to improve the system efficiency.

2.2.3.4 From Location Based System to Context Aware Framework

In conclusion, Location Based Systems are limited to one type of context (i.e. location). Context Aware Systems take account of more than one type of context (i.e. location, time, user's environment, society, etc.) which is potentially more useful in a ubiquitous computing environment. This is because there are vast amounts of information that will be available and this information is important for the system to utilise in order to obtain a better understanding of the user. However, Context Aware Systems tend to be limited to specific applications or domains. Context Aware Frameworks, on the other hand, are the most reusable and generic, taking account of more than one type of context in a flexible way.

Table 2-4 shows the differences and similarities in different frameworks. It compares different frameworks according to significant essentials in context-aware system development:

- Context representation: How does the framework represent the context in the architecture?

- Context Processing: How does the framework reason about the context?

- Context Model: How does the framework model the context? What types of context are taken into account? This is different from the context representation because the context model is about what are taking into account as context in the system but the representation is about how the context is used or implemented in the architecture.

- Architecture: What is the framework's mechanism for the architecture? How the system can be implemented?

- Sensing: How does the framework gather data from the sensors?

- Historical Context Data: How does the framework support the use of historical context data during reasoning?

- Resource Discovery: How does the framework support resource discovery?

- Security and Privacy: How does the framework support security and privacy of the user and data?

Table 2-4 shows that the existing frameworks' researchers have one vital objective in common. This objective is attempting to make sensors transparent to the context-aware system. The frameworks have a sensing module (in Table 2-4) to separate the acquisition and representation of context from the delivery and reaction to context. As a result when there are new sensors, they can be added to the system without affecting the use of information from existing sensors in the system. However, the existing frameworks focus on context modelling and context processing (i.e. reasoning rules or inference) for particular domains or applications. Existing context modelling and context processing are either technology-driven or deal with certain types of context and embed context processing in the context model. This means that the predefined context model and context processing for inferring the situation are very specific to a particular application, domain and types of context. This causes a weakness in the

frameworks when there are new applications, domains or new types of context to take into account.

When there are new applications, domains or types of context, the frameworks do not cope well. The existing context model and reasoning process are not suitable to be reused or expanded. This is because the context model and reasoning process are irregular combinations of low level context (i.e. sensor data) and, in some cases; high level context (i.e. translated data) depends on the availability of the context at the time of implementation. Moreover, the context model and reasoning process are hidden from users. As a result, even though the sensors are transparent to the system, when a new application, domain or type of sensor is used, the context model and its reasoning process require modification. Thus, the designers and developers have to get involved in the process of adding the functionality (i.e. subscribing to new types of context, remodelling a predefined context model and reasoning rules) to cope with new applications, domains or types of context. The process is time consuming and inefficient if users have to turn to developers for every new application, domain or type of context.

Existing context-aware systems process the context by either using predefined rules or context models (i.e. the inference part) in order to infer the situation where the user requires support from the system. These rules and models are stored in the system database. Most of the systems also provide availability of historical context data in the system database. However, systems such as Hydrogen and Context Managing Framework infer the situation using predefined rules but do not provide the historical context data. This eliminates the possibility of the system's exploiting context history to improve efficiency.

With the Context Aware Frameworks, researchers are given more flexibility to design and develop context-aware applications without having to concern

themselves with sensors. This is because of their sensors transparency, unlike in the Location Based Systems and Context Aware Systems where the implementation of sensor module is embedded in the applications. The Context Aware Frameworks provide the facility of separating the sensing module (i.e. context providers, sensor nodes and sensor access module in Gaia, CASS and MidCase projects respectively) from the applications. Therefore the researchers are able to concentrate on implementing and improving the ability of context-aware applications without worrying about changes in sensor technology.

	Context Representation	Context processing	Context model	Architecture	Sensing	Historical context data	Resource discovery	Security and Privacy
Gaia '02	4-ary predicates (DAML+OIL)	First order logic and Boolean algebra (predefined rules)	technology driven	MVC (extended)	context providers	available	discovery service	supported (e.g. secure tracking, location privacy, access control)
CASS '04	relational data model	inference engine (forward chaining technique) and knowledge base	technology driven	centralised middleware	sensor nodes	available	n.a.	n.a.
MidCase'07	Context tuple Space	rule engine and fact base	technology driven	peer to peer services with centralised middleware (5 layers & 2 crosslayer modules)	sensor access module	available	service registry	hardware authentication
Context Toolkit '00	attribute-value tuples	context interpretation (query by attribute) and aggregation	technology driven	widget based	context widgets	available	discoverer component	context ownership
Hydrogen '03	object oriented	interpretation and aggregation of raw data only	Time, Location, Device, User, Network	three layered	adapters for various context types	n.a.	n.a.	n.a.
Context Managing Framework '03	ontologies (RDF)	context recognition service (fuzzy logic to build higher level concepts from uncertain sensor data)	Location, Time, Environment, User, Device	centralised blackboard	resource servers	n.a.	resource servers + subscription mechanism	n.a.
SOCAM '04	ontologies (OWL)	forward and backard chaining rules engine	home domain,office domain etc. and defined individual low-level ontologies in each sub-domain	distributed with centralised server	context providers	available	location service	n.a.
CoBrA '03	ontologies (OWL)	rule based logical inference engine and knowledge base	physical places, agents, location, activities of agents	agent based with centralised context broker	context acquisition module	available	n.a.	Rei policy language
STU21 '06	ontologies (OWL)	rule based inference engine and knowledge base	person, room, objects	distributed agent based	agents	available	resourceAgent	rules

Table 2-4 Summary of Context Aware Frameworks

2.2.4   Types of Context-Aware Applications

In the past, researchers have tried to classify context-aware computing so that they can better understand and use context more effectively. In this section, different classification systems of context-aware *applications* are discussed.

Schilit, Adams and Want [Schilit *et al.*, 1994] have categorised context-aware computing by its tasks (whether a *task* is to get information or to execute a command) and actions (whether the *actions* are triggered manually or automatically). They categorise context-aware computing into four types as follows:

- *Proximate selection application*: Retrieve information for the user manually based on available context. Nearby objects are emphasised or otherwise made easier to choose via the user interface.

- *Automatic contextual reconfiguration*: Retrieve information for user automatically based on available context. New components are dynamically added while existing components are removed or connections are altered.

- *Contextual information and commands*: Execute a command for a user manually based on available context.

- *Contextual-trigger actions*: Execute a command for a user automatically based on available context - based on the "if-then" rule.

Pascoe [Pascoe, 1998] introduced another classification system based on context-aware features. This is a set of basic capabilities that the context-aware computing system should have. These capabilities are as follows:

- *Contextual sensing*: The ability to detect contextual information and present it to the user, augmenting the user's sensory system. This is

similar to the *proximate selection application* of Schilit, Adams and Want.

- *Contextual adaptation*: The ability to execute or modify a service automatically based on the current context. This is similar to the *Contextual-trigger actions* of Schilit, Adam and Want.

- *Contextual resource discovery*: The ability to allow a context-aware application to locate and exploit resources and services that are relevant to the user's context. This is similar to the *automatic contextual reconfiguration* of Schilit, Adam and Want.

- *Contextual augmentation*: The ability to associate digital data with the user's context. This is a new ability that Pascoe has added, compared to the classification of Schilit, Adam and Want.

Dey and Abowd [Dey and Abowd, 1999] tried to simplify and combine the above as follows:

- Presentation of information and services to a user: This is a combination of *proximate selection application, contextual information and commands, contextual sensing* and *contextual resource recovery.*

- Automatic execution of a service: This is a combination of the *contextual-trigger actions* and *contextual adaptation.*

- Tagging of context to information for later retrieval: This is equivalent to the *contextual augmentation.*

The last classification of context-aware computing that will be discussed here is by Chen and Kotz [Chen and Kotz, 2000]. They classified context-aware computing according to how context is actually used in an application and identified two types of context-aware computing as below:

60

- *Active context awareness*: An application automatically adapts to discovered context, by changing the application's 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.

Researchers	Types of Context Aware Computing				
Schilit, et al (1994)	Automatic contextual reconfiguration	Contextual-trigger actions	Proximate selection application	Contextual information and commands	
Pascoe (1998)	Contextual resource discovery	Contextual adaptation	Contextual sensing		Contextual augmentation
Dey (1999)	Automatic execution of a service		Presentation of information and services to a user		Tagging of context to information for later retrieval
Chen and Kotz (2000)	Active		Passive		

Table 2-5 Types of Context Aware Computing

From existing publications it is evident that the study of active context-aware computing is more popular than passive context-aware computing. This may be because it introduces new levels of interactivity compared to the traditional interactivity level, personalisation. The personalisation level is where the computer lets the user specify her own settings for how the computer should behave in a given situation [Barkhuss and Dey, 2003]. Context aware computing

has introduced these new levels of interactivity to the user. However, compared to more traditional methods of interactivity such as personalisation, active and passive context-aware systems reduce user control. An important question should then be raised regarding "how users feel about context-aware computing taking control away from them".

Active context awareness performs tasks for users automatically. In order to provide a large degree of autonomy, researchers have tried to use different context models. Active context awareness takes control away from the user completely. This introduces the problem of loss of control. Barkhuss and Dey [Barkhuss and Dey, 2003] have carried out research into this but it is still at an early stage. They have concluded that users are willing to accept a large degree of autonomy from applications as long as the application's usefulness is greater than the cost of limited control. Other researchers who support active computing are Brown and Randell [Brown and Randell, 2002]. They state that the user could cope with autonomy as long as context is used "defensively". This defensive use of context means that contextual information is used to decide what the device does, but only in a way which would not be likely to cause irritation or bother to the user if the inferences made from context are incorrect.

Passive context-aware systems automatically represent new context to the user although the user still has some control over how to use the context. Brown and Randell [Brown and Randell, 2002] suggest giving simple resources to users so that they themselves can decide how best to use these resources in what they do. They also argue that context is of great value when it is presented to users themselves to interpret. However, they warn that the context that is represented must be in a simple structure so the user can make sense of the context. These arguments by Brown and Randell support passive context-aware systems while attempting to avoid the problem that occurs in active context-aware systems where

complex context models are used to fully reason with human behaviour. However, problems exist with passive computing too. For example, a user is still involved with a large number of explicit interactions. Also, there could be too much information represented to the user.

In conclusion, active context-aware computing takes control away from the user but the user still requires at least some kind of explicit interaction. This type of computing helps reducing tasks overload for user but as mentioned it can be defensive to user. In ubiquitous computing, there are immense amount of devices and services in the environment and constantly changing around users while they are trying to perform their multitasking. Therefore, by having the system to automatically, support the users can improve users' efficiency. However, experimented by Barkhuss and Dey [Barkhuss and Dey, 2003] demonstrates that the users are willing to use the system if the automation usefulness is greater than the loss of control. At the same time, the system should prepare to provide an option for the user to have their control back. Passive context-aware computing gives the user some control but could present too much information to the user. Personalisation can give the user a high level of control over the system but requires a much higher amount of explicit interaction.

This kind of approach may help the researchers to further the field by developing support for different types of applications at the framework or architecture level. For example, in active applications, the architecture may support the user by managing the presentation of context to users so that they have an understanding of how the system came up with its decisions and feel less loss of control.

2.2.5 From Previous Context Awareness to the Present

Researchers have tried to enhance their understanding of context by defining and classifying context. Previous context classifications are diverse and cover

different elements of context. From context classifications, researchers have developed context-aware systems. Classifications can be used to model and understand the user in context-aware systems.

Context aware systems progress from sensor based systems, which are limited to one type of context, to frameworks that support multiple types of context and sensors that are transparent to the applications. This advantage of the framework allows developers to replace, remove or add new sensors from/to the system without affecting the applications. Existing frameworks use different architectures and context models. Architectures support different services to systems and users such as resource discovery and security. A context model is essential in developing the context-aware system. This is because the context model is used by designers of the system to make inferences about the user in different situations in order for the system to react to the situation appropriately in real time. The process of building a context model of users' situations can be expensive and time consuming.

The field of context awareness is still immature. There are many major challenges facing researchers. Crucial current challenges are discussed in the next section.

2.3   *Problems in Context Awareness*

2.3.1   **Impossible to Acquire Context**

Many previous context-aware applications [Helal, et al., 2005; Kim, et al., 2004; Muñoz *et al.*, 2003; Park, et al., 2006] were implemented based on a context definition that was defined as *any information* that can be used to characterise the situation of an *entity* [Dey, 2001]. However, it is impossible to attach a sensor to a device for every relevant type of context. Even though there are new technologies that allow us to sense various types of information, sensor technology is still in a

64

developing stage. All the information that could in principle be used to characterise the situation of an entity from sensors cannot therefore be collated. Moreover, the data from sensors can become unavailable or inaccurate due to the capabilities of the technologies together with the nature of the open and dynamic environment of ubiquitous computing users. For example, GPS or Bluetooth may have problems with power consumption and signal range. Accelerometer based motion sensors may be inaccurate, for example due to sudden changes of position. Therefore the data from sensors can also be insufficient, uncertain, dynamic and too heterogeneous for the system to make reliable inferences about the user.

2.3.2 Expensive to Process Context

In order to use context in a context-aware system, there are several ways and steps of transforming raw data from sensors to meaningful data for the system. First, the raw data may be processed to reduce noise [Roberts, et al., 2005]. Secondly, for applications that use a single type of sensor, the raw data may be processed into more meaningful – to the user – information. For example, the raw data from GPS is a combination of latitude and longitude. Longitude and latitude information are numerical data that identify positions on the Earth's surface relative to a datum position. Therefore they are not intuitively readable for many users. However, longitude and latitude may be processed and translated into useful information such as addresses with hierarchical structure using a special database, for example, "7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan" [Aizawa, et al., 2004]. Thirdly, the useful information or raw data from the sensor is used to provide information or services to the user.

If the context-aware system gathers information from more than one sensor or type of sensor, the useful information or raw data from different sensors can be combined and processed together to be used in various ways to support the users. Different types of system use different reasoning methods. Combining data from

multiple sensors or multiple types of sensors can become complicated and thus expensive, and can cause conflict between raw data from different sensors during the context reasoning process.

2.3.3   Not Being Used in the Real World

In Section 2.2, it was shown that the majority of the previous research in context awareness has not really been tested in the real world. The two main reasons for context-aware systems not being used in the real world are discussed below. First, a lot of research in the context awareness field appears to be based on the assumption that in some application domains, context is not continuously changing and that it is therefore feasible to represent context in rather static data structures.

Secondly, the computation and reasoning processes in context-aware systems can be very complicated. Previous research suggests that there is little hope that this problem will soon be overcome to enable systems that are context-aware in a non-trivial sense. However, the difficulty of the problem does not suggest abandoning research into context awareness in the real world. It suggests keeping these problems in mind. The potential for failures should be taken into account when designing context-aware systems or applications (e.g., [Bellotti and Edwards, 2001; Lueg, 2001]).

Largely for the two reasons above, implemented context-aware systems are error prone. The system often does not provide the appropriate service or information to the user, causing annoyance and a poor user experience. Therefore users stop using the system because it causes annoyance and because they do not know how to solve the problems [Lueg, 2002].

66

2.3.4   Broad Definition of Context

The most frequently referenced definition of context was presented by Dey [Dey, 2001]. According to Dey, context is defined as any information that can be used to characterise the situation of an entity. An entity should be treated as anything relevant to the interaction between a user and an application, such as a person, a place or an object, including the user and the application themselves. This is broad, but conveys an important perspective in its emphasis on people [Winograd, 2001]. By being broad, it can mean different things to different people [Crowley, 2006]. Instead of focusing on developing a better understanding of the nature of context, many researchers have naturally taken a technology-driven approach. The researchers are limiting themselves to whatever technology is available. It drives them away from concentrating about user's requirement. By taking a technology-driven approach, it is relatively simple to design and develop a system. However, it limits the development of a context-aware system that has an adequate model of the user and her intentions and activities in the world. If the context-aware system does not have such an adequate model, the system is prone to producing errors through its reasoning about context and its presentation of the results through a user interface, thereby causing annoyance to the user.

Developing an adequate model of context is far from easy. It has been a problem in classical representation-based artificial intelligence (AI). Pylyshyn [Pylyshyn, 1987] proposed that the problem is to do with what aspects of the world have to be included in a sufficiently detailed world model and how such a world model can be kept up-to-date when the world changes. Indeed, the problem has been shown to be intractable in realistic settings (e.g. [Dreyfus, 2001]). The real world is constantly changing, intrinsically unpredictable, and infinitely rich [Pfeifer and Rademakers, 1991]. The problem has often been treated as a technical problem but it can also be understood as an ontological problem, as aspects of the world included in a world model determine the understanding of the world based on the model. However, facts not included in the model and not derivable from the

model cannot be explained based on the model. Hence, the problem in AI is directly related to trying to understand any notion of context [Lueg, 2002].

Greenberg points out that it may be difficult or impossible to determine an appropriate set of canonical contextual states and it may also be difficult to determine what information is necessary to infer a contextual state [Greenberg, 2001]. Goodwin and Duranti argue that it does not seem possible at the present time to give a single, precise, technical definition of context, and eventually we might have to accept that such a definition may not be possible [Goodwin and Duranti, 1992].

Even though it seems difficult to find an appropriate definition of context, a context-aware system design tool that at least takes account of this problem would help developers in their efforts to produce better context-aware systems.

2.3.5 Infinite Context Classification

Since previous context definitions were so broad, several researchers have tried to develop better understandings of context by producing classifications of the key elements of context, as shown in Section 2.2.4.

Previous classifications have covered different aspects of context. These findings have shown that there are a large number of elements that make up context. Indeed, the scope of context is potentially infinite, encompassing everything that may in one way or another influence the user. Before we use context in applications, we need to have an understanding of what the system should take into account as context. Clearly, we need a way of reducing this infinite set to something more manageable so that it is more possible to implement. It is therefore important that the most influential and critical elements of context that

have an influence on a human's activity in ubiquitous computing are identified and fully analysed. To date, this level of analysis has not been carried out and there is confusion surrounding the various elements.

2.3.6   Lack of Uniform Relationships between Elements of Context

There has been little research exploring the relationships between different elements of context and how these relationships can affect the efficiency of context-aware applications. As mentioned previously, context is not static; the same set of data from sensors can be interpreted differently according to the situation. This relationship between context elements is an important part of providing an adequate context definition or classification [Kaenampornpan and O'Neill, 2004a]. These relationships are also valuable in order to use context to represent the world of the user and to help the system to better understand the user's activities and intentions, acknowledging that humans assimilate multiple items of information to perform everyday tasks. For example, the relationship between the user and the people around them could bring social status into the context reasoning process. The social role of a user with respect to the same set of people around her can be different according to the time and place that she is situated in. For example, a user's social role could be a work colleague when she is at work but the role can change to a friend or member of a cycling group when she is on the cycle group's trip or the group meeting about the cycle trip. This illustrates that the role of a user changes over time and space [Muñoz *et al.*, 2003].

Moreover, unclear relationships between context elements lead to inconsistency in the reasoning process about user's context. As a result, designers use different combinations of context elements and implement methods of inferring about the user in the reasoning process in their own way. As discussed in Section 2.2.3, existing context models and reasoning processes are inconsistent combinations of low level context (i.e. sensor data) and, in some cases, high level context (i.e.

translated data) depending on the availability of context at the time of implementation. There are no uniform relationships between context elements to help the researchers in the field to communicate and share understandings. With this inconsistent reasoning in the context awareness field, the expensively collected data for each context element can be difficult to reuse or extend. By establishing uniform relationships, designers can reuse collective data of context elements in the context model from their existing reasoning processes.

2.3.7 Lack of Systematic Tools

Although current technology for context awareness, such as sensors, has its limitations, the technology is not the fundamental problem as it is increasingly improving. The real problem is the understanding of context and the definition of context. So far, integration of the notion of context directly into the design process is not straightforward. This is partly due to the fact that current research on context is a synthesis of different points of view, ill defined and ambiguous definitions, [Kaenampornpan and O'Neill, 2004b]. Baldauf, et al have also noted that currently there is no standard context model for sensing contextual information from various sources to enable reuse across various context-aware systems and frameworks [Baldauf *et al.*, 2006]. Furthermore, what design tools there are in ubiquitous computing, especially in context awareness, lack a bridge between requirements and implementation. There is no tool to help designers to transform raw data from requirements into implementation. Moreover, how the context-aware system should use the context has not been dealt with comprehensively. Designers and developers have little or no guidelines, step by step tools or a simple standard context model to move from the requirements or scenarios into implementation.

2.3.8   Technology Driven

Due largely to the limitations in the state of the art discussed here, in the implementation process context-aware applications have utilised only isolated subsets of their context, such as a location or a device's state; e.g. [Abowd and Dey, 2000; Luyten and Coninx, 2004]. Implementation of context-aware systems remains largely driven by technology. Often, developers implement applications according to the types of information that happen to be provided by available technology. Clearly, a programmer who is writing an application whose behaviour will depend on a user's location should not have to be concerned with details of how location is determined: whether there is a camera-based vision system, an active badge, a magnetic tracker, or some new kind of device not yet envisioned when the program was written [Winograd, 2001].

In addition, the designer should not have to be concerned with the types of information available to her as technology improves. The context model and its reasoning process for applications should not have to be remodelled when new types of information are available or in use. The only changes should be the transformation of sensor data into the information for context elements in the context model, not the context model and its reasoning process. Thus, the context model and its reasoning process should not restrict themselves to the availability of specific technologies. The applications should have a plug and play ability where the designers concentrate on user requirements and how the system should take advantage of context awareness. The system should provide an interface for each application to have the same manner of accessing context model through the uniform set of context model. The model, in this case, takes into account of information that has influence on user's activity rather than an inconsistent subscription of different information and context elements depending on availability of the technology. In order to improve the system efficiency, information that the system needs can be designed as a set of information in the database rather than ignore it and wait for the availability of the technology. In

future when a new sensor is available, the information from the database can then be replaced with one from the new sensor.

2.3.9   Summary of Problems in Context Awareness

From previous research in context awareness reviewed in this chapter, context awareness plays a key role in ubiquitous computing . Throughout the context awareness literature, researchers have tried to use different types of context in their applications. For the most part, however, context-aware applications have utilised only isolated subsets of their context, such as a location or a device's state. A truly context-aware system needs to take account of the wide range of interrelated types of context and the relationships amongst them. As a precursor to implementing such systems, researchers need an approach to modelling context that takes account of this complexity and architecture that supports the model in a ubiquitous computing environment.

Thus context-aware researchers need to consider design principles differently in order to address the challenges in context awareness discussed in this chapter. The next section identifies a number of requirements that the context model, design tool and architecture must fulfil to enable designers to deal with context more easily.

2.4   *Deriving Solutions from Problems*

The field of context awareness could benefit from a context model and systematic design tool that facilitated bringing researchers together through having a common understanding of context. This section will discuss the requirements of a context model and design tool based on the current challenges in context awareness discussed in the previous section. Furthermore, requirements for an architecture

are discussed in order to be able to support implementations based on the common context model and design tool.

2.4.1   Requirements for Context Model and Design Tool

To be able to develop a context-aware system that has the ability to cope with the issues discussed in Section 2.3, the context model and design tool should meet the requirements presented in this section.

Consistent Support for Shared Understandings of Context

Researchers use different types of context in their applications. The context model and design tool should provide a common model that contains consistent types of context so that researchers, designers and developers in the field can refer to and develop shared understandings of context and understand what key elements should be taken into account in order to have a better understanding of users' behaviours [Kaenampornpan and O'Neill, 2004a]. By using a common context model throughout the implementation, the context-aware systems should provide a consistent context model to represent to the user during runtime. The consistency in the context model should facilitate the userin having a better understanding about the context that is used by the system, which was itself developed using the same context model. Therefore the users should be able to make corrections to the system during runtime when the system makes inappropriate decisions in supporting the user through context awareness. Furthermore, by having an understanding about the system, the users should be able to adapt the underlying context model to suit their needs during runtime.

Identification of Context Elements

A context model that supports true context awareness should provide a model of context that supports both simple and complex situations. The context model

should identify important elements of context that influence the user's behaviour in a ubiquitous computing environment. The user may not be working on her own at a desktop, so her behaviour may be influenced by objects, people, the environment and society around her. The applications should not be technology-driven. The context model should allow designers to concentrate on what types of context have an influence on a user's behaviour, not what technology is available to the designer. As the context model provides elements of context that the designer can deal with in a consistent way, it will allow the designer to be able to expand the system as new technology becomes available.

By not being driven by technology, the context elements should help encourage the designer to consider different types of context as the context does not just come from sensors. Dourish [Dourish, 2004] suggested that context is characterised as "information of middling relevance". The context can also come from user profiles or schedules and/or be user supplied. These types of context have different properties such as their persistence and their uncertainty [Henricksen, et al., 2002]. The context elements in the model should guide the designer on how to deal with them differently. Moreover, by identifying the elements, the designers can spend more time concentrating on how to meet the users' requirements.

Context Interpretation

Each element in the context model should show a clear boundary of what type of information is to be taken into account. The boundaries should help the designers clarify context elements from a user's requirement and/or scenario into the context model for building context-aware applications.

Instead of using the easiest way to build a context-aware application, by directly hardwiring the drivers for sensors used to detect context into the applications

74

themselves, the designers should use the model as a guide to group different types of context in the scenarios. The code for sensors used to detect context is then developed separately from the application code. This is because before raw data from a sensor is passed directly through the application, the context model guides the developers on how to acquire and handle sensor data so that it is inferred or interpreted into information for each context element. This also transfers the raw sensor data into different levels of information in the context element, which is important as different applications may require different levels of information from the same sensor data. For example, location coordinates can refer to the identity of the building or refer to a room a user is in.

Due to the uncertainty in raw data from a sensor, the interpretation would also help to reduce the uncertainty in context as well. For example, the sensed data from two sensors and other profiles (e.g. user profiles) can be inferred to provide information for one context element. Also, when one of the sensors does not work, the inferred data of context element can be obtained from the second sensor and other profiles, where the profiles are databases that were created to hold information that cannot be obtained through sensors. The developers design the database so that the system can refer to important information that sensor cannot provide but has influence on user's activity such as their preferences.

Separation between Context and its Reasoning

By having a clear separation between context elements, developers have a common way to acquire and handle context from sensors and profiles. This first removes the burden of rewriting the code to acquire and handle data from sensors. Secondly, the context can efficiently be reused and has less uncertainty. However, truly context-aware applications deal with more than one type of context. Relationships between context information exist to describe how information is obtained from other pieces of information [Henricksen *et al.*, 2002]. The context

model should provide possible relationships between context elements (different types of context). The applications require these relationships for further reasoning in order to combine different information from context elements and derive conclusions about the user for that situation. The reasoning is mainly integrated in the context [Saternus, et al., 2007; Tonnis, et al., 2007]. In the past, each application required its own reasoning code to deal with context elements. The reasoning code normally embedded context elements and its reasoning in the particular application. Building a reasoning method can be very complex and time consuming. Different research groups deal with the reasoning method differently. It is therefore difficult to reuse the context elements and reasoning code even for a new application within the same domain. It is impossible to reuse it across domains or research groups. For example, the applications in Context Toolkit subscribe to different aggregators and widgets. Each application first of all has to be given a different code, depending on what widgets or aggregators are subscribed to. Then each application has a code for reasoning and providing decisions about the user according to subscribed widgets and aggregators. On some occasions, the applications subscribe to the same aggregators or widgets but using different reasoning rules. Even though ontologies support the separation of content and its reasoning code, the context models that previous studies opted contain inconsistent context elements. Moreover, as a result of different subscription of inconsistent combination of information and high-level context, its reasoning code and context normally are embedded in an inconsistent way. It is therefore difficult (sometimes impossible) to reuse the code for different applications.

In order to reduce the burden in building reasoning code for each application, the context model should provide a clear consistent relationship between context elements. This then provides a common way to derive context elements and provides decisions about the user's objective in the same manner. The application only needs to know how to support the user and determine what information is

required according to the user's requirements. For example, in a conference assistant, the application needs to show the user the conference schedule with highlighted talks of interest to the user to minimise explicit user input and time to make decisions about which talks to attend.

History and Time

Context is dynamic information as it is changing all the time. To deal with dynamic information, the system may not be solely interested in the current state but also in future or past states, or changes in the state over time [Henrickson, 2003]. There are three main usages of the time and history of context. First, the history is exploited to predict users' actions from the current context. Secondly, in order to detect changes in context, the current context is compared with the previous context. Lastly, information about a user's future plans can serve as a useful type of context information. For example, a user's schedule can indicate to the system what the user's task will be in the next half hour and what the appropriate support will be for the user by the system. History and time become part of context and therefore the context model should support the use of the time and history of context.

In order to take advantage of design based on a context model and design tool that meet the requirements above, a new architecture is required. The architecture needs to support reusable context elements and changes in reasoning for different domains. It will need to meet the requirements that are discussed in the next section.

2.4.2 Requirements for an Architecture to Support Context Aware Systems

Dey [Dey *et al.*, 2001] identified a number of requirements that the architecture should fulfil to enable designers to deal with context more easily. These requirements are discussed in this section based on our main requirement for the

context model to provide a separation between consistent context elements and reasoning.

Separation of Concerns

The context model should support the context interpretation by providing clear boundaries between context elements and a clear separation between context and its reasoning. The architecture of a context-aware system should provide the ability to support these qualities of a context model.

Clear boundaries between context elements in a context model guide designers through how to group data into information for each context element. Then the designers and developers decide what sensors are available. For each sensor the developer requires to translate the sensor data into meaningful information for each context element. This forces the developer to implement the code for sensors separately from an application code so that it can translate the data before being used in application. By separating the sensor code from the application code, it will reduce the burden in writing the code for acquiring and handling context which can be a complex and time consuming process. It also supports good software engineering practices by enforcing separation between application semantics and the low level details of context acquisition from an individual sensor [Dey *et al.*, 2001].

By passing sensor data through an interpretation process, when there are new applications, the developers do not have to rewrite the code for the sensors as they can reuse the existing code to get information from sensors efficiently. Moreover, when there is a new sensor, there are no changes required in the application as it is only dealing with information in context elements. The developers just need to write the code for the sensor to acquire and handle data so that it transforms into useful information for each context element.

78

Context Interpretation

Clear boundaries between context elements in the context model guide designers through how to group the data and guide developers to implement the code for acquiring and interpreting data into the abstract level of clear boundary of each context element. The developers may need to implement code for each sensor or profile to get raw data. Then there may be multiple layers that raw data go through before information is grouped into each context element. For example, to get information about a user's environment, the lowest level may be to obtain the latitude and longitude data from a GPS. The next level may be to translate the data into a building name. At the next level, this information could be combined with the translation of raw data from a thermometer to get "it's cold outside building A" as part of the user's environment context.

Furthermore, with its consistent context elements and its reasoning process, the context model provides a common way to derive context elements and provide decisions about users to the applications in the same manner. The applications no longer have their own code containing inconsistent context elements and reasoning processes. The application only concentrates on how and what information is required to support a user. For example, the application only needs to know that the user is lost and wants to see the map and directions from building A to Reception. So, once the interpretation of sensor data for each context element is completed, the system requires another layer for further reasoning between context elements in order to derive conclusions about the user in that situation from the current context model gathered from sensors data for all applications based on the consistent context model and its reasoning. The applications only have an interface in the architecture layers to access the current context model which contains information about user's current activity and information that has influence.

From an application designers' perspective, the use of these multiple layers should be transparent. In order for the interpretation to be easily reusable by multiple applications, it needs to be provided by the architecture.

Continuous Availability of Context Acquisition

Context aware applications should be able to access the same piece of context without having to initiate individual components that provide sensor data. Therefore the architecture should support the components that acquire context executing independently from the applications that use them. The application designers then do not have to worry about instantiating, maintaining or keeping track of components that acquire context, while allowing applications to easily communicate with them.

The context acquiring components run independently of applications therefore they should be available at all times. The components must be running continuously to allow applications to contact them when needed. The components should be supported by the architecture so that first, it can be available to multiple applications continuously and secondly, the designers do not have to rewrite the code for each application.

Context Storage and History

Since context history may be used to infer future context values, the context acquiring component should be able to store a history of the entire context it obtains. Therefore the architecture must support context storage so that the analysis, interpretation and inference can be performed at any time for multiple applications. Moreover, the context history stored in the architecture can be reused for new applications. Context history can be very complex and difficult to

gather, but once it is collated, it will reduce the burden for application designers as they will be able to reuse the existing history of context.

Resource Discovery

The architecture needs to support a form of resource discovery so that it can efficiently hide the detail of where and how to acquire data from sensors in distributed computing. With a resource discovery mechanism, when an application is started, it could specify the type of context information required. The mechanism would be responsible for finding any applicable components and for providing the application with ways to access them. So instead of hardcoding the sensor in the application, the architecture's resource discovery will notify the application when there are changes in context.

Security and Privacy

The vast expansion of sensor networks and new technologies in ubiquitous computing allow designers to be able to gather various context data which may include sensitive information on people. This leads to issues of security, privacy and trust in context awareness. The availability of context information can also offer new opportunities to establish, to enhance and to manage trust, privacy and security. It is therefore important to be able to add this ability into the architecture instead of hardcoding for each application.

The requirements described above for context models and architecture motivate the research aims and objectives for this work. These aims and objectives are presented in the next section.

2.4.3 Research Aims and Objectives

This chapter discusses the lack of a common context model that takes account of a wide range of interrelated types of context and the relationships among them. Researchers have implemented context-aware systems without a common knowledge or view of context. Therefore researchers have taken account of different types of context elements in their systems. Moreover, they have embedded different reasoning processes into the context elements. As a result, it is difficult to extend existing system and typically impossible to reuse context in different systems. Based on the requirements for a context model, design tools and architecture described above, there are three main aims that this dissertation addresses. First, we - produce a ***context model*** to support designers during the design process. The objectives of using the context model are:

- To support researchers in developing a shared understanding about context. Moreover, it can then be used to support communication between designers, developers and users to have a shared understanding of context. By having a shared understanding of context in the context-aware system, it will help reduce misunderstandings about the system during design. Furthermore, mistakes made by the system during runtime can be more easily recovered by the users if they understand the underlying context model.

- To identify key elements that influence the user in achieving her objectives. For every situation for which the context-aware system may support the user, the context model facilitates developers in identifying context elements following the user requirements instead of limiting themselves to a technology-driven approach.

- To demonstrate a uniform reasoning process for the interpretation of context. Researchers can build systems based on the uniform reasoning process for top level context. By having a uniform

reasoning process, different researchers can easily extend and update the context data to suit their situations. Moreover, the consistency that the uniform reasoning process provides to the system allows the user to build an understanding of how the system reaches its decisions. This will help to recover from breakdowns during runtime.

- To show the separation between context and its reasoning, past projects have embedded different reasoning processes onto the context. This causes difficulty in extending the existing system and makes it is impossible to reuse context in different systems. The process of building context data for different situations is time consuming. For example, it is a tedious process to just build context data about the user or the room layouts for hospital. It is therefore vital that the existing context data can be reused in different situations, domains and context-aware systems.

- To represent the use of history and time. Human past experience plays an important role in the everyday decision making process. We refer to the past in order to support the way we complete our current tasks. For example, in the past we burnt our tongue by tasting boiling soup so this time we blow on a small spoon of soup and make sure it is not too hot before we taste it. For the system to process the context and reach a decision about the user's current objective, the system should be able to access the history of context and combine it with current context knowledge to improve the reasoning.

Based on the context model, we then aim to produce a systematic *design tool* to support designers during the design process. The objectives of using the design tool are:

- To provide a systematic design process for developing context-aware systems. At the moment, there are no systematic tools for designers to follow in order to build context-aware systems. It is a complicated and time consuming process to extract the context and reasoning process from the user requirements for different situations and domains. To reduce time and complication the designers require a systematic tool to transform the user requirements into context data for the system to reason and deliver the context-aware service to the user.

- To provide a design that is more consistent and extendable. Designers currently develop systems based on a specific domain and they tend to be technology-driven. The systems' capacity for extension and reuse can be limited. There is no systematic tool that encourages and facilitates them in building a reusable and extendable system.

Based on the context model and design tool, our final aim is to propose a *system architecture* that supports designs based on the design tool. The objectives behind the architecture are:

- To provide a separation of concerns. First, the sensors should be transparent to the applications. The architecture helps developers in easily changing or adding new sensors that provide the context data. As technology grows rapidly, there are new sensors available; the developers should be able to change the sensors without affecting the applications. Secondly, the architecture should support the separation between context elements and the reasoning process. By supporting this, the architecture allows the data of the context elements to be reused in different situations and domains.

- To provide a uniform structure of context interpretation. The context reasoning process is done in the architecture. Based on the design tool, the uniform reasoning process on context is constructed. Therefore the architecture should be able to support it. Moreover, by doing so it will be able to present a uniform structure of context reasoning to the user.

- To provide simultaneously available context data to multiple applications. This means that the architecture supports the access of context by multiple applications and users.

- To provide a uniform set of storage for context and its history. By gathering the context data based on the design tool, the data is represented in a uniform set of context elements. The architecture should provide storage that keeps the data in the uniform set so that it remains reusable and extendable. Moreover, the uniform set will ease the process for the application to refer to and use the context during runtime. It should provide storage for the uniform set of context history so that it can be accessed by the system during reasoning processes.

The main aim of this dissertation is to provide a context model for identifying the context elements and relationships between context elements. The context model gives the designers and developers a uniform systematic design tool for developing context-aware systems that support ubiquitous computing users in different domains. Additionally, an architecture to support this design process is introduced.

Our interests are also related to other aspects of context-aware research which we will not address other than in passing, such as:

- Work in HCI on usability of the interface to context aware systems

- Representation of the context model for user understanding

- Techniques for sensing data including reducing noise in data

- Techniques for searching and matching algorithms

Chapter 3

A New Approach to the Design of a Context-Aware System

Following on from the research questions presented in Section 2.4.3, this section will introduce the approach this dissertation takes in order to answer those questions.

Activity Theory is introduced as a potentially valuable approach to modelling the relationships amongst the elements of context that should be taken into account when designing a context-aware system. Towards the end of this chapter, the proposed context model is presented.

Existing context definitions and classifications discussed in Chapter 2 suggested that the concept of context can be very complex in mobile and ubiquitous computing. It is impossible for researchers to build a context-aware system that encompasses all of this complexity. Therefore researchers often develop a context

model as a representation or description of context in ubiquitous computing . In short, it is an abstraction or conceptual object used in the creation of a predictive formula. The next section explains the reasons for proposing a simple context model.

3.1   *Why Represent Context in a Simple Model?*

The context model should be able to identify the information that is needed within that context, together with the processes or procedures to gather or arrive at that information [Grant, 1992]. In ubiquitous computing, the user can be dealing with different devices and services at the same time. A user's activities in ubiquitous computing can be complex and modelling the context for complex activities can be overwhelming. Context can be represented as an infinite set of information. The context model could easily become too complex. Such a complex context model could be impossible for developers to use as an aid to designing and implementing the system.

Humans cannot fully understand the full moment-to-moment richness of other humans' activities, states, goals and intentions [Baldwin and Baird, 2001]. Humans often have only a simple model of the other person's intention and knowledge and beliefs, and in short their context. Yet they manage successfully and fluently to interact in many highly contextualised ways. Thus, a relatively simple model of context can enable very rich human-human interaction. It sometimes fails: each of us has experienced misinterpreting the intentions or meaning of another person. But we typically deal with such breakdowns and move on. Setting the bar higher for computers, suggesting that they should capture every aspect of context and interpret a human's intentions and meaning correctly every time, is both unrealistic and unnecessary. Therefore we suggest that a relatively simple model of the influences on users' activities may be adequate for representing context in the design of a mobile and ubiquitous system.

The context model should be simple but at the same time be able to cope with complex activity. Therefore a simple context model should not be too simplified. It should identify the necessary elements of context and the relationships between them. Moreover, a simple model has the additional advantage that it is easy to use by the designers of the system. The developers will then have a tool to help them make decisions that need to be taken in that context to infer about user's activity and implement a uniform reasoning method for the system.

As well as having a system that attempts to understand the user, the user should also be in a position to understand the system. Lueg [Lueg, 2002] argues that the user should be able to understand what the system is doing. Therefore the system should provide information about itself to the user so that the user can understand what the system is doing, why the system comes to a particular decision and what the system is going to do. Johnson-Laird [Johnson-Laird, 1983] introduced the concept of mental models as structural analogies of the world. The idea is that humans use these mental models to understand the world and how to interact with it. The mental models people create of computer systems are typically inaccurate [Norman, 1983]. Having an inaccurate model of how a system works may cause problems while interacting with the system. Many products, incorporating much research, exist which represent the state of a system to the user during run time to help the user build a better mental model of the system. As a simple example, a mobile phone shows the current state of its battery level so the user understands why it just switched itself off when the battery runs out. By showing the current state of the system in a simple manner, the system can improve the user's mental model of the system.

With a better understanding between the user and the system, the user will be able to build an appropriate mental model about the system. This mental model will allow the user to correct errors that are made by the system. The system can then

use this correction to improve its efficiency in the future. By having a simple model that is well structured with identified elements of context and the uniform relationships between them, developers can use its simplicity and consistency to provide information about context to the user in a more straightforward manner. The context from different sensors can be grouped and labelled based on the simple model. Effectively labelling data allows the end user to understand the context information. Instead of having to see the raw information from a sensor which might not make sense to them, they get to see the higher level of data i.e. information about the context element. Services and software objects can be named by intent, for example "the nearest printer", rather than by something obscure such as an IP address. Moreover, raw data from sensors can be irregular in different situations. This inconsistency can cause confusion between the user and system.

Activity Theory is introduced next as a basis for developing a simple context model that defines context elements and relationships.

3.2   *Activity Theory*

Activity Theory was developed by Russian psychologists of the former Soviet Union, Vygotsky, Rubinshtein, Leontiev and others at the start of the 1920s [Kaptelinin and Nardi, 1997]. Activity theory is a philosophical framework used to conceptualise human activities. Vygotsky proposed how tools or instruments mediate activity. Tool use influences the nature of external behaviour and also the mental functioning of individuals. Many researchers took this idea and the idea of object-orientedness and produced the first generation of Activity Theory. This first generation of Activity Theory suggested that an activity is composed of a subject and an object, mediated by a tool. A subject is a person or a group engaged in an activity. An object (in the sense of "objective") is held by the subject and motivates activity, giving it a specific direction. The mediation can

90

occur through the use of many different types of tools, material tools as well as mental tools, including culture, ways of thinking and language.

Mediating
Tools
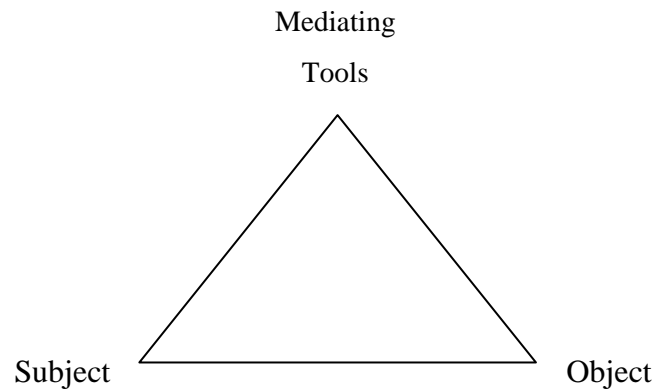
Subject   Object

Figure 3-1 Mediation between Subject and Object

In Vygotsky's early work the unit of analysis was object-oriented action mediated by cultural tools. There was no recognition of the part played by other human beings and social relations in the triangular model of action. Leontiev extended the theory by adding several features based on the need to separate individual action from collective activity [Mappin, 2000] as shown in Figure 3-2. Activities can be broken down into goal-directed actions that have to be undertaken in order to satisfy the object. Actions are conscious and are implemented through automatic operations. Operations are behaviours that have become so well learned they do not require conscious effort to execute. Operations are automatic responses to perceived conditions of the current state of the object with respect to the actions and goals that are to be fulfilled. Activity Theory maintains that the elements of activity are not fixed but can change dynamically as conditions change.

Activity   → Motives

Actions   → Goals

Operations   → Conditions

Figure 3-2 Leontiev's Model

The flexibility of the basic concepts makes them useful in describing development processes. On the other hand, it also means that it is in fact impossible to make a general classification of what an activity is, what an action is etc, because the definition is totally dependent on what the subject, object etc are in a particular real situation. We extracted the example shown in Figure 3-3 from [Kuutti, 1995]. It tries to provide an overview of how the levels of the activity, actions, operations hierarchy could be recognised in theoretical, individual-level activities.

Activity level	- Building a house	- Completing a software project	- Carrying out research into a topic
Action level	- Fixing the roofing - Transporting bricks by truck	- Programming a module - Arranging a meeting	- Searching for references - Participating in a conference - Writing a report
Operation level	- Hammering - Changing gears when driving	- Using operating system commands - Selecting appropriate programming language constructs	- Using logical syllogisms - Selecting appropriate wording

Figure 3-3 Activities, Actions and Operations [Kuutti, 1995].

As stated above, there are no firm borders: a software project may be an activity for the team members, but the executive manager of the software company may see each of the projects as actions within his or her real activity at the level of the firm [Kuutti, 1995].

One of the most important contributions to Activity Theory is by Engeström. In 1987 [Engeström, 1987] he expanded Vygotsky's mediating triangle with a social component that also mediates our action. He proposed a triangular structure of human activity. This triangular structure of human activity is based on the previous work in the Activity Theory field and the idea of the general structure of animal forms of activity as shown in Figure 3-4. The structure of the animal forms of activity consists of an individual, the natural environment and the population. Engeström adapted this structure to fit with Activity Theory as shown in Figure 3-5. Engeström supported the main concept of Activity Theory that individual's actions are influenced by their socio-cultural context and therefore cannot be understood independently of it [Little, et al., 2003]. This provided the activity theoretical community with a powerful tool for the analysis of social systems.
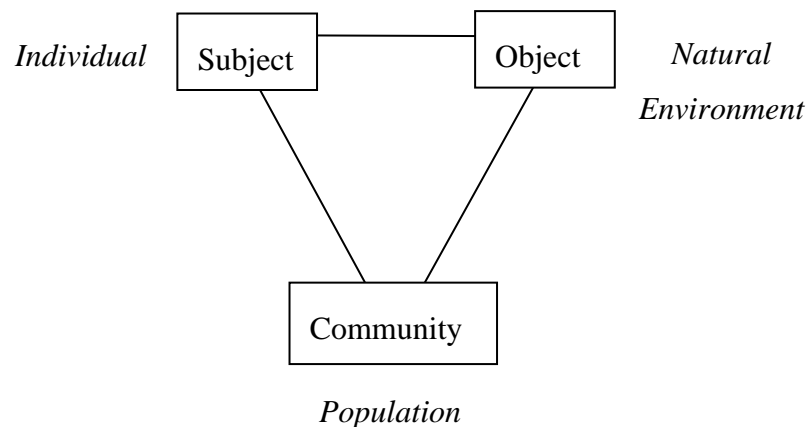
Figure 3-4 Structure of the Animal Form of Activity

The full triangular structure of human activity that was introduced by Engeström suggests that the relationship between the subject and the community is regulated/mediated by rules and that the relationship between the community and the object is regulated/mediated by a division of labour. The full structure of human activity that was introduced by Engeström is shown in Figure 3-5. To conclude, the main concepts of this model are:

- Subject: Information about an individual or a subgroup chosen as the point of view in the analysis such as user's age, sex, ability, level of experience.

- Tools: Information about tools can mean either physical or psychological tools such as a programming tool, a handbook, a PDA, language, maps, diagrams.

- Community: Information about individuals or subgroups who share the same general object such as other user's location, age, sex, job title.

- Division of labour: The division of tasks between members of the community such as different roles, rights.

- Rules: Explicit or implicit regulations, norms, conventions that constrain action or interaction such as formal rules on paper, social rules.

- Object: Target of the activity within the system. It could mean the raw material or problem space at which the activity is directed and which is transformed into outcomes such as e.g. manage the system or create a timetable for students.

- Outcome: The result from transforming the object. Ideally, the desired outcomes are the same as the ultimate objects [Gay and Hembrooke, 2004]. But if the object is not met, the outcome will be different from the object.
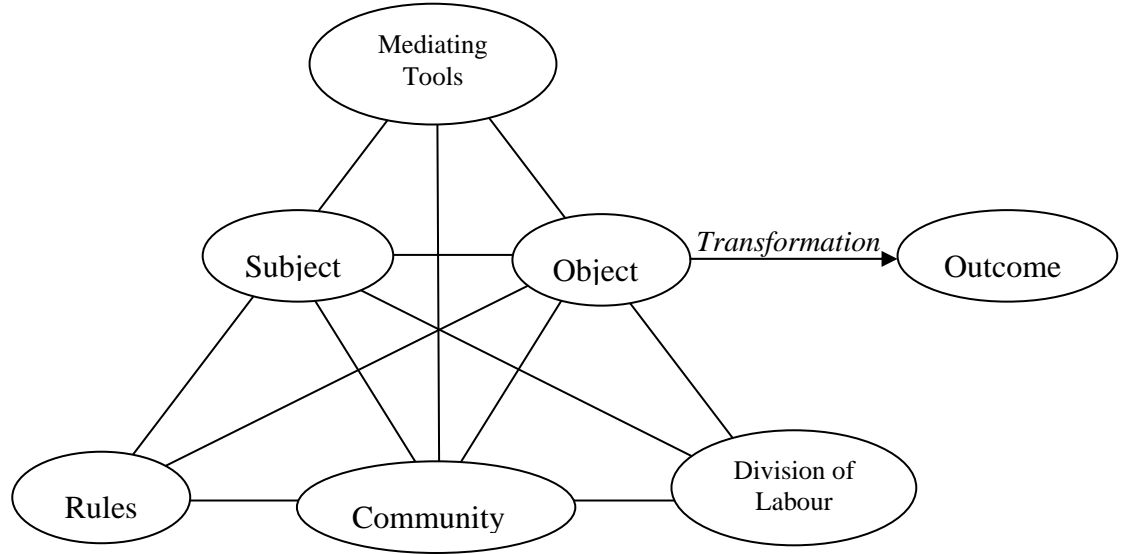
Figure 3-5 Structure of Activity Theory (Engeström)

Activity Theory provides an organised and consistent way to describe and understand the structure of human activity. Section 3.3 describes a concept that provides a link between Activity Theory and Usage-Centred Design [Constantine, 1995]. Usage-Centred Design is a model-driven process for user interface and interaction design. It relies on abstract prototypes to model the organisation and functional content of user interfaces without regard to details of appearance or behaviour [Constantine, 1998]. It provides established and effective methods for putting activity-centered design into practice.

3.3 *Activity Modelling*

Constantine encourages the use of Activity Theory. He suggests that Activity Theory and Usage-Centred Design are connected as they both represent the participation of actors in activities and the hierarchical nature of performance of activities [Constantine, 2006]. In order to make it easer for practising designers such as software engineers to represent activities, Constantine introduced Activity Modelling to capture essential insight and understanding about the context of

activity and to reflect this understanding in their designs. The aim of Activity Modelling [Constantine, 2006] is to create an easily grasped *modelling language* anchored in a consistent, coherent vocabulary of well-defined concepts that link task modelling based on essential use cases [Constantine, 1995] to the established conceptual foundation of Activity Theory. As a result, Activity Modelling extends Usage-Centred Design by introducing new notations which are related to the Unified Modeling Language (UML) used in software engineering [Fowler and Scott, 1997]. The new notations are action, activity, player and tool in order to take advantages of the three level hierarchical nature of activity, community, tool in Activity Theory.

Figure 3-6 shows the summarised notation for Activity Modelling. With the new notations, the additions and alterations have been made in Activity Model (which includes Activity Map, Activity Profiles and Participation Map), Role Profile and Task Model to incorporate systematic Activity Modelling into Usage-Centred Design.

Symbol	Name	Description
	actor, user actor	activity participant interacting with the system of reference
	role, user role	relationship between an actor and the system of reference
	system actor	non-human system (software or hardware) interacting with the system of reference
	player*	activity participant not interacting with the system of reference (but often an actor with other systems)
	artifact, tool*	any artifact employed within an activity
	activity*	collection of actions or tasks undertaken for some purpose
	task, task case	action by an actor in interaction with the system of reference for some goal within an activity
	action*	action by a player for some goal within an activity

* new notation introduced for activity modeling

Figure 3-6 Extended Usage-Centred Design Notation for Activity Modelling
[Constantine, 2006]

Based on the requirements for the context model discussed in Chapter 2, the next section presents the rationale for using Activity Theory in this dissertation.

3.4 *Reason for Using Activity Theory*

As shown in Chapter 2, there are a multitude of classification systems (See Table 2-1). Researchers have tried to classify context into different elements that have influence on a user's activity in the ubiquitous computing world. Previous classifications cover different elements of context for reasoning about human behaviour but named them differently. Some groups are overlapping. Together they cover elements that have influence on user behaviour. Therefore we need a

new model that covers all the key elements so that the system can have a better understanding about user. From Table 2-1, we identify 5 high level categories of concepts that we need to take into account in modelling context to minimise the repetition and a too specific classification. As shown in the columns in Table 2-1, Location and condition are too specific in this content to classify them separate from each other.

Therefore we would like to group them together and called Physical Environment as many researchers have done in the past. Computing Environment and Device Characteristics could be grouped together in order to make the classification simple because they both contain information about technology or tools that are available for user in the ubiquitous computing. Information about user and user activity are grouped together because this information is related and provides information about particular user. Social is important in the ubiquitous computing as users are in different society. Therefore social should be separated from the human factor. It is hard to capture therefore the social information is hardly been used in the past applications. To conclude, I propose that the context model should cover these elements:

- Human factor contains information about user (such as mental state, habit, preference) and user's action.

- Physical environment is separated from the computing environment. This is because of its differences in features. Therefore the way it captures and reasons will be different. Moreover the impact on user behaviour is different.

- Technology is to group all information about the devices (not limit to computing devices) and computing environment

- Social is separated from human factor because it contain information on the relationship between user and other users that will be capture and process different from human factor. This will result from the human factor?

- Time is for keeping the history of context.

Moreover, the relationship between each element of context is unclear. We would like to combine these similarities and differences to develop an adequate theoretical model, which is currently lacking in this field. In this case, an adequate theoretical model means that this model can be applied to any real simple or complex situation in the ubiquitous computing world. It should cover key elements of context that influence user activity. Moreover, it should be able to explain how elements influence the user's activity in any real situations. This model can then be used by the system to better understand the user. It will also have potential to improve communication between researchers in the field and promote shared understanding.

A classification system is needed to help the system use context to build a conceptual model of user activity. Therefore we want to introduce a theory that describes the relationships between the elements that have an influence on human activity. There are several concepts for understanding human activity or tasks such as Activity Theory [Engeström, et al., 1999; Rogers and Scaife, 1997] and Task Analysis [Preece, et al., 1999]. For the purpose of classifying context, Activity Theory is chosen as it has the main characteristics described below.

3.4.1 It Provides a Standard Form for Describing Human Activity

There are several studies of modelling human activity such as Activity Theory, Task analysis, HTA. Activity Theory provides a simple standard form for modelling human activity whereas Task Analysis, for example, does not. Activity Theory treats activities as an ongoing process with a stable structure involving people, a motive or "objective" and the tools that they use.

With techniques such as Task Analysis, the modelling of human activities can be flexible in order to model the complexity and contingency of tasks in reality [Mori, et al., 2002; O'Neill and Johnson, 2004; Paternò, 1999; Van Der Veer, et al., 1996]. Aside from the simplistic hierarchies sometimes used in, for example,

HTA [Shepherd, 1989; Shepherd, 1998], there is no fixed form for task modelling in Task Analysis approaches. HTA is useful for interface designers because it provides a model for task execution, enabling designers to envision the goals, tasks, subtasks, operations and plan essential to users' activity. HTA is useful for decomposing complex tasks, but has a narrow view of the task and is normally used in conjunction with other methods of task analysis to increase its effectiveness [Crystal and Ellington, 2004]. HTA does not provide a uniform tool that supports the designers to decide which elements have an influence on each task.

In modelling context for context-aware system design purposes, we argue for using a simple standard form to model the aspects of human activity that are associated with key elements of context and their relationships. Although a simple standard form cannot represent the full richness and complexity of human activity, it does not have to. As humans, we cannot and do not form complete models of other humans' context, especially with regard to their internal goals and intentions. Despite using partial and simplified models, we manage to communicate and collaborate with our fellow humans very effectively and efficiently. As noted above, from time to time we do get it wrong and, for example, misinterpret another person's intention or meaning. We then invoke repair mechanisms and feed the information generated through this experience into our future models. Since humans manage so well with relatively simple and partial models of other humans' goals and activities, it is both unreasonable and unnecessary to demand more of computer-based context models. Activity Theory provides a suitable simple model in a standard form.

3.4.2 It Provides a Representation of the User

Activity Theory emphasises the importance of including a representative user in the activity that the designers are concentrating on. It takes into account that

information about the user has a large influence on the activity. This is important in context awareness as the properties of users the context-aware system is attempting to support have an impact on the system's reasoning. The context reasoning in the system obtains information about a user to increase the efficiency of inferring the user's objective and so provide suitable services. By including a representative user in the model, the model directs the designers to take the user into account during the design process.

3.4.3   It Relates Individual Human Activity to Society

In a ubiquitous computing world, users are not isolated workers at a desktop, in one location.   The 'traditional' use of computers is increasingly being complemented by residential and nomadic use, thus penetrating a wider range of users' activities in a broader variety of environments and societies such as the school, the home, the market place and other civil and social contexts [Stephanidis, 2001].   Users access computing services within society and that society will have an influence on the user's behaviour.   As a result of being in society, users have roles in society. They decision of performing activity is driven by their roles such as secretary or dad. At the same time, they perform the activity under a set of rules that constrains their actions*, for example, only using a company account when he holds role as secretary or put expense on his personal account when he holds role as dad.* For example, the user may act as a secretary at work using his mobile phone to book a flight for the boss with a company account and he may also act as a dadat home with his family using the same phone to book a holiday for them from his personal account.

Therefore the context classification should allow the system to take account of what can have an impact on human behaviour within society. Activity Theory explicitly takes society into account in its modelling.

3.4.4   It Provides a Concept of Tool Mediation

Ubiquitous computing users may use multiple devices to access information or services, thus dealing with different screen sizes and interaction methods such as touch screen PDAs, laptops, mobile phones and wearable computing. Moreover, the availability of services is changing all the time as one service may be available at one situation and may not be available in the next. Therefore the tools and services may be changing all the time. Characteristics of tools and services have an influence on users' behaviour in completing their activities. Activity Theory explicitly includes this in its modelling. It provides a framework for understanding the cyclical relationship of application and evaluation as a user applies a tool to accomplish a goal.

3.4.5   It Maps the Relationships amongst the Elements of a Human Activity Model

Activity Theory maps the relationships amongst the elements that it identifies as having an influence on human activity. This provides us with a potentially useful way to classify and relate the elements of context and maps them very closely to the key elements of context. The relationships between the elements are important in helping the inferring process to be manageable in a uniform manner.

These five reasons illustrate that Activity Theory is satisfactory for use in attempting to develop a context model that meets the user requirements mentioned in Section 2.4.1. The next section describes how Activity Theory meets the context model requirement of representing the history of context. The importance of history in the context model is also discussed.

3.5   *History*

Leontiev [Leontiev, 1979] proposed that activity is *not a reaction or aggregate of reactions*, but a system *with its own structure*, its own internal transformations, and its own development. As mentioned above, Activity Theory structure breaks down into three levels: activities, actions, and operation. The basic structure is shown in Figure 3-7. Operations become routinised and unconscious with practice and they depend on the conditions under which the action is being carried out. That means that operations are situated in or related to the world by an unconscious orientation basis established through *experience* with the conditions or constraints for the operation [Rodriguez, 1998].

With reference to internal transformations, Activity Theory [Kuutti, 1995] suggests that activities are not static or rigid entities; they are under continuous change and development. This development is not linear or straightforward but uneven and discontinuous. All levels can move up and down. For example, an operation can become an action when "conditions impede an action's execution through previously formed operations". Actions can become operations though *experience* of performing actions in the past. When the user performs actions so many times, they become automatic and are no longer performed consciously. This is when an action transforms to an operation. This means that each activity also has a history of its own. Part of the older phases of activities often stay embedded in them as they develop, and historical analysis of the development is often needed in order to understand the current situation. Thus, history is an important element in Activity Theory as it influences the internal transformation.

In ubiquitous computing , context awareness supports the user at different levels. By using history, context awareness can improve usability by supporting the action level instead of the operation level when users have experienced the operation level several times in the past. For example, at an early stage, the context-aware

system supports the user by automatically filling in parts of a form such as the name and address before the user explicitly submits the form. After a while, the user becomes familiar with how the system pre-fills the form and how the system recognises that the user usually agreed with the pre-fill information and explicitly submitted the forms in the past. The system then transforms the action of submitting the form to operation by automatically submitting the form for the user. This shows that the history helps the system support the user in structurally different levels of interaction.

Activity   Motive

↕   ↕

Action   Goal

↕   ↕

Operation   Conditions
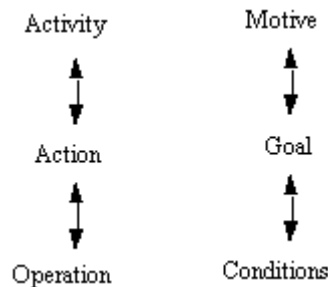
Figure 3-7 Basic "Structure" used with Reference to Human Activity

Chalmers [Chalmers, 2004] discussed the history aspect of context and how it is important for humans in referring to their current task. Through experimenting with different applications such as Seamful Game, he suggested that, with experience of its use, the tool may become understood and familiar to the individual, i.e. more ready-to-hand and embodied. Therefore he sees significant potential in making more use of the past in context–aware systems design.

Moreover, people often refer to experiences in the past while performing their current activity, using such experiences to guide their current actions. For example, in the past, a user got a virus from an email received from a certain email address. When the user receives a second email from this email address, the user deletes the email from the email address instead of opening it.

An increasing number of context-aware projects [Hariharan and Toyama, 2004; Helander, 2005; Kröner, et al., 2006; Mayrhofer, 2005; Mohr, et al., 2005] pay attention to the use of history in their reasoning process for particular sensor data, applications or domains. Instead of a fixed rule to recognise the situations that the system should support the user, the history of situations is used to infer about the situation. Several projects [Mayrhofer, 2005; Mohr *et al.*, 2005] have studied the inference algorithms that take history of context into account in order to improve the inference mechanisms. However, these studies were conducted for particular domains and applications. The modelling of context history is still in its infancy. There are no guidelines for researchers to decide what to take into account in context history.

History is a critical part of context. A few previous context-aware projects have considered time as context. However, they have typically looked at time simply as current time that can be sensed from the device. For example, they compare current time to the user's timetable and provide support for the user's current task in her timetable [Agarawala, et al., 2004; Hertzog and Torrens, 2004]. Time is used to sequence the events in order to be able to compare the interval between them. It has a direction where past lies behind while the future lies ahead. Therefore time is crucial for recording the events in history. It is used as a reference point for the events in the context history.

The next section presents the proposed context model to meet the requirements described in Section 2.4.1.

3.6   *Proposed Context Model*

Researchers in context awareness refer to context differently. In order to further the field of context awareness, researchers should consider context in the truly ubiquitous computing world instead of only part of the context that is available through current technology. Moreover, researchers, developers and users could benefit from having a model that aids shared understanding of context. This section introduces a proposed context model that aims to support designers during the context-aware system design process. It facilitates bringing researchers together through having shared understandings of context. It underpins context-aware system development by providing a uniform relationship between consistent sets of context elements. Ultimately, it provides a consistent representation of the system's model of context to users. With these features, it potentially bridges some of the gaps between researchers, designers, implementers and users.

3.6.1   The Context Model

The main objective of a context-aware system is to support a user's current activity. In order to support the user, the system has to know what the user's current objective is. In other words, it needs to know the answer to the question: "What is the user is trying to do?" Even humans with all their senses can find it difficult to know another human's current objective. Human reasoning is complex and impossible to build into the context-aware system. It is important to be aware of the complexity and to opt for a simple systematic model to represent the reasoning in the system. The simple systematic model will allow the developers to implement the system more easily. Moreover, by using the systematic model, it will allow a user to understand the system more easily than a system with a complex or unsystematic model. The consistency of the system will help the user to build mental model of the system easier and improve usability by minimizing the user's cognitive effort. This understanding will allow a user to be able to correct the system when mistakes occur.

People often refer to experiences in the past while performing their current activity, using such experiences to guide their current actions. Chalmers [Chalmers, 2004] notes a range of research that refers to activity as an ongoing temporal process of interpretation. He found significant potential in making more use of the past in context-aware system design. The features of Activity Theory provide key elements that exert influence on human activity. However, although Activity Theory captures key elements of human behaviour, the Activity Theory model only captures information about the user's current situation or context and the outcome when the current activity is performed. The Activity Theory framework argues that activities are under continuous change and development. It suggests that the user's experience has an influence on transforming between an operation, action and activity. However, it does not provide an adequate account of a user's current object or intention, or of the user's past actions and contexts in the uniform model. Without a systematic representation of past, present and future in the Activity Theory model, the context-aware system may find the model is difficult to use to infer about a user's current objective by referring to the history because the history does not exist in the systematic model.

For the context model to be used to support uniform and systematic reasoning about context in complex situations, it should not only be able to identify the key elements of human behaviour and relationships between them, it should also be able model relationships between the past, present and future behaviours in the systematic model.

To represent the history element, we add a temporal dimension, a "timeline", to the Activity Theory model (see Figure 3-8). The timeline includes not just current time, but also past time (that contributes to historical elements of the context) and future time (that allows for prediction of users' activities from the current context). Activity Theory is used in our context model to analyse information about a

particular user so that it concentrates on one point of view of the user and her individual level of hierarchy. This is because the activity of one user can be an action or operation of other users. Therefore the timeline in our context model represents reference points of time at which the particular user engages in achieving the objectives. Through the addition of a timeline to the Activity Theory model, the context model can represent the history of context for the user.

Our extension to the Activity Theory model provides the basis for a systematic context model (see Figure 3-8) to be used as a design tool to aid designers and developers in building a shared understanding of context. It helps make design issues explicit and forms a basis for design choices. It also encourages the designer to focus on aspects of the system affecting usability. Time is a crucially important part of context.
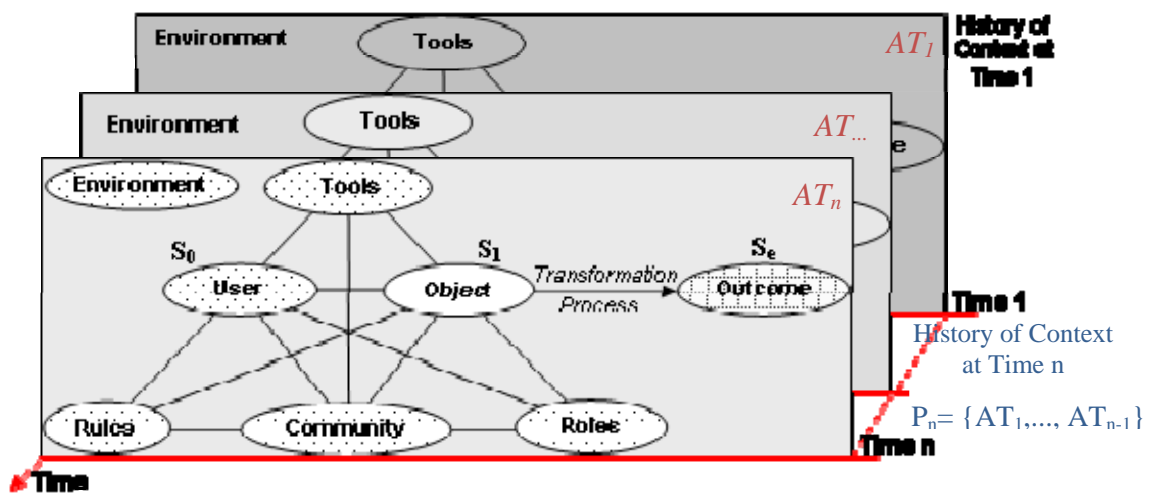
Figure 3-8 Proposed Context Model Adapted from Activity Theory

In our extended Activity Theory model, history is modelled as a set of states in the past (**P_n**). A state is an event when the system identifies a user's objective. During design stage, these states are gathered from the scenarios in the user's

108

requirement by the developers and stored in the history of context. During run time, a state is normally identified from system constantly accesses the related sensor devices and profiles to get context elements in the context model ($\mathbf{AT_n}$). The model is then compared with previous model ($\mathbf{AT_{n-1}}$) that was stored a moment ago. The differences between the two models will originate state, which means the switching of scenario. Each past state at time n is represented as an Activity Theory model ($\mathbf{AT_n}$),, which captures the context of activities in the environment at that time. This information includes the initial state ($\mathbf{S_0}$), object or intention ($\mathbf{S_1}$), and outcome or end-state ($\mathbf{S_e}$) of the activity. The initial state ($\mathbf{S_0}$) includes the current information about environment, time, user, tools, community, rules and role, which is gathered by the system through transforming data from sensors and profiles. The object or intention ($\mathbf{S_1}$) models information about the user's current objective, i.e. what the user is trying to achieve. This information about user intention ($\mathbf{S_1}$) can be inferred from the history of context ($\mathbf{P_n}$) and the initial state ($\mathbf{S_0}$). In order to infer the user's intention ($\mathbf{S_1}$) during run time, the context elements found from sensors and profiles in the current context model ($\mathbf{AT_n}$) are then compared to the context elements in different states in the history of context ($\mathbf{P_n}$). If the system found the most matched of the initial state ($\mathbf{S_0}$) in current model and the one in history of context, the user intention ($\mathbf{S_1}$) in the model in the history of context is assigned as user intention ($\mathbf{S_1}$) in the current context model ($\mathbf{AT_n}$). Once the user has performed the activity, we have information about the real outcome ($\mathbf{S_e}$). Then the initial state ($\mathbf{S_0}$), intention ($\mathbf{S_1}$) and outcome ($\mathbf{S_e}$) are stored as context model at time n ($\mathbf{AT_n}$) and become part of the history of context ($\mathbf{P_{n+1}}$). It will be used to infer the user's intention or goal in future situations. By adding timeline to the Activity Theory, it allows us to represent past, present and future in the context model. Therefore the history can then be used to infer about user's current intention through a consistent model.

For each slice of the Activity Theory model, in order to reach an outcome it is necessary to produce certain objects (e.g. experiences, knowledge, and physical

products). As mentioned in Section 3.4, human activity is mediated by artefacts (e.g. tools used, documents, recipes, etc.). Activity is also mediated by an organisation or community. Also, the community may impose rules that affect an activity. The user works as part of the community to achieve the object. In the society, an activity normally also features a division of labour or role. To be used in the context-aware system design, the definitions of context elements on each slice of Activity Theory model that influence a particular situation can be described as follows:

User: For our current purposes, information about user that the context model is supporting. Information may include name, preference, schedule, devices, etc. *(Answers: Who is the user that the context-aware system is supposed to support?)*

Environment: Information about the physical and virtual environment that has an influence on a user's activity in the situation *(Answer: Where is the activity achieving?)*

Time: For our current purposes, time is the occurrence of events in the past, the present and the future. Each point of time shows the occurrence of the user achieving the objective or goal. *(Answer: When is the activity achieving?)*

Tools: Information about the tools that are available to user and their availability, including device characteristics, public services – applications, and computing environment such as network availability. *(Answer: What are the tools supporting the user to complete the activity?)*

Community: Information about people around the user (in both the physical and virtual environments) that may have an influence on her activity. The community

can be referred as particular set of known users (such as the community contains Jenny and Paul in the situation) or a group of unknown users (such as there are more than 5 other people in the situation). *(Answer: Who are the people influencing the process of completing the activity?)*

Role: Roles of the user in completing the objective of the activity in that situation including who can perform which tasks on the object. *(Answer: What is the role of the user in the society?)*

Rules: Norms, social rules, policy and legislation within which the user relates to others in her community. *(Answer: What are the rules restricting the user in the current society?)*

Object: The user's intention and objective of what activity that user wants to perform. The system uses all the elements above to decide on the user's intention or objective. *(Answer: What is the objective of the user to complete the activity?)*

Outcome: This is the result of the user's activities, which may or may not achieve the objective. *(Answer: What is the result from the activity that user is performing?)*

The context model contains consistent context elements and relationships between the elements for the designers to refer to during the design process. Moreover, adding a timeline to the model provides a systematic way to represent the past, present and future context of the user. We intend this systematic context model to help support designers in developing context-aware systems driven by user requirements rather than by the availability of particular technology. During design stage, the designers concentrate on the user's requirement scenario and

111

using the elements in the context model to extract the required context information for each context element in the scenario rather than availability of the technology. For the information that cannot be gathered from the sensors, the designers develop profiles which are stored in the database to hold information that the system can refer to during run time. The system therefore can access the information that cannot be gathered from the sensors rather than ignoring information that has influence on user's activity.

3.7   *Summary of the Proposed Context Model*

Context awareness requires a systematic context model that represents context in a ubiquitous computing situation. The proposed model can help bring researchers together by bridging the varieties of different context elements that different researchers utilise in their context-aware systems. In context awareness, the context model is used to infer a user's current objective. The inferring process in the system utilises context information to make decisions about the user's objective. The process acts somewhat like human reasoning. However, human reasoning is so complex that a complete understanding of how it works does not actually exist. It is impossible to build a context-aware system based on this complexity. A simpler systematic context model is therefore introduced for designers to develop inference processes in context-aware systems.

Humans make their decisions based in part on their past experiences. History plays an important part in user's decision making. Therefore the context model not only has concise constructs for presenting the current context but it is also important for the context model to represent history of context. In this dissertation, the systematic context model is developed by adding a temporal dimension, a "timeline", to the standard Activity Theory model. Activity Theory provides a consistent set of context elements and relationships between them which supports researchers or designers in having a shared understanding about

context. Moreover, the timeline systematically adds representation of the past, present and future context. Designers can use the context model as a reference to decide what types of context the system should use to infer about a user's current objective. The simple systematic model allows the developers to be able to implement the system that infers the user's current objective based on the current context and history of context that has been stored in a common format according to the timeline in the model. The consistent context reasoning provided by the context model is also intended to make it easier for the eventual user to build a mental model of the system because of its consistency. Therefore the user understands the system and is able to correct mistakes, for example in inference, made by the system during run time. The design process introduced by the context model will be discussed further in Chapter 4.

Chapter 4

Turning the
Context Model into a
Design Tool

The previous section described the proposed context model in which we extended Activity Theory to provide a consistent and structured way of representing context for context-aware mobile and pervasive systems. This chapter will explain further how this context model can be used during design as a design tool enabling a new design process for context-aware system design.

With its consistency and structure, the context model can be used as a design tool to aid designers in building a common understanding of context. It helps make design issues explicit and forms a basis for design choices through a consistent set of context elements and relationships. The context elements provide a set of consistent vocabulary that encourages the designers to focus on aspects of the system affecting usability rather than the availability of technology.

The context-aware system requires a set of context values stored in a database to be used by the system during runtime to recognise the current user's objectives. The context values are a set of information for each context element which has been assigned to the context model. During design stage, the context values are extracted from the user's requirement scenario. For example, the context value for the environment in the scenario can be a room number in a building or town name. In addition to the sets of context values available from sensors, profiles are stored in the database. These profiles hold information that cannot be gathered from the sensors and are generated by the designers, such as information about user's preferences where user has filled in the form during registration. The context model helps the designers to generate a set of such values in the database to act as a practical structured model from the descriptive situations in the scenario. By concentrating on usage scenarios and user requirements, the context model guides the designers to create profiles in the databases. Moreover, the relationships help designers generate consistent real time reasoning process using the profiles and history of context models in order to infer the values for context elements that cannot be found from sensor data and the user's current objective. The six steps in the process of using the context model as a design tool in context-aware system design are discussed below.

4.1   *Step 1: Define Scenarios in which the System will be Applied*

User requirements in ubiquitous computing have mainly been determined from analysing the problems in scenarios [Derntl and Hummel, 2005; Dong, et al., 2006; Gellersen, et al., 2002] or field studies [Desmet, et al., 2007; Jiang, et al., 2004] in different domains. Both scenarios and field studies give the designers descriptions or stories of people and their activities in different domains. For example, on Monday morning, Jane R is running a bit late for her meeting. She wishes to show the presentation on the projector in the meeting room in Building 1 West at the University of Bath whilst she is walking into the room. At the same

time she wishes to access a memo on her presentation in a folder on her laptop computer. However, the folder is covered up by a presentation that Jane wishes to refer to while reading the memo. The presentation is so large that it nearly fills the display. Jane pauses for several seconds, minimises the presentation, finds the desktop that is connected to the projector on the network and sends the presentation to the desktop, opens the memo on her laptop, and starts presenting.

Designers use scenarios and field studies to analyse how technology could improve a person's ability to complete tasks and create scenarios to show how the technology can support people. Carroll [Carroll, 1999] suggested five reasons for using scenarios during design. First, scenarios evoke reflection in the content of design work, helping developers coordinate design action and reflection. Secondly, scenarios are concrete and flexible, helping developers manage the rapid changes of design situations. Thirdly, scenarios provide multiple views of an interaction, helping developers manage the many consequences caused by any given design decision. Fourthly, scenarios can also be abstracted and categorised, helping designers to recognise, capture, and reuse generalisations, and to address the challenge that technical knowledge often lags behind the needs of technical design. Finally, scenarios promote work-oriented communication amongst stakeholders, helping to make design activities more accessible to the great variety of expertise that can contribute to design, and addressing the challenge that external constraints, designers and clients often distract attention from the needs and concerns of the people who will use the technology. Ubiquitous computing users may not necessarily have a full understanding of the system as it might be very new to them. Scenarios not only provide ideas to designers but also may be useful in order to give a better understanding to users about this field that is still in its infancy. Designers are therefore suggested to transfer scenarios or field studies into context-aware scenarios which describe how context awareness could support the user in different situations.

4.2   *Step 2: Define Situations in which Context Awareness Can Support Users*

As the scenarios and field studies are descriptive stories, the designers have to break the stories into smaller situations in order to have a better understanding of users' requirements. Each situation is normally defined by how each activity is carried out by the user. The designer can then decide how context awareness can support the user to complete each activity in each situation. By having smaller situations that concentrate on each activity in scenarios or field studies, it becomes easier for a designer to concentrate on the activity that needs support from context awareness. These situations are used as a guide for storing a set of basic activities in the history of context. This set of basic activities is used by the system to recognise the state and trigger event where the context-aware system should support the user with relevant services or information. The history of context is then used in real time for inferring the user's current objective from sensor data.

After the scenarios are gathered, for every scenario the designers need to extract the situations for the context-aware system to support the user. As noted in the previous chapter, the context model is used to concentrate on the user that the system is supposed to support. For each situation that has been extracted from the scenario, the concept of three levels of activity in the Activity Theory is referred to in order to gather the activity, actions and operations. Following Leontiev's model (Activity - Actions - Operations), our design tool has six activity level questions for the designers and developers to follow for each situation. It has introduced a systematic 6 question guide for the designers to use for analysing each situation. The first 3 steps are:

Question 1:   What is the activity for the context-aware system to support in this situation?

Question 2:   What are the actions that a user may need to perform?

Question 3:   What are the operations that a user may need to perform?

In Question 1, designers attempt to answer why the user is in this situation so that the activity may be used to identify the objective from the situation. For example, from the situation of Jane R, the objective of the situation is for Jane R to show the presentation on the projector and see her memo on her laptop on time. Therefore the activities in this situation are "showing the presentation on the projector and seeing the memo on the laptop".

In order to gather the actions for the activity in Question 2, the designers consider goals and sub-goals of the user in the situation. The designers then attempt to draw up a list of actions that the user is required to perform in order to meet the goals and sub-goals. For example, the actions in Jane R's situation are searching for folders for presentation slides and the memo, searching in the network for the projector, and opening the slides and memo.

From the actions in Question 2, the designers consider the set of operations that the user performs unconsciously under the conditions in order to meet the actions. In other words, the operational structure of the activity is typically automated and is not a conscious concrete way of executing an action in accordance with the specific conditions surrounding the goal. For example, the operations in Jane R's situation are double click to open folders for slide and memo, resize the folders, and copy/paste the slide file. These have become operational because Jane R has become used to opening the folder by double clicking. She does not have to think consciously about how to double click on the folder. She only has to make sure that she has the permission to access the folder and the window of the folder is not opened too large so that it overlaps the other folder or is over the screen size.

These questions allow the designers to identify the three levels of activity in the situation. As a result of applying these three questions, the lists of activity, actions

and operations may be represented using a practical model driven approach such as Activity-Task Map from the Activity Modelling approach [Constantine, 2006] described in Section 3.3.

From the lists of activity, actions and operations, the designers decide at what levels that the context-aware system should support the user. The difference in levels represents the user's level of awareness in order to perform an activity, action or operation. The level of awareness can help the designer not to take control away from the user completely.

According to Chen and Kotz [Chen and Kotz, 2000] classification, there are two types of context-aware computing: *active* and *passive*. The classification is based on the ways that context-aware computing supports the users. *Active context-aware* computing automatically adapts to discovered context by changing its behaviour whereas *passive context-aware* computing presents new or updated context to an interested user or makes the context persistent for the user to retrieve later. An example of active computing is context sensitive conference schedules that highlight/narrow down the track that might be of interest to a user. The designers assign a schedule application and the selected context is information about the user – i.e. an element of user context. An example of passive context-aware computing is that the conference assistant shows information about the current colleagues' locations or tourist attractions on a map according to the user's current location and time.

For situations where designers decide context awareness is required to support the user, the designers have to assign application features (Active, Passive or both) and information that the application might need to support the user in order to complete the activity more efficiently based on the classification of context-aware computing. Thus, the breakdown lists of activity, actions and operations

120

encourage the designers to concentrate on the effect of context awareness at the level of user interaction and usability.

From the application features of Active and/or Passive introduced by Chen and Kotz, the lists of activity, actions and operations are revisited in order to assign different types of support from the system. The lists offer ideas of where and how the system should support the user and where the user should have control over the system. The designers adopt the idea of active and passive support to get rid of time-consuming or unnecessarily explicit inputs that interrupt the ubiquitous computing user to complete the main activity. Furthermore the designers could consider replacing the user's explicit input with the system based on the system's ability to complete them better and quicker than a human with its features such as finding a file in the database or finding matching words in a file. Therefore the next design questions address the support that the system should provide to the user, as shown below:

Question 4: What operation level supports is the system going to provide?

- Double click to open folders for slide and memo >> Active

- Resize the folders >> Active

- Copy/paste the slide file

Question 5:   What action level supports is the system going to provide?

- Search in the network for projector >> Active

- Search for folders for slides and memo >> Active

- Open presentation slides

- Open the memo

Question 6:   What activity level support is the system going to provide?

-       Show the presentation on the projector and memo on the laptop

As mentioned that during design stage, the designers extract context models from user's requirement scenarios, these context models are stored in the database as a reference to infer user's current objective in real time. Once the objective is inferred, the system selects the support for the user according to the assignment done during this design stage. For example, the application should be *Active* so it can automatically show the combination of "a folder of the memo", "the presentation" and "a shared folder" on the desktop in the meeting room on Jane R's laptop. Hence, Jane does not have to resize the windows and go through the process of searching for the folders and trying to detect a public desktop that is connected to the projector in the meeting room on the network. Jane can then explicitly select the right files in the folder. She can then transfer the presentation file to the shared folder on the desktop and thereby show the presentation on the projector and see the memo on her laptop more easily and quickly.

For each situation, answering these six questions provides the designers with guidance on how and at what level the system should support the user. The next step describes how the context model gives the designer a consistent and non technology-driven way of modelling the context for each situation. The context model provides a consistent set of vocabulary for the designers to consider about grouping the information in the situation. The predefined context model of each situation is stored in the context history. These are then used by the system to infer about user's current objective. The inference process allows the system to recognise state and trigger events where the context-aware system should support the user with relevant services or information. This is described in section 5.1.1.

Meeting room, Building West at University of Bath

Tuesday 5 Jan 2006 9:10am

Projector on Desktop

Jane R

Present slides on projector and see memo

Transformation

Process

Presented slides on projector and see memo

Do not show

Attendants

Presenter
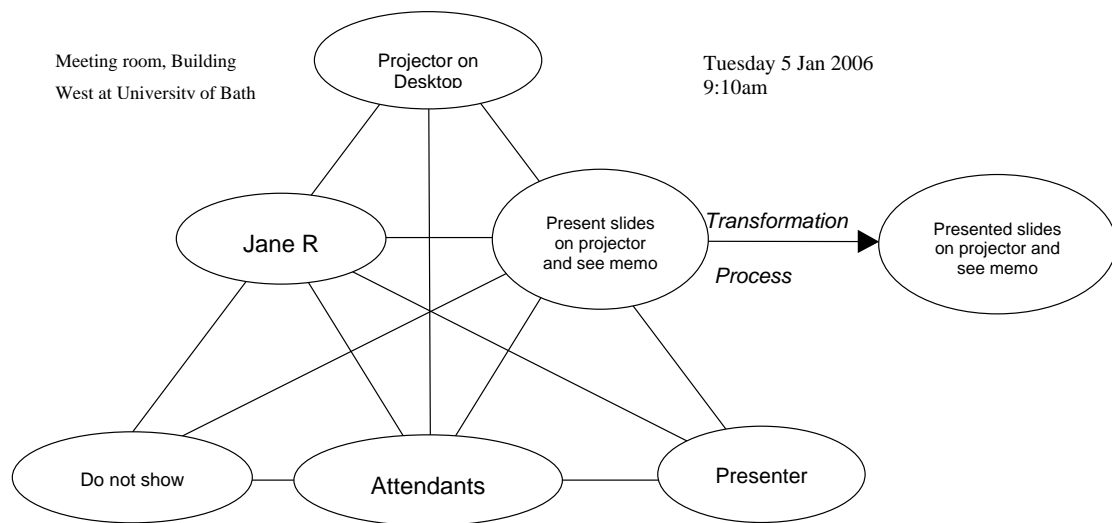
Figure 4-1 Jane R's Situation Extracted into Context Model

4.3   *Step 3: From the Situation to Elements in the Context Model*

As the scenario is broken down into smaller situations that describe the user and activity for that situation at a particular time, the designers can then identify the types of context information for the situation.   The information that has an influence on a user to complete the activity is taken into account from the situation.   The designers then group the information in the situation into information for each element of the context in the slice of context model for that particular time.

As mentioned in Section 3.6, there are nine key abstract elements of context for each slice of the context model (environment, time, user, community, tool, role, rule, objective, and outcome).   In this section, the designers follow the definition of each element in the proposed context model to make decisions about each element of context.   Regarding each situation, the designers answer the nine questions below:

1. User element: *Who is the user that the context-aware system is supposed to support?*

For the particular user, other elements in the model are identified according to the user;

2. Environment element: *Where is the activity being performed, both physical and virtual?*

3. Time element: *When is the activity being performed?*

4. Tools element: *What tools are supporting the user to complete the activity?*

5. Community element: *Who are the people influencing the process of completing the activity?*

6. Role element: *What is the role of the user in the current community/society?*

7. Rules element: *What rules restrict the user in the current community/society?*

8. Object element: *What is the objective of the user to complete the activity?*

9. Outcome element: *What is the result from the activity that the user is performing?*

Table 4-1 shows how the context model transforms into a simple table that designers can use to refer to the context model and assign values in the situation to each element in the context model. The designers use the definition of each context element and the nine questions above as a guide to decide on what value of information should be in each context element of the context model. As a result, the designers group the information about the situation into nine categories following the nine key elements of the context model.

By following the questions, the designers use the table to model possible context for the situation. The table and questions give the designers a flexible but

systematic way for considering possible context models for the situation that the system can use to support the user. As a result, the designers can assign many possible context models to be stored in the history and to be used by the system to trigger events the system can use to support the user during run time [see Chapter 5]. Table 4-1 shows examples of values extracted from Jane R's situation.

Context Elements	Definition	Values from Situation
Environment	Information about physical and virtual environment that has influence on the user's activity in the situation.	Meeting room 2.4 in Building A, University of Bath, Bath, Avon, UK
Time	Point of time that the situation is occurring.	Monday 5 Jan 2007 9.10 am
User	Information about user including identity, preferences, schedule, devices.	Jane R, who has a Laptop and N95 mobile phone
Tools	Information about tools that are available to the user and their availability, including device characteristics, public services – applications, and computing environment.	Printer 1 no queue in meeting room 2.4 Desktop 2 connected to the projector in meeting room 2.4 Wireless network Room booking schedule
Community	Information about people around the user (in both the physical and virtual environments) that may have an influence on user's activity.	3 attendants including Attendant A, Attendant C and Attendant D
Roles	Roles of user in completing the activity in that situation including who can perform which tasks on the object.	Presenter
Rules	Norms, social rules, activity rules and legislation within which the user relates to others in the community.	Can access and edit the presentation slides and memo But cannot show the memo to others
Objective	User's intention and objective of the activity that the user wants to perform.	Show the presentation slides on the public projector and see memo on her private device
Outcome	This is the result of the user's activities, which may or may not achieve the objective.	Gave presentation on the public projector and see memo on her private device quickly

Table 4-1 Table Designers Use in the Context Model

4.4   *Step 4: From Context Elements to Sensors and Profiles*

Instead of using descriptive scenarios to communicate with the developers, the designers have generated the values for each context element in the context model using the definition of each element together with the nine questions about context elements described in the previous section.   The set of values provides a more conceptual and simple form of context for each situation as shown in Figure 4-1. It can then be used as a common reference to understanding the context model for each situation.

Environment		
Values	**Sensor**	*Attributes in the Database*
Cold	Thermometer	*Condition*
Room 2.4	Barcode, Bluetooth or NRFID	*Room*
Building A	GPS	*Building*
University of Bath	GPS	*Area*
Bath	GPS	*Town*
England	GPS	*Country*

Table 4-2 Example of Assigning a Sensor and Attribute Name

As the context model groups the information into nine context elements, each context element has its own database that holds the information about that element.   The implementers who have a better idea of what sensors and profiles are available can then refer to the context model and discuss the availability of sensor data.   Then both designers and implementers can work out how to combine the data to derive as many types (attributes in the database) of information for each

element of context as possible. The combination of the data will also guide the developers in implementing the multiple layers of interpretation. For example, from the information for environment context, developers can see from the table of information that the most specific area that designers require to know is the room number in the building. Therefore the implementers might suggest to the designers that radio beacons, e.g. Bluetooth, should be used to refer to each room and this can be combined with GPS to get the building and town or country, when only the building name or town is required. Furthermore, the thermometer could be used to capture the temperature of the room and infer if the user is inside or outside the building as well.

This step not only supports the designers and implementers in deciding about the sensors and the translation process within the context-aware system, it also supports the developers in designing the storage model for the database (See Table 4-2). For example, the database for an environment sample from one situation incorporates storage of a set of information, which represented as a set of values in the attributes in the database, including room number, condition of the room (cold, wet, hot, dark, etc), building name, street name, town name and country name. Each database has an ID attribute to hold the identification of the set of values in the database.

Environment Database						
ID	**Condition**	**Room**	**Building**	**Area**	**Town**	**Country**
1	Cold	Room 2.4	Building A	University of Bath	Bath	UK
…	……..	…….	……..	……….	…….	……….

Table 4-3 Information from Table 4-2 Assigned to Attributes in Environment Database

This step not only helps designers and implementers decide on the sensing and interpretation, it also allows the developers to assign attributes in the database of each context element as shown in Table 4-3. Thus, each context element has its own database and the values in the attributes can be referred to as a set of information in the profile as well. The profile is generated by the designers where the information cannot be gathered from sensors or they considered it is a better resource of the information required by the system. For example, tourist information which includes information such as attractions, locations, entrance fees, etc. This is done so that the descriptive information is separated from the context model. The descriptive information is grouped and separated from the context model according to its quality and persistence.

Table 4-4 is extended from Henricksen's summary of the typical properties of context information [Henrickson, 2003]. It shows how the context model guides the designers in grouping the context information according to its properties. By grouping the context information, the developers also create the storage structure for context information and separate the information from the context model. By separating context elements which hold different groups of information and the context model, the storage of descriptive information about each context element is separated from the context model storage. The context model only refers to the reference point of each context element in order to refer to the set of values in the context element database, which holds the descriptive information about the context element.

Class of Information	Persistence	Quality issues	Sources of inaccuracy
Sensed	Low	Maybe inaccurate, unknown or stale	Sensing errors; sensor failures or network disconnections; delays introduced by distribution and the interpretation process
Static	Forever	Usually none	Human error
Profiled	Moderate	Prone to staleness	Omission of user to update in response to changes
Derived	Variable	Subject to errors and inaccuracies	Imperfect inputs: use of a crude or oversimplified derivation mechanism

Table 4-4 Typical Properties of Context Information [Henrickson, 2003]

The separation of descriptive information and the context model does not only allow easier editing of the values of the context element, it also allows easier addition or removal of the values in the attributes or the attributes themselves in the context element. When the attribute value in the set of information about the context element needs to be changed, the values in the context models in the history database that refer to the set of information in the context element that hold changed value do not need to be changed but will be affected the changes in the context element.

4.5   *Step 5: From Context Elements to Reasoning*

Not all information in each context element can be inferred from sensors or existing profiles in real time. Context elements such as rule, role, objective and outcome are normally difficult to infer directly from data from sensors and profiles. Hereafter, the information that is difficult to infer from data from sensors

and profiles is called an *undiscovered value*. The developers have to create new databases to store the information for these context elements. The developers use the databases to store values extracted from the scenario situations at design time (e.g. Table 4-1). The relationships between elements of context in the context model are used to guide the developers in implementing the structure of these databases. The databases can then be used consistently to infer the *undiscovered values* of the context elements in the context model in real time.

In order to be able to reason about the user's current objective consistently, the missing elements need to be found. Using the identification concept simplifies the reasoning method because it separates the descriptive information about the context element from the context model that will be stored in the context history. For example, the descriptive information of role (Presenter role is used when the user A interacts with community B on Monday at 9:00 am in Meeting Room 4) is stored as a set of values in the attributes in the role database and the context model that is stored in the database refers only to the reference point (ID) to the information in that role database. When one of the values in the descriptive information about the role changes (for example, user X instead of user A), the value only needs to be edited in the attribute in the role database. The context models in the history database that refer to the role do not need to be changed. Separating the descriptive information from the context model not only allows easier editing of the values in the context element without changing the context models but also allows easier addition or removal of the values in the attributes or the attributes in the context element without affecting the context model in the history. This can be useful when the designers or developers require the modelling of a new scenario of different users who have the same role but have different personal devices. The context model in the history, which could be time consuming to create and store, can be reused. This reduces the work for the designers and developers. In addition, it may provide opportunities for end users to change the values themselves via the context model at run time [see Chapter 5].

The definition of the role context element in the context model is as follows: it is the role of the user to perform the activity in order to meet the current objective according to the community around her in a particular situation. Therefore the role context element can be inferred from who the user is and the community that has an influence on the user's activity at a particular environment and time to meet the objective, as shown in Figure 4-2. This can guide the developers in implementing another reasoning method to get the value of the role of user in the particular situation. For example, Jane R held a presenter role at the time she entered the meeting room with User A, User B and User C who attend the presentation (where User X represents a person named X who are present in the situation). At the same time a week after, Jane R enters the same room with User A, User B and User C gain but this time the User C has access to the projector. Therefore Jane R now held role of Attendant according to the community's characteristic.

According to the relationships in the context model, the value of Jane R's role is stored in the database with reference points (IDs) to the IDs of the user, community, time and environment which hold descriptive information of each context element in the particular situation. By storing them in a separate database, the values in the role database can be changed by the user if required in real time through the interface representing the attribute values (name of role, user ID, Community ID, Time ID and Environment ID). With the consistency of the attributes in the database, the user can easily make corrections or amendments, for example change the community ID to the set of values that they believe is appropriate to their current situation.
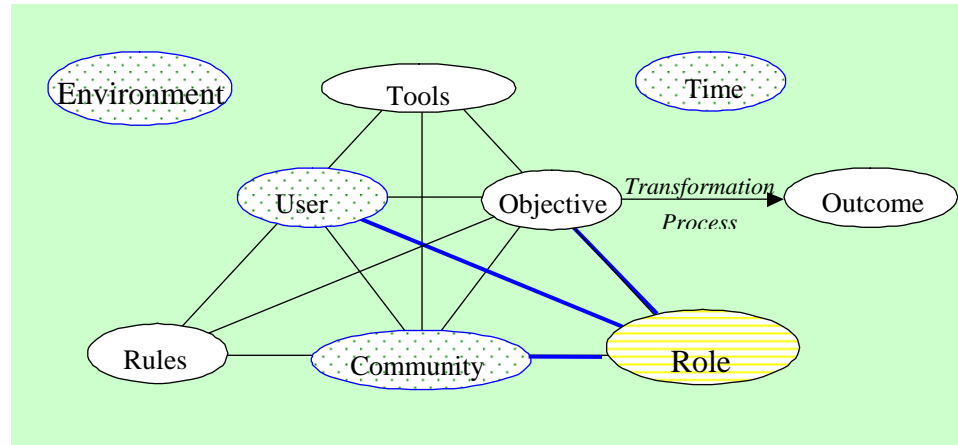
Figure 4-2 Relationships related to the Role Element in the Context Model

Similar to the role context, the rule context has an influence on the user, community, environment and time of a particular situation. When there is not enough information about the rule context from the sensor data or profiles, such as from network privacy rules or authorisation policies, the developers could design the database for the rule element further informed by the role context, as they are influenced by the same elements of context. For example, when the role is presenter, the rules for this role are to be able to access and edit the memo and to avoid showing the memo to the community. At the same time, as a presenter, the user can access *public* tools such as the network and desktop computer. As a result, the rule database should hold the information that is linked between the role and rules context. Therefore when a new role is created in the role context, a set of rules should be created to associate with the new role. For example, the presenter role is associated with the presentation rules of only showing the presentation slides to the public but do not show the memo.
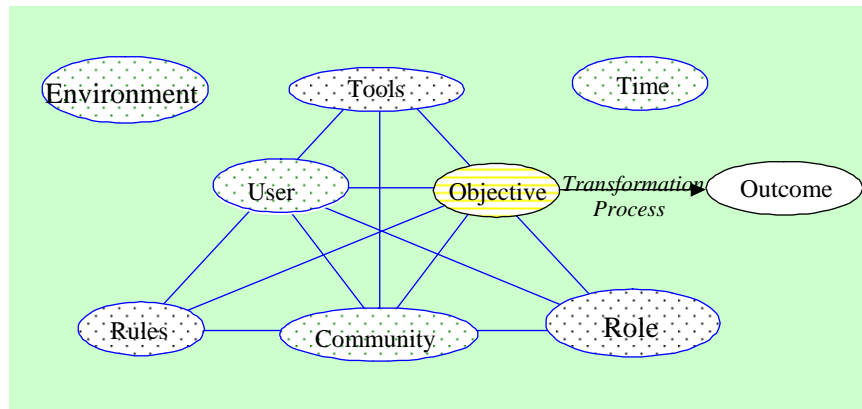
Figure 4-3 Relationships in the Context Model including the Objective Element

The objective context is defined as a user's objective in performing a particular activity in a particular situation – in other words, the user's intention. The information for the objective context is important information that the context-aware system has to find out in order to determine how to support the user. Moreover, the user's intention cannot be sensed, therefore the developers have to design and implement a further interpretation layer in order to be able to infer the information for the objective context in real time. After completing the steps above, the developers should be able to identify the information for all the elements in the context model. The information will then be used to reason about the user's objective against the history of context stored in the database. The context model and its relationships guide the developers to create the interpretation layer that takes information of other elements of context apart from outcome context to compare with the sets of values of the context models in the history in order to infer the value of the current objective context.

Lastly, the context model shows that the outcome context is the result of the objective context so the developers can create a database to be used in order to

infer the outcome context. As a result, the database should hold pairs of information between objective and outcome context. Therefore when a new objective is created in the objective context database, an ideal outcome should be created to associate with the new objective and stored in the outcome database.

4.6 *Step 6: From Outcome Context to Selected Application and Context Information*

From the defined situations and results of following the six questions in Section 4.2, the designers and developers decide whether or not the existing applications can be used to support the user. If none of the existing applications are relevant or useful, the developers use the features that the designer extracted from Section 4.2 as user requirements and guidelines in implementing a new application.

According to the classification of context applications proposed by Chen and Kotz [Chen and Kotz, 2000], the context-aware system should be able to decide what application or service should support a user in a particular situation and the context-aware system should also be able to provide the context information for a particular situation. Therefore for every new outcome for the outcome context, for the active context-aware computing, the designers should assign services and also types of context information that the system should provide to the user in a particular situation. For example, the existing applications are the Microsoft Power Point or opening folder command and the new service could be an extended opening folder command by auto-opening a particular folder instead of starting by searching from the *My Computer* folder.

For passive context-aware computing, the system considers how to represent the context information to the user. For example, the map representation on the user's

PDA screen with information about user's location and other users' location in the community. It will leave user to decide what to do with the information herself.

In order to separate the context reasoning from the application, the design tool presented here guides the designers to create a new database that relates each outcome with the selected service and types of context that should be used to support the user in a particular situation. From Jane R's scenario, the chosen service is "auto detect shared folder on the devices". The context information this service required was information about the environment (room number), tools (public shared folder) and user profile (folder name of the presentation). The service can then use this information to detect a shared folder on the public device in the meeting room and open the presentation folder on the user's laptop computer. Jane then only needs to transfer the presentation file to the shared folder on the desktop and open the memo on her laptop. The context reasoning is no longer embedded in the application as the outcome database holds a reference to information about the current context model and information about the service or application, instead of directly embedding the context reasoning in a bespoke application.

This section described how the context model can be used as an integral part of a design process for context-aware system design. This design process is systematic and easy for designers to follow. The next section discusses where the design tool meets the requirements discussed in Section 2.4.1.

User Database

ID	Name	Surname	Device	...
1	Jane	R	-	…
2	Jay	Kaenam	Laptop	…
	……	……	……	…
n	……	……	……	…

Community Database

ID	UsersID	Initial	Devices	...
1	2, 5, 9	JK, LP, ??	Printer, Laptop, ??	…
2	2, 7	JK, CW	Mobile, Laptop	…
	……	……	……	…
n	……	……	……	…

Role Database

ID	Name	user ID	Community ID	.
1	Receptionist	1	1	…
2	Presenter	1	2	…
	……	…	……	…
n	……	…	……	…

Tools Database

ID	Tool IDs	...
4	Tuesday	...
...	...	...

Context Models Database

ID	User	Environment	Tools	Role	Community	Rules	Objective	Outcome	Time
1	1	1	4	2	2	2	3	5	4
...	...	...	...	...	...	...	...	...	...

Projector

Meeting room, Building West

Tuesday 5 Jan

Environment Database

ID	Room	Building	...
1	Meeting	Building West	...
…	……	……..	…

Jane R

Print Note — *Transformation Process* → Printed Note

Time Database

ID	Day	...
4	Tuesday	...
...	...	...

Rules Database

ID	Name	Role ID
2	Do not show note	2
...	...	...

Do not show

Attendants

Present

Objective Database

ID	Name	...
3	Print note	...
...	...	...

Outcome Database

ID	Name	Objective ID
1	Presented Slides on Projector	3
...	...	...
5	Printed note	3
n	……	……

Application Database

ID	Application	Outcome ID	Context Info
3	PTT presentation	5	Tools
...	...		...

Figure 4-4 Schema with Context model and Databases

136

4.7   *How the Design Tool Meets the Design Tool Requirements*

The previous section discussed the use of the design tool with examples. This section evaluates the design tool against the design tool requirements mentioned in Section 2.4.1.

Consistent Support for Shared Understandings

An abstract level of consistent elements of context and their relationships is provided in the context model as a basis for shared understandings about context for researchers i.e. designers, implementers and users. With its consistent vocabulary, designers can refer to the context model when they discuss context elements with the developers and users. The developers can refer to the context model when they want to refer sensor data and their interpretations to the designers.

Identification of Context Elements

The context model identifies the key elements that have an influence on the user's behaviour. At the same time the model is not too complex as there are nine consistent key abstract elements of context (environment, time, user, community, tool, role, rule, objective, and outcome) that designers need to refer to when they try to extract relevant context from the user's requirement or scenarios. With this structured but simple context model, the designers expand their design outlook away from the availability of current technology. The designers can concentrate on what types of context have an influence on a user's behaviour in the situation rather than what technology is available to them.

Context Interpretation

The boundaries between elements in the context model provide the designers with a consistent tool to transfer a descriptive knowledge of a user's requirement to a consistent structured context model of the situation. The designers use the practical model to communicate with the implementers. Based on the nine key elements in the context model, the boundaries of the elements also help the designers and developers to group and form the interpretation of data from different sensors and profiles into an abstract level of information for each context element. By having the nine key elements of context as a uniform guide for the designers, the designers have to group information for each context element. This means the designers have to transform the data from sensors and profiles before the values can be assigned to the attributes in the context element database. So rather than embedding the sensor data acquisition in the context elements or context model directly, the implementers develop the interpretation of sensor data separately from the sensor data acquisition.

Separation between Context and its Reasoning

The relationships between context elements in the context model guide the designers on how to reason or infer the context elements to determine a user's current objective. From the consistent context elements and the uniform relationships in the context model, the designers can derive a consistent inference process in the context system independent of the applications. From knowing the user's current objective, the developers can assign what and how the system should support the user. The applications do not have to concern themselves about context reasoning.

History and Time

The context model provides a temporal dimension where each point on the timeline captures the context of the situation at that time. By storing a context

model at different points in time, the system automatically stores a set of context models in the past – i.e. it maintains a context history. With the timeline, the history of context can be represented for the context-aware system. The history can then be used during reasoning about user's current objective.

From the evaluation above, the proposed context model introduces a systematic design tool that meets the requirements described in Chapter 2. It provides a systematic design tool for designers and implementers to develop a context-aware system, with its uniform context elements and relationships between them helping to steer the designers away from a technology driven approach.

4.8   *From Context model to New Design Tool*

This chapter described the use of the context model as a design tool. A new systematic step by step design process for context-aware system design was introduced. Following the three levels of activity, six activity level questions are introduced to assist designers in transforming a descriptive scenario into a structured set of requirements. The structured requirements help designers make decisions about when and how the system should support the user. The context model provides a consistent set of vocabulary for designers to build understanding about context. Using the context model, the design process helps designers to design a system to meet user requirements rather than design a system driven by technology. Moreover, the design tool introduces a consistent approach to context and its reasoning that may be used to help build understandings of context by researchers, designers, developers and users. At the same time, it also takes into account valuable information about context such as time and history. Furthermore, the design process shows the possibility that the design can be developed consistently. This can help reduce the time taken to design and develop new systems as it becomes more straightforward to reuse or expand existing systems built based on the shared understanding of context.

However, in order to take full advantage of our design tool and process, a new architecture is required to support fully the functionalities of the context-aware system design introduced by the new design process. The next chapter will discuss this new architecture.

Chapter 5

A System
Architecture for
Context Modelling

The design tool presented in the previous chapter introduces a consistent approach to identifying and representing context elements, their relationships and history. Moreover, it also supports separation between the context model and its reasoning. In order to benefit from these advantages that the design tool introduces, a new architecture is required to support the functionalities when moving from design to implementation of a context aware system.

This chapter introduces our system architecture to support the results of using the design tool previously introduced. First, an overview of the architecture presents a data flow through the architecture. Based on our context model and design tool, the data flow shows how context is inferred about the current user's objective from

current context and its history. An overview of the architecture is then discussed further to explain the features of the three layered structure of the architecture and its advantages. This chapter provides a conceptual account of the architecture to support developers during implementation. The implementation of the architecture is discussed in detail in Chapter 7. At the end of this chapter, the requirements described in Section 2.4.2 are discussed to demonstrate how the architecture meets these requirements.

5.1 *An Overview of Context Aware System Architecture*

The separation between sensor and context model introduced by the use of our design tool leads to an architecture which contains three separate layers including Sensor Engine layer, Context Engine layer and Application Engine layer. Each layer deals with different types and levels of data (see Figure 5-1), separating the handling of sensor data, the interpretation of sensor data and profiles and context reasoning. Each layer consists of different objects. These objects can be initiated on a single device or multiple devices. The context elements in the context model provide a structure for the objects in the Sensor Engine layer to transform the data from sensors into a consistent sets of information according to the context elements. The relationships between context elements in the context model provide a uniform structure through which the objects in Context Engine layer infer information about user's current objective. Along with the three layers, the architecture includes databases which hold information about the context elements in the context model and the history of the context models. In Figure 5-1, the databases and translated data are represented as XML but different languages can be used to represent the database in the system.

142
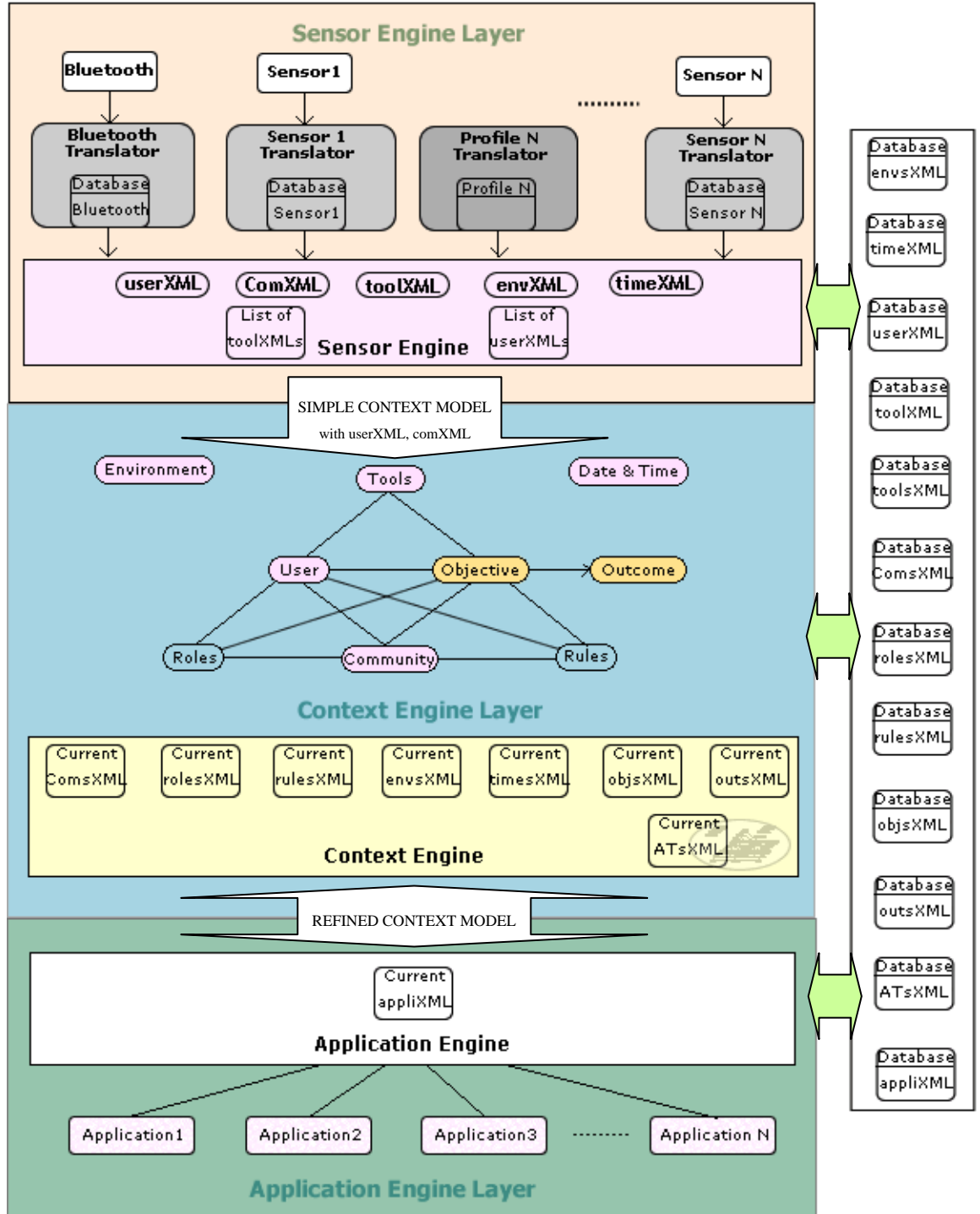
Figure 5-1 Overview of Architecture of the Context Aware System

5.1.1 The Flow of Data

As a result of using our context model presented in Figure 3-8 and our design tool, Figure 5-2 shows the flow of the data in the architecture and the possibility of coping with misunderstandings between the system and the user.

First layer, the sensor engine layer, contains objects that deal with different types of sensors (♠) and profiles (♣). This layer transforms raw sensor data into meaningful information for the attribute values in each context element. The meaningful information is then combined and translated into information of context elements (♥) in the context model that can be gathered from sensors and profiles based on the history of each context element to represent current state of the user. The meaningful information is raw sensor data processed into more accurate data and/or information that has a meaning to the user. For example, raw GPS data may be transformed into a building's name.

The second layer, the context engine layer, uses information of context elements (S_0) from the sensor engine layer to translate and infer other context elements in the context model. The context elements (S_0) are used with the history of the context model (P_n) to infer the user's objective for a particular situation. As mentioned, during design stage, the context models are extracted from the user's requirement scenarios. These models are stored in the history of context model to be used to as a reference during run time. The current context model from the sensor engine layer is compared to the history of context model. The model in the history that has the best match to the current context model is then used to infer the missing elements in the context model such as roles and objective to get a complete current context model ($S_0 + S_1$).

Third layer, the application engine layer receives the current context model with the inferred user's objective and outcome. It then provides support to the user according to the inferred outcome in the current context model. The application engine layer uses the outcome context element in the current context model to access application database to provide support to the user accordingly. If the

user's activity (♦) is not what the system predicted, the application engine can then update the value of the outcome of the situation in the application database. Therefore the value of the outcome needs to be updated in the inferred context model i.e.

1. The preferred application is assigned with the outcome in the application database.

If the new outcome is added to the application database:

2. The new outcome is added to the outcome database and assigned with the current objective.

3. The context model in the database is updated with the new outcome.

This layer provides the possibility of allowing user to be able to make changes to the context model together with the automation of the system. Further studies need to be done in order to understand the involvement of the user without irritating the user.

As a result of using the design tool, the architecture has the flow of data shown in Figure 5-2. Table 5-1 illustrates how the context-aware system implementation based on the design tool provides the separation of context according to its properties. The design tool guides the designers to assign sensors to different groups of context. The context model guides the designers to group different types of context information and separates them according to the context elements. As a result, the developers implement an architecture that supports separation between different types and levels of data. Moreover, the architecture supports the processing of different levels of data separately in different layers.
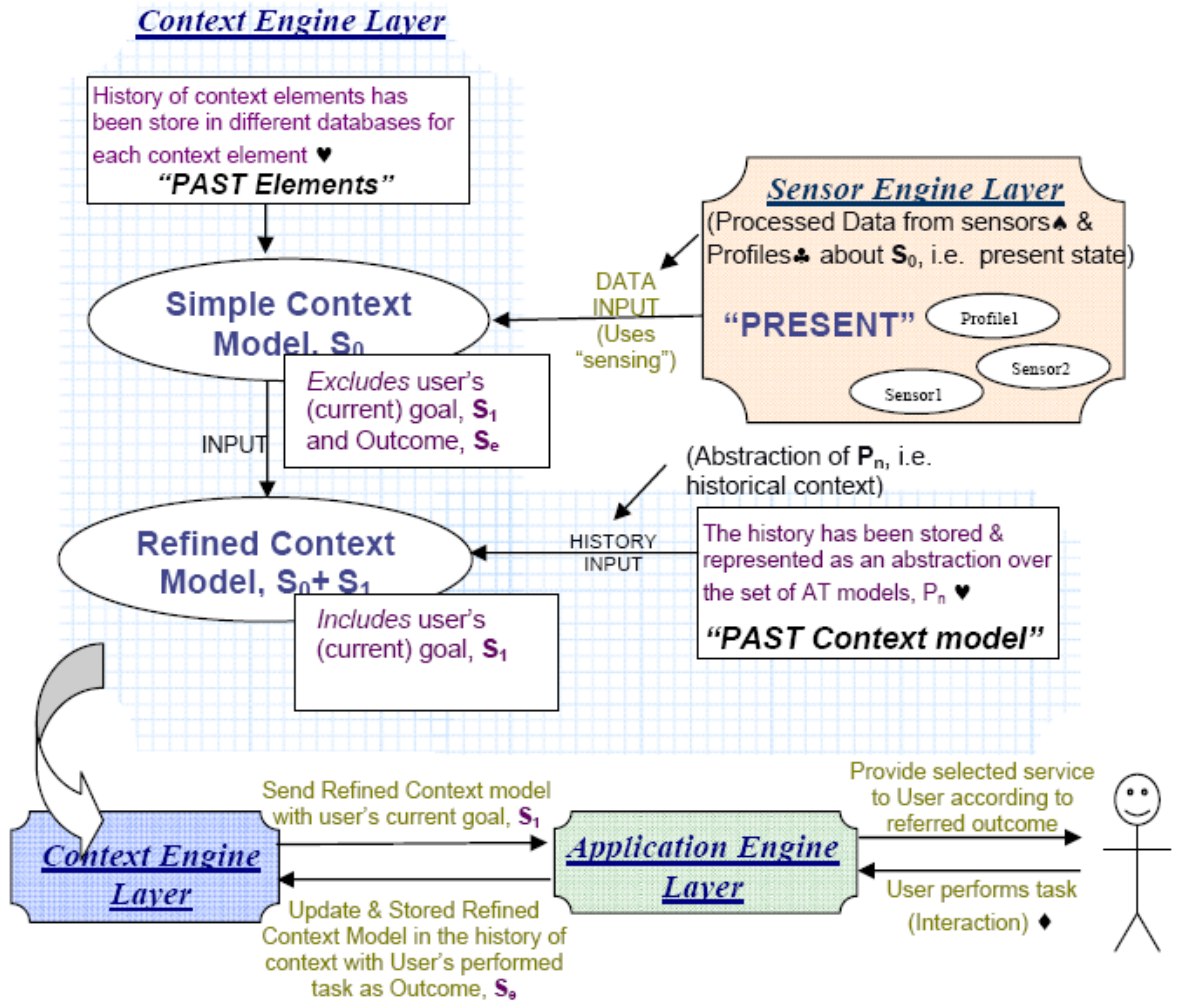
Figure 5-2 Flow of Data in the Architecture

Class of Information	Persistence	Quality issues	Sources of inaccuracy	Via the context model
Sensed	Low	Maybe inaccurate, unknown or stale	Sensing errors; sensor failures or network disconnections; delays introduced by distribution and the interpretation process	Raw data or interpretation of raw data from assigned sensor♠
Static	Forever	Usually none	Human error	End user interactions♦
Profiled	Moderate	Prone to staleness	Omission of user to update in response to changes	Profiles ♣
Derived	Variable	Subject to errors and inaccuracies	Imperfect inputs: use of a crude or oversimplified derivation mechanism	Interpretation layers♥

Table 5-1 Typical Properties of Context Info [Henrickson, 2003] Separated Via the Context Model

First, the most dynamic context, which is context from sensing (♠ in Figure 5-2 and Table 5-1) is stored in the sensor translator separately from the profiles and other databases such as context elements and context model. Secondly, user feedback or interaction from the user to the application (♦ in Figure 5-2 and Table 5-1) which has a static property is dealt with in the application engine. For example, the system might have provided the user with a presentation service as a result of the values stored in the application database but in real time the user wants to use a tourist map service and the user wants this to apply in the future. The user can then change the value in the application database for the current situation to provide the tourist service instead of the presentation service. Thirdly, the profiles of the user or tool e.g. user's research interest, favourite food (♣ in Figure 5-2 and Table 5-1) which can be updated by the user and which can be

dynamic to a moderate level are stored separately and dealt with in the sensor engine. Lastly, derived data (♥ in Figure 5-2 and Table 5-1) such as data in the context elements (e.g. building name, town, room number derived from GPS, Bluetooth, etc) and context model (such as the value of user's role, user's current objective, etc) has its own separate database. As a result of the separation of different types and levels of data, if the developers want to take the properties of context into account during the objective inference process, this can be done without remodelling the context. For example, if the developers want to take the frequency of use of the derived data (♥ in Figure 5-2 and Table 5-1) into account, it can be done without affecting other types of data by adding another attribute representing the frequency of use to the database of the derived data.

5.1.2   Databases

As mentioned in the design stage, the context elements and context model have their own databases containing information about them. Each element contains a set of attributes that hold information about the particular context element. The values of the attributes can be gathered from other databases. In order to be able to refer to other databases, a unique reference point (ID) concept was introduced. The unique reference point is assigned to each set of attribute values in each database. The design tool prepares the developers to create separate database for each sensor, context element, and context model. It also helps developers prepare the profiles database where the information cannot be gathered directly from the available sensors. The clear separation between sensor, profile, context element and context model introduced by our design tool provides an opportunity for the developers to produce separate databases for each sensor, profile, context element and context model.

As each context element has its own database, its database can be stored anywhere in the system so long as the developers provide the system with code that allows

other objects in the system to communicate with the database (represented as ⟺ in Figure 5-1). The developers use the assigned attributes in the databases designed as in Chapter 4 in developing the code for managing a set of values in the database. Hereafter, the code for managing a set of values in a particular database is called *Database* Object (i.e. *Environment* Object holds the code that manages the set of values in the environment database). Apart from managing the set of values in the database, the code also has two main functions:

1. Storing a new set of values: a function that allows a new set of attribute values to be added to the relevant database. To store a new set of values efficiently, the function has an ability to check if the set of values does not already exist in the database before storing it, to avoid repetition in the storage space.

2. Accessing (reading/editing) the existing sets of values in the database: a function that allows the other objects in the architecture to be able to access the values in the database. The function uses the given ID to find the set of values in the relevant database. Once the set of values is found, the function has abilities to read and edit any attribute values of the identified set of values in the database. The function then updates and saves the set of values in the database where there are some changes to the set of values.

These functionalities of each database allow the objects in the system to store and access the values in the database in real time.

As these databases are separated and of uniform structure, the developers can implement a GUI based on the attributes in each database to provide easier access for the user to view, edit and add to the values in the database during design, training stage and real time use. This not only allows the user to be able to update

the values in the sensor, profiles and context elements databases but also allows the user to update the reasoning process by adapting the values in the context model database and the application database.

5.1.3 Sensor Engine Layer

The sensor engine layer consists of three main elements including a sensor acquirer for each sensor, a sensor translator for each sensor and a sensor engine. The main objective of this layer is to deal with each sensor and its raw data so that the data is translated into meaningful data for each element of context. The source of data is not just from sensors but also from profiles (such as map profile, building profile, etc) where necessary. This data from sensors and profiles is dynamic [Henricksen *et al.*, 2002]. The data may need to be processed constantly.

Sensor

From the results of using the design tool, the sensors are assigned to acquire different types of raw data. The developers finalise the type of available sensors that will be used in the context-aware system. The Bluetooth object and sensor1 object etc in Figure 5-1 represent the sensor acquirers for different sensors. Each sensor object in Figure 5-1 contains code that communicates with the sensor for acquiring its raw data. Then the sensor object sends the data to its translator object by notifying a resource discovery. The resource discovery in the Sensor Engine layer has functions to detect the sensors or profiles that are available to the system in the current situation and triggers them to start sending the data to their translators. The code for the sensor object can be stored in the sensor itself or in the same device as the architecture, as long as it has methods of acquiring the raw data and sending the data to the resource discovery and therefore to its translator.

Sensor Translator

A sensor translator contains code that processes the raw sensor data and translates it into information for the attribute values in different context elements. The attributes are assigned during the design stage as a result of using our design tool. The first step that the developers might consider in processing the raw data is to reduce the noise in the raw data. This is because the sensor may not be as accurate as it should be. The raw data from the sensor can be inadequate to use directly in inferring about the situation. Then the second step for each new raw data is to translate it into values that are suitable to be stored as attribute values in each context element, as illustrated in Table 5-3. Based on the attributes that the sensor was assigned to sense the values for (during the design stage), the developers implement the translation code for the raw data in order to get those values. The set of values are then stored in the database for each sensor data.

The sensor translator object has a database for each sensor. When raw data is sent to the translator, it has an ability to detect that the raw data has already been stored in the sensor database. Therefore the translation process can be reduced as the database can refer to the old translated values and transfer that information to the context elements.

For information that cannot be sensed by sensors, the developers implement profiles. The profile database has attributes that hold a set of information about each item in the profile. For example, a tourist map profile database has attributes that hold information about different tourist maps with reference points to tourist attractions in the town, points of interest, events etc. A common profile example is the user's preference profile that might hold information such as professional interests, list of allergies, food preferences, tourist interests, etc. The values in the attributes in the profile database can be assigned to the attribute values in the context element. Similar to the sensor translator, the profile translator requires

having a method to allow the profile to communicate with other objects. This method will allow other objects to access the values in the profiles.

Bluetooth 1	
ID	000e0797f047
Owner Name	Clematis
Device Name	Clematis 6680
Device Type	Nokia 6680
Location	Mobile within 50m

Bluetooth 2	
ID	00119fc048e5
Owner Name	Kat
Device Name	KatDesktop
Device Type	Desktop
Location	Room 2.2 Building J

Table 5-2 Raw Data from Bluetooth Translated to Meaningful Information

Bluetooth Database				
ID	**Owner name**	**Device name**	**Device type**	**Location**
000e0797f047	Clematis	Clematis 6680	Nokia 6680	within 50m
00119fc048e5	Kat	KatDesktop	Desktop	Room 2.2
…..	…..	……	……	…..

Table 5-3 Info from Bluetooth used as Info in Context Elements Database

As a result, for each sensor, the developers first need to implement the code for processing the raw data into a meaningful set of values for each attribute in the database as shown in Table 5-2. The raw data from Bluetooth such as MAC address (Device ID) and Device name are translated further. For example, the

MAC address is used to get information about Device type (and also information such as Owner name and Location when available) from the profile of that device which is stored in the database and referenced via the Bluetooth MAC address.

Secondly, for each set of data, a method is needed for assigning the values to the attributes in the sensor database. The method also has a functionality to assign a unique reference point to each set of values from the raw data, shown as ID in Table 5-3.

Thirdly, the developers need a method for detecting the sensor data that already exists in the database. By detecting existing data before the translation process begins, the translated values in the database are used instead of retranslating the raw data where possible in order to reduce processing time.

The new or old translated meaningful values are then transformed into information for the attributes in the sensor database. For example, from Table 5-2 and Table 5-3, each Bluetooth data is translated into information about other detected people – or their devices – around the user that the system is serving, e.g. "Clematis" who owns Nokia 6680 mobile phone device appears to be situated within 50 meters of the user.

Lastly, the sensor translator object requires a method to send the set of values from the sensor data to the sensor engine object. The sensor engine object uses the translated attribute values in the sensor databases to assign to the attribute values in each context element.

Sensor Engine

The sensor engine is used to combine different sets of values from different sensors and profiles in order to get appropriate information for each context element as illustrated in Figure 5-1. For example, the GPS provides information about the location of the environment context element and the thermometer provides information about temperatures of the current environment. This information is combined so that the information of the environment context element can be now represented as "outside building A in cold weather". The sensor engine does not only combine the values from different sensors to get better information for the context element. It also combines the values from different context elements to get the information for another context element.

For example, different sensor translators (for Bluetooth, beacons, RFID, etc) can translate data from sensors to represent different people in the environment. The translator transforms sensor data into meaningful information about detected users. For example, the data from an RFID sensor can be translated to a user's name and preferred device. The information is then assigned to the attributes of the user context element for each detected person from different sensor. By combining the detected user context element for each person detected by different sensors, the sensor engine object can infer the information about the community context element for the particular situation. Similarly, from the combination of different tools in the environment, the sensor engine gets the information about the tools context element. Moreover, where the information cannot be gathered from sensors, the information from an attribute value in one context element can be used to refer to the available profiles in order to get further information about different attribute values in the context elements. For example, the RFID sensor detects data which is translated to the user's name. From the user's name, the food preference for the user can be found in the user's profile.

154

By following the design tool, the developers use the context element in the context model as a guideline in creating the code for combining the sets of values from different sensors for each context element where the sensor data is available. The code assigns values to the attributes in each context element, normally the user context element, community context element, tools context element, environment context element and time context element respectively. These context elements were gathered from the available information from sensors and profiles. We call the combination of these context elements the Simple Context Model (see Figure 5-1 and Figure 5-2). Figure 5-1 demonstrates the transformation from Bluetooth data to the information for the user context element and the result of other context elements.

In order for the Context Engine layer to access information about current available context elements from the Sensor Engine layer, the developers require a method in the Sensor Engine layer that has the ability to send the information of current available context elements to the Context Engine layer. The method sends the current available context elements to the Context Engine layer as different sets of information for different context elements. The Database Object code of each available context element is used to allow the Context Engine layer to access different sets of information for different context elements. For example, the Environment Object, in which the Sensor Engine layer gathers available information about current environment that has influence on the user's objective, is passed to the Context Engine layer.

Bluetooth Database				
ID	**Owner name**	**Device name**	**Device type**	**Location**
000e0797f047	Clematis	Clematis 6680	Nokia 6680	within 50m
00119fc048e5	Kat	KatDesktop	Desktop	Room 2.2
…..	…..	……	……	…..

User	
ID	1
Name	Clematis
Surname	Wallis
Device	Nokia 6680
Food Preference	Vegetarian

Time	
ID	4
Time	12.00
Day	Monday
Date	12
Month	November
Year	2006

Community	
ID	1
Users IDs	1,2,5
Users initials	CW, JK, unknown
Devices	mobile, laptop, unknown

Tools	
ID	2
Tools IDs	1,3
Names list	Printer room2.5, unknown

Figure 5-3 Bluetooth data Transformed into Info for User Context Elements

5.1.4   Context Engine Layer

The Context Engine layer consists of the context engine object. The aim of the context engine object is to reason about the context elements in order to infer the user's current objective or activity. It therefore transforms the Simple Context model to the Refined Context Model as shown in Figure 5-1 and Figure 5-2. The design tool supports a consistent process of transforming the available context elements, which have been gathered from sensors and profiles, to infer other elements in the context model. The developers implement the code based on the relationships between context elements in the context model in order to infer about user's current objective.

First, the sets of available context elements (Simple Context Model) from the Sensor Engine layer need to be checked to determine if they are already exist in the context element databases. The Database Objects of the available context elements are compared with the values in their databases. If it does not exist in the database, a new unique reference point (ID) is assigned to the set of information for the purpose of simple referencing and storing (see Figure 5-1). The information is then stored in its context element database. If it is found in the database, the existing set in the database can be updated if necessary. A method that provides an ability to detect the existence is therefore required in this layer. The method should also have an ability to update and store the information in the database for the context elements. After this first step, the IDs of the available context elements are recognised and ready to be used in the next step.

User 1	
ID	1
Name	Clematis
Surname	Wallis
Device	Nokia 6680
Food Preference	Vegetarian

Community 1	
ID	1
Users Ids	1,2,5
Users initials	CW, JK, ??
Devices	mobile, laptop, ??

Tool 1	
ID	1
Name	Room 2.5 Printer
Owner	University
Type	Printer
Location	Room 2.5 BJ

Figure 5-4 Sensor Data Transformed into Info for Context Elements

Secondly, the context engine object is required to reason between available context elements in order to get the attribute values for missing context elements starting from the role context element as described in Section 4.5. The developers require a best match algorithm to infer the information about the role context element from the identities (IDs) of the information from different context elements – including the user context element, community context element, environment

context element, time context element and tool context element respectively against the history of role element in the role database – rolesXML in Figure 5-1. By using the identities (see Figure 5-5) instead of the information of each element itself, it reduces the complexity of the reasoning process. It hides the interpretation within the context elements and therefore the reasoning about the role context can easily be done consistently using IDs of available context elements. The developers should implement code for their chosen matching algorithm that takes IDs of available context elements including the user context element, community context element, environment context element, time context element and tool context element and compares them with the corresponding attribute values in the past context models in the context model database.

Thirdly, the additional information about the rules context element can be found from the roles context element. Using the design tool during the design stage, the designers developed a database of rules for the different roles of the user from the scenarios. During the design stage, the designers analysed the scenarios and assigned different set of rules for each role in different scenarios. As mentioned in Section 3.6.1, these rules are not just limiting to the legal law that user must not break but it is also including norms that guide user to behave as good citizen but it is acceptable to break these rules. As the rule database is separate from the other databases and only refers to the ID of roles, it can easily be updated. The profile of rules is stored in the rule database – rulesXML in Figure 5-1. In the rule database, a set of rules is stored with a reference to the role value's identity as shown in Figure 5-5. Therefore once the role information is found, the developers require code that uses the role identity to refer to the information of the rule context element from the rule context element database.

Rules Database

ID	Name	roleID	...
1	Able to see hospital patient database	1	…
n	...	**....**	...

User Database

ID	Name	Surname	Device	...
1	Clematis	Wallis	Printer	…
2	Jay	Kaenam	Laptop	…
	……	……	……	…
n	……	……	……	…

Role Database

ID	Name	userID	CommunityID	...
1	Receptionist	1	1	
2	Presenter	1	5	…
	……	…	……	…
n	……	…	……	…

Community Database

ID	UsersID	Initial	Devices	...
1	1, 2, 5	CW, JK, ??	Printer, Laptop ,??	…
2	1, 2	CW,JK	Mobile, Laptop	…
	……	……	……	…
n	……	……	……	…
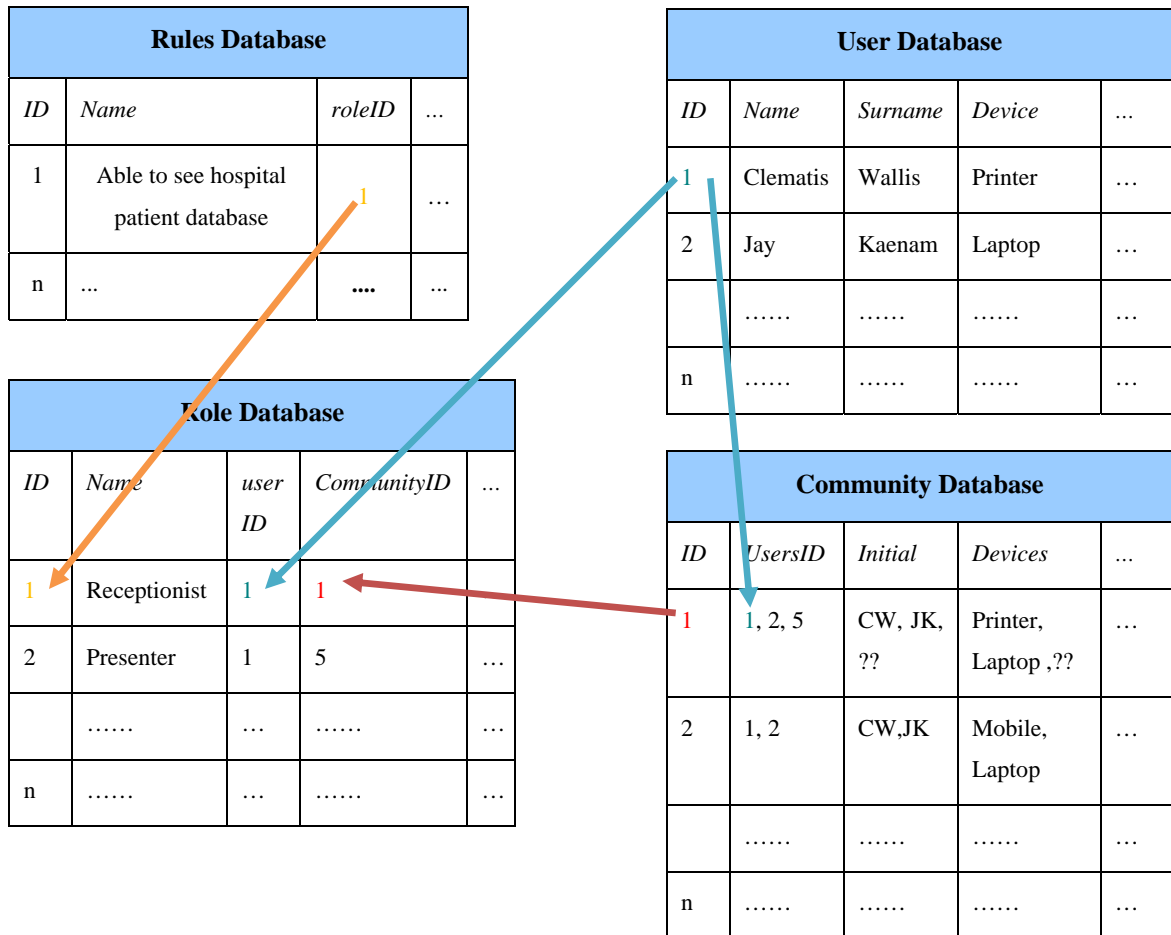
Figure 5-5 Database Examples (show how a value in one can be used in other context elements)

Fourth, the implementers require code that performs a best match algorithm in order to reason about context from IDs of available context elements in order to infer information about a user's current objective or activity. As mentioned in Section 5.1.1 that the current context model from the sensor engine layer is used to compare to the reference context models in the history of context model, the matching algorithm is used to compare the combination of IDs of the context elements in the current context model against ones in context model in the history of the context models in its database – ATsXML in Figure 5-1. Following the

relationships between context elements in the context model, the role is extracted from the model that has the most similarity of combination of IDs of context elements in the current context model provided by the sensor engine layer. From the role ID found, the rules ID can be found in the rule database. This will give us the current context model with the combination of IDs of the user context element, community context element, tool context element, environment context element, time context element, role context element and rule context. The current context model is used to find best matched context model in the history of context model. The best matched model is found from comparing the ID of each element of current context model with one in each context model in the history of context models. It compares similarity of the IDs in the current model and ones in the models in the history in order to find the best possible model in the history that has the most similarity IDs to the ones in the current context model. Once the best matched context model is found, the current objective can be referred to from the ID of the objective context element in the found context model. The history of context models can be created by the designers during the design stage or by the user while using the system. The designers can assign values in the context model to store in the history by analysing the scenarios. As a result of using the design tool, the objective database is created in order to store information about the objective with its unique reference point that can be referred to by the outcome database.

Fifth, the implementers follow the context models of different situations that the designers have extracted from the scenarios to create a database for the outcome context element. Each set of values of the outcome context element is paired with the reference point to the objective context element which is used to refer to the set of values of the objective context element. This outcome context element database provides a simple form of the history of the outcome context element that is stored in the database.

The sixth method that the developers are required to implement is the method for storing the information of each element in its context element database. Not only is the information of context element stored in the database, the information of the context model at a particular situation is stored in the Activity Theory based context model as well. This is an important part for inferring the information of missing context elements and the user's current objective. For each context model, only the identities of the context elements are stored in the context model database to minimise the storage requirement and reduce the complexity in the inference process. The interpretations within the context element are hidden from the reasoning between the elements to provide simpler consistent reasoning between the context elements. The interpretations within the context element are done separately and, after it is done, ID is used to represent the set of values of the context element. The ID of the context element is used to infer the user's objective according to the relationships between context elements in the context model against the history of context model. Therefore when the interpretation within the context element changes, it does not affect the inferring process of user's objective through the context model. Moreover, the context model takes the effect of the changes through the use of ID of the context element without changing the context model itself.

Lastly, a method for sending the information of the context model for a particular situation to the Application Engine layer is required so that the application can refer to current context information from the identity of each element in its database if it requires it.

5.1.5 Application Engine Layer

The Application Engine layer is used to assign suitable support for the user in order to complete the predicted objective and meet the predicted outcome. However, the actual outcome, which is the achievement from the user's actual

actions, may not be the same as the predicted outcome. The architecture should provide the user with a misunderstanding recovery system when the context-aware system provides the wrong service to the user as a result of predicting the user's objective wrongly or inferring about context elements wrongly.

The Application Engine layer first implements a resource discovery to find out what applications or supports are available for the user in a particular situation when it receives the context model of the current situation from the Context Engine layer.

According to the classification of context-aware applications provided by Chen and Kotz [Chen and Kotz, 2000], applications are divided into 2 categories, passive and active as shown in Table 2-5. Active context-aware applications perform tasks for users automatically while passive context-aware applications automatically represent new context to the user. For the Application Engine to decide at what level to support the user, during the design stage the designers extract different values of the outcome context element from the scenarios to assign to different applications or services that the user might need. For each assigned application, the information from the context elements that is required by the application or service to support the user during completing the activity is also assigned in the application database. In order to provide suitable support to the user, the Application Engine layer should be able to access the application profile that contains:

1. Identity (ID) for each set of values of the information about the application: the ID is used by other objects in the architecture to refer to the set of values in the application database.

2. Outcome identity: in order to link between the current context model and the prospective support for the user, the outcome ID is used in the application

database to refer back to the outcome database in order to be able to get more information about the current values of the outcome and the context model. For example, for each set of values in the application database, it holds ID of the outcome that has value of "1". The outcome ID of 1 is used to refer to the set of values in the outcome database that holds information about the particular outcome, including information such as outcome name ("Jane presents the presentation on time").

3. Names of the application or service: it is used to give a shared understanding about what application or service will be provided to the user by the system. For example, opening Microsoft Power Point or opening folder command are used as Names of the application in Jane's scenario.

4. Names of the context elements that are required to be used with the application or to represent context information to the user: the names of the context elements are used by the Application Engine to access information about the context elements in the current context model. From the names of the context elements, the application engine accesses the current context model and gathers the ID values of the required context elements in the current context model. The ID values allow the application engine to access further information via the values in the attributes of the particular context element in the current context model (such as the room number in the environment context element, public shared folder and devices in tools context element and user's profile holding folder name of the presentation in user context element).

By using our 6 step design tool described in Chapter 4, the designers provide the developers with an abstract set of data from different situations in the scenarios. (See Chapter 6 for examples.) The developers build the application profile and store it in the application database. In order to find suitable applications or supports for the current situation, the Application Engine object implements a

method to take the identity of the outcome element from the received context model to find the best match in the application database. Once the best match is found, the set of values in the application database is chosen to support the user. The Application Engine can then use the information in the set of values (such as name of application, name of context elements, etc) to provide services to the user.

The information from the database is used by the Application Engine to initiate the application or service for the user with the information from selected context elements. Therefore the developers implement a method to first get information of the required context elements from their database. Once the matched set of information is found in the application database based on the best match of the current outcome ID, the value of Names of context elements attribute in the application database is extracted from the best match set of values. The value in that attribute contains a list of context element IDs. The IDs are then used to refer to the information from the context element databases. The information is transformed into the information that the application requires. For example, the ID of the user is taken by the method to find the data with the matched ID in the database to get details of the user. The information about the user leads to the user's profile, which holds information about the location of the folder of the presentation file. A method for initiating the chosen application or service is then required in the application engine object. For example, with the information of the location of the folder in which the user stores the presentation file; the application engine object gets the Names of the application, such as the opening folder command. Then it triggers the command to open the presentation file's folder. From the information about the tools, the application engine object gets information about the available shared devices. The method takes the chosen device's information into account in order to provide support to the user in the most appropriate manner. For example, from the tools information, the application engine object finds that the desktop computer is connected to the projector in the shared tools. The application engine object then assigns the device to show the

presentation. Contextual mediation [Chalmers *et al.*, 2004] can be used to improve the usability of data here by selecting a suitable format and device to represent to the user.

The next section discusses how well the new architecture meets the requirements for a context-aware system architecture described in Section 2.4.2.

5.2   *How the Architecture Meets Each Architecture Requirement*

The new architecture is introduced in this chapter in order to support the functionalities that our context model and design tool introduce to the context-aware system. The functionalities of the new architecture are compared here to the context-aware system architecture requirements described in Section 2.4.2.

Separation of Concerns

This chapter introduces the architecture that consists of three layers as a result of using the design tool during the design stage. The three layer architecture aims to provide separation between sensor, context reasoning and application.

As the top layer (Sensor Engine Layer in Figure 5-1) deals with different sensors, each sensor has its own code that enquires and translates sensor data. New sensors can provide their own descriptions and template interpretations as long as it has the translator to register the information to different attributes in context elements. The Sensor Engine Layer deals with sensor data separately from the context elements, context model and application. Together with the unique reference point (ID) concept, each sensor is therefore independent from the context elements, context model and application.

For example, a new sensor is introduced to the system. The sensor gathers further information about the environment. The environment database is changed and updated with the new information from the new sensor by easily adding new attributes to the existing sets of values in the database. The existing Activity Theory context model database holds the ID of the set of values from the environment database in each predefined context model in the context model database. Each predefined context model in the database will take the new values in the environment database into account without changing anything in the context model database itself. This is because the sensor has its own database which can be referred to by using the ID instead of referring directly to the attribute values in the database.

In the Application Engine layer, when a new sensor is added to the system, the application does not need to be changed in order to use sensor data from this new sensor as the Sensor Engine layer and the Context Engine layer will process the sensor data to a form that applications can access. On the other hand, when there is a new application, the code for the sensor and context model reasoning does not need to be changed or rewritten.

The architecture is aimed at supporting the developers in implementing and acquiring sensor data code for each sensor and the application code separately. This avoids the burden of rewriting the code and provides an easier way of adding new sensors and applications to the system. As a result, the sensors and applications can also be programmed in different languages or run in different platforms. Since the sensors only require the sending of raw data to the interpretation layers, they do not need to know how to translate the raw data. At the same time the applications only need to know what they are supposed to do to support the user.

The architecture hides the interpretation and context reasoning in the Context Engine layer from the sensors and applications so that only the relevant abstract level of information is passed to the sensors and applications in the Sensor Engine layer and Application Engine layer respectively. The architecture supports the separation of concerns through this ability.

Context Interpretation

In order to be able to support the separation between sensors, context reasoning and applications, the architecture is built upon the context model that supports the separation between sensor data, the information of context elements and context reasoning between elements. First, it provides the Sensor Engine layer to support the codes for acquiring raw sensor data. Secondly, it supports the interpretation from raw sensor data to more meaningful data by using the sensor translator. With the consistency of the context elements in the context model introduced by Activity Theory, the developers can implement the Sensor Engine object to translate and combine the more meaningful data and profiles to get information for each context element in the context model consistently. Thirdly, in the Context Engine layer, the architecture provides the developers with a uniform reasoning process between context elements in order to infer the user's current objective or activity so that that the system can support the user. Lastly, the Application Engine layer deals with interpretation about the support for the user separately from dealing with the sensors and context model. However, with the use of the unique referent point (ID) concept, the Application Engine object can access the information from the sensors and context model consistently. Therefore when there is a new sensor, the developers only have to concentrate on the code for acquiring the data and how to translate the raw data into information for each element of context. They do not have to worry about how to reason about the context elements in order to infer the user's current objective or activity. When there is a new application, the developers do not have to worry about rewriting the code for reasoning about the user's current objective or activity. They only have

to assign the new value of each context element for the situation that the application would be used in to support the user. The architecture supports the interpretation of the sensors, context model and application separately in a consistent manner. The interpretation can then reused by multiple applications.

Constant Availability of Context Acquisition

The architecture separates the components (such as sensor object, sensor translator and context engine) that acquire sensor data, interpretation for each context element and context reasoning. The components execute independently from the applications that use them. For each sensor, the code for sensor object is used to initiate the sensor in order to acquire sensor data. For example, the GPS object initiates the GPS receiver in order to get latitude and longitude values. The sensor translator then processes the sensor data to be used by the context engine. For example, the GPS translator translates the latitude and longitude values into name of the country, town and building and then assigns them to the values in the context elements. The context engine reasons about the context elements and stores the context model in the database for the application engine to access at its own time. The architecture allows the components to execute independently from the applications that use them. The components work independently from each other. As a result, the components are available to multiple applications continuously.

Context Storage and History

As there is a clear separation between sensor data, context elements and context reasoning, each level of context can be stored easily and consistently. First, the architecture provides storage for each sensor in order to store meaningful data after the raw data from a sensor has been transformed with reference to each new sensor data by the sensor translators. Each sensor has its own sensor database that holds information that can be used as meaningful information for context elements.

Secondly, the information from sensors and profiles are translated and combined into the information of each element of context and this is also stored in each context element database in the architecture. Thirdly, as a result of using the design tool, after the context elements have been reasoned and inferred from to get the user's objective in the context model, the architecture provides storage for the context model (context model database, ATsxml) so it can store the history of the context model for each situation. Lastly, the architecture supports the storage of information of 'what and how' the application should support the users to complete their objective or activity (i.e. the outcome context element in the context model) in each situation. The architecture provides a separation between different levels of data by having the sensor databases, context element databases and context model database). By having consistent and separate storage for each layer in the architecture and the use of the unique reference point, the developers can then easily edit or update the database without affecting other layers or levels of information.

Resource Discovery

As the architecture hides the sensors and context reasoning process from the applications, the Sensor Engine layer has the resource discovery mechanism in the sensor engine to notify the system of the available sensors and profiles that are available to the system. The application only needs to know what and how to support the user, the architecture separates the sensors from the applications and uses the resource discovery mechanism to provide information about available sensors and profiles in the data sets in the databases instead of hardcoding the sensors into the application. The resource discovery will notify the system when there are changes in context. The Sensor Engine layer and Context Engine layer hide the detail of where and how to acquire sensor data from the application. The new applications or sensors can then easily be added to the architecture.

170

Security and Privacy

A proper treatment of security and privacy is beyond the scope of this dissertation and currently the architecture does not deal with security and privacy issues. However, to a very basic degree a simple level of security is provided by the role and rule context elements. The role and rule elements show the potential of the security and privacy mechanism. The rule element can hold rules of ownership or use policies. The rules or policy support security and privacy issues by controlling access to the data or devices. For example, if the meeting folder is accessible to certain people who attend the meeting, the rule in the context model will refer to the role of the user whether she is part of the meeting or not. If the user is part of the meeting, the rule will set so that the user can access the folder.

This chapter presents an architecture that aims to support the functionalities introduced by the design tool presented in Chapter 4. Section 5.2 discussed the functionalities of the architecture and how the requirements for the context-aware system architecture presented in Section 2.4.2 are met and not met. The next section summarises the transformation of the functionalities introduced by the context model to the structure of the architecture.

5.3 *From Context Model to New Architecture*

Building on the design tool presented in Chapter 4, the new architecture consists of three layers: a Sensor Engine layer, a Context Engine layer and an Application Engine layer. The first layer (Sensor Engine layer) deals with different sensors and profiles in order to transform raw data into more meaningful and less noisy data that is ready to be referred to as part of the context elements in the context model introduced by Activity Theory. The second layer (Context Engine layer) uses the information of current context elements and the history of the context in the database to infer about a user's current objective and possible outcome. The

Application Engine layer uses the value of the outcome context element in the current context model from the previous layer to provide support to the user and update the current context model if necessary. Table 5-4 shows an overview of the responsibilities of each components in the architecture.

The architecture provides a separation between applications and sensors so that it gives flexibility to changes in the sensors without affecting the applications. Moreover, it supports the separation between context elements, their relationships and their history. As a result, it provides a separation of their databases (i.e. sensors databases, context element databases and context model databases). With the unique reference point (ID) concept and the separation of the databases, this allows the context to be reused, expanded and updated easily as the process can be done in different part of the data without changing everything in the system. This is significant because the process of modelling and gathering context is expensive. Moreover, the architecture also provides the potential for developers to provide mechanisms for security and privacy.

The previous chapters described the features of the new context model, design tool and architecture. In the next chapters we will test their use in two extended scenarios.

Layer	Component	Responsibilities	Information	
			Receive	Send
Sensor Engine Layer	*Sensor*	Enquires the data from sensor		Sensor Data
	Translator	Translates raw data from sensor into a meaningful information for attributes in context elements	sensor Data	Meaningful information
	Sensor Engine	Assigns information to get information about context elements	Meaningful information	Partial context elements in current context model
Context Engine Layer	*Context Engine*	Infers user's current objective	partial context elements in current context model	Complete current context model
Application Engine Layer	*Application Engine*	Provides service or information for the user	complete current context model	

Table 5-5 Overview of Responsibilities of each components in Architecture

Chapter 6

Evaluation of the
Context Model and
Design Tool

In ubiquitous computing , scenarios and field studies are often used to motivate user requirements [e.g. Abowd *et al.*, 1996; Dey, et al., 2001; Agarawala *et al.*, 2004; Brown, 1996; Hinze and Viosard, 2003; Hopper *et al.*, 1997; Kim *et al.*, 2004; Schilit and Theimer, 1994]. This chapter presents the application of the proposed context model and new context-aware system design tool described in Chapter 3 and Chapter 4 to two scenarios in order to demonstrate their feasibility. The first scenario is adapted from a common scenario that has been used previously with a simple location based system [Haghighat *et al.*, 2004; Helal *et al.*, 2005; Hinze and Viosard, 2003; Hsu *et al.*, 2007; Kim *et al.*, 2004]. This scenario describes how a simple tour guide and conference assistant uses context to provide new services to the user. The second scenario is based on ethnographic studies in the Accident and Emergency (A&E) department of a London hospital [O'Neill, et al., 2004] and is more complex. The healthcare staff in this setting

work under pressure. Timing is crucial because it could affect the lives of the patients. Moreover, the staff have to deal with multiple tasks within short periods of time and with interruptions. As a result, patients can feel that they have been interrupted during services or been ignored. The hospital field studies that are examined in this section demonstrate the use of the context model in more complex situations. As a result of applying the design tool to these scenarios, the implementation and evaluation of the architecture can be demonstrated in Chapter 7. At the end of this chapter, the use of the context model and design tool is assessed against the context model and design tool requirements presented in Section 2.4.1.

6.1   *Scenario 1: A Simple Tour Guide and Conference Assistant*

"Adam is attending a technical conference in Hamburg, Germany. The conference features a large number of presentations and demos spread over multiple tracks. Adam is attending the conference with his colleagues Bob and John and they have decided to try to attend different presentations. When Adam picks up his conference package on the first day, he provides his contact information and the topics he's most interested in. He also mentions that his colleagues Bob and John are attending. Along with the conference proceedings, he receives a personal conference assistant, software for his handheld device designed to guide and assist him throughout the conference. Adam has a hard time deciding what to attend for the first session. The sessions start in five minutes. He turns to the conference assistant. Based on his known interests, as represented in his profile, it recommends a presentation and a demo that have similar keywords. Adam chooses the presentation and the system then gives him directions to get to the presentation room.

At lunch time Adam wants to catch up with Bob and John before the next session starts. He only has ten minutes to look for them so he uses the conference assistant to find Bob and John's locations in the building. The assistant knows that

Bob and John are Adam's colleagues, so it has automatically shown Bob and John's locations on the map relevant to Adam's location and not everyone else's locations.

After the conference ends, Adam has one day to look around Hamburg. He does not have much time before catching his flight back to UK but he wants to see a little bit of Hamburg. He again turns to the conference assistant. Based on his current location and check-in time, it shows a map with his current location and five attractions closest to him with a short description of each place when he clicks on them. Adam chooses the attraction closest to him by clicking on it. The assistant then displays the route to the attraction and estimated time of walking there. Adam follows the route on the display."

6.1.1   Step 1:   Defining Scenarios in which the System will be Applied

The scenario above described the possibility of how the assistant would support the user (Adam in this case) in different situations in the conference and tourism domains. These two domains are frequently used in developing context-aware applications. The descriptive scenario is used to provide a better understanding between users and designers. With the descriptive scenario, the designers can engage real users in evaluating the scenarios before the system is even implemented as it is easy for the users to imagine themselves in the descriptive scenario. The scenario is used by the designers to identify situations for each activity where the system will support the user, as shown in the next step.

6.1.2 Step 2: Define Situations Where Context Awareness Can Support the User

The designers extract situations for the user from the scenario so that they can be modelled into different models following the elements in the context model. This will provide a simpler form for easier referencing with the developers about the situation. This will also help designers analyse the situation and design the functions of the application to support each activity of the user. For each situation, the designers are guided to answer the six activity level questions about the situation. The situations here are extracted from Adam's scenario and the answers to the six questions for each situation are as follows:

6.1.2.1 Situation 1

Adam has a hard time deciding what to attend for the first session. The sessions start in five minutes. He turns to the conference assistant. Based on his interests, it recommends a presentation and a demo. Adam chooses the presentation.

From this situation, the designers follow the six questions in order to analyse the situation and gather the user's requirements. The designers make decisions about the support that the context-aware system provides to the user. The analysed data from the situation are then stored in the system databases so the system can use them in real time to detect the situation, where the system should provide support for the user, from the current context.

Question 1: What is the activity for the context-aware system to support in this situation?

As the aim of this situation is to be able to select the talk that Adam wants to attend quickly, the activity in this situation is "selecting the presentation to attend". This leads to the answer for the question in step 2.

Question 2:   What are some actions that a user may need to perform?

In order to select which presentation to attend, there are two main goals that Adam is trying to achieve. The first goal is to find the presentations that are on in the next 5 minutes. The second goal is to find the presentations that match his interests. Therefore the actions are first narrowing down the presentations to ones that are on in the next five minutes and then narrowing things down further to the ones that are relevant to his interest. From these actions, the operations for the next questions can be answered.

Question 3:   What are some operations that the user may need to perform?

To narrow down the presentations to the ones that are about to be on in the next 5 minutes, the operations are first to open the conference timetable and then select the current time to show the list of presentations that are about to be on. In order to meet the goal of finding a suitable talk, the operation is to click through the presentations list from the previous operation and find the one that is relevant to Adam's interest.

The designers identify the level of support that will be suitable for activity, actions and operations from the previous questions. In this case, the system automatically looks through the timetable database with the search conditions of a starting time within 5 minutes and keywords of the presentation that are matched to the user's interests. Then it automatically shows the narrowed down list of relevant presentations to Adam. As a result, the actions and operations of narrowing down the list are assigned as *active supports* from the system. The system then provides *passive support* by presenting the narrowed down conference timetable. The system however lets Adam pick the presentation that he wants to attend himself as it can be too specific for the system to decide when there is more than one

presentation that starts at the same time and matches the same keywords. As a result, the system lets the user have control.

Question 4: What operation level support is the system going to provide?

- Open the conference timetable *>> Passive*

- Select the current time to show the presentations list *>> Active*

- click through the presentations list *>> Active*

Question 5: What action level support is the system going to provide?

- Narrow down presentations to ones that are on in the next 5 minutes *>> Active*

- Narrow down further to the ones that are relevant to Adam's interest *>> Active*

Question 6: What activity level support is the system going to provide?

- Selecting the presentation to attend
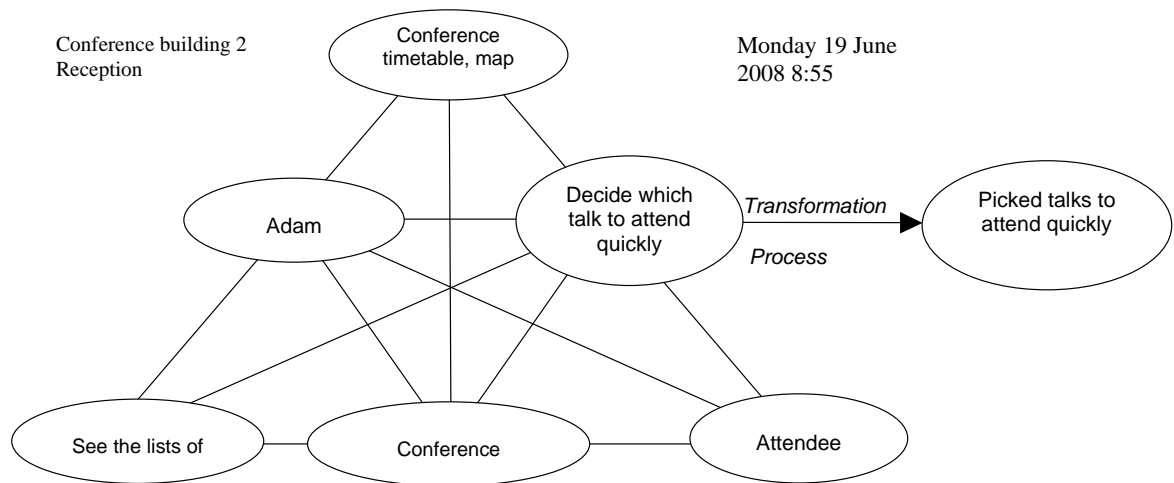
Figure 6-1 Context Model of Situation 1- Deciding which talk to attend

From the object and outcome elements in the extracted context model of Situation 1 (see Figure 6-1) and the scenario description, the designers require applications that will reduce the information about the conference timetable and emphasise the relevant information to Adam. By reducing information, it will reduce the time for Adam to scroll through the small PDA screen to see the talks available at that time as the screen is too small to show the whole schedule. By emphasising the information, Adam will be able to spot which talk is most relevant to his interest quicker. Thus it will reduce Adam's decision making time.

A recommendation is therefore added to the timetable in order to provide a highlighted timetable to show which talks are the most relevant to Adam's interest in his profile. As the application will be used in Adam's PDA which has a small screen and cannot show the whole timetable at once. The highlighted timetable uses the current time to minimise the information of the schedule by showing only the talks that start now and after the current time. As it is likely that the users do not need to include past talks in the decision making process.

The designers repeat the same steps in order to complete the situations in the scenario.

6.1.2.2   Situation 2

When the talk is selected, the assistant shows the directions to the room according to Adam's current location.

Question 1:   What is the activity that the context-aware system is to support in this situation?

- Get to the presentation room on time

Question 2:   What are some actions that a user may need to perform?

- Find the quickest route to the destination

Question 3:   What are some operations that a user may need to perform?

- Open the map

- Look in the building map for his current location

- Look for the presentation room on the map

Question 4:   What operation level support is the system going to provide?

- Open the map *>> Passive*

- Look in the building map for his current location *>> Passive*

- Look for the presentation room on the map *>> Passive*

Question 5:   What action level support is the system going to provide?

-     Find the quickest route to the destination *>> Active*

Question 6:   What activity level support is the system going to provide?

- Get to the presentation room on time *>> Both*
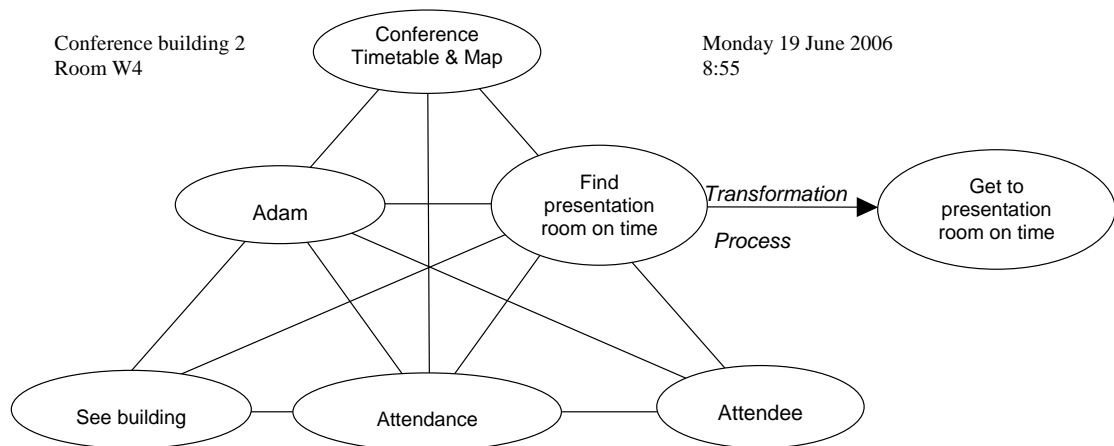
Figure 6-2 Context Model of Situation 2 - Get to the presentation room on time

Similarly, the designers refer to the answers of the six levels of activity questions. The system uses context including information about the location of the user and information from the timetable about the room of the presentation in order to find the quickest route for the user. The system provides *active supports* by automatically finding Adam's location and the presentation room on the map instead of using the user's explicit input of the location of himself and the presentation room. The system can get the information itself and find the quickest route sufficiently. The system then provides *both active and passive support*s by automatically finding the quickest route to the destination from the present location and showing Adam's route information on the PDA screen.

The route finder application is therefore added to the timetable in order to provide support for getting the user to the presentation room. When the user selects the presentation that he wants to attend, the application shows his location and destination on the map with the quickest route to reach the destination. It takes information about the user (current location) and room number from the conference schedule to show the directions to the location which allows the user to get to the destination quickly.

6.1.2.3   Situation 3

He only has 10 minutes to look for them so he uses the conference assistant to find the location of Bob and John in the building.

Question 1:   What is the activity that the context-aware system is to support in this situation?

        - Find Bob and John in the building

Question 2:   What are some actions that a user may need to perform?

        - Find the quickest route to Bob and John

Question 3:   What are some operations that a user may need to perform?

        - Open the map

        - Look in the building map for his current location

        - Look for Bob and John on the map

Question 4:   What operation level supports is the system going to provide?

- Open the map >> *Active*

- Look in the building map for his current location >> *Active*

- Look for the Bob and John on the map >> *Active*

Question 5:   What action level supports is the system going to provide?

- Find the quickest route to Bob and John >> *Active*

Question 6:   What activity level support is the system going to provide?

- Get directions to Bob and John >> *Both*
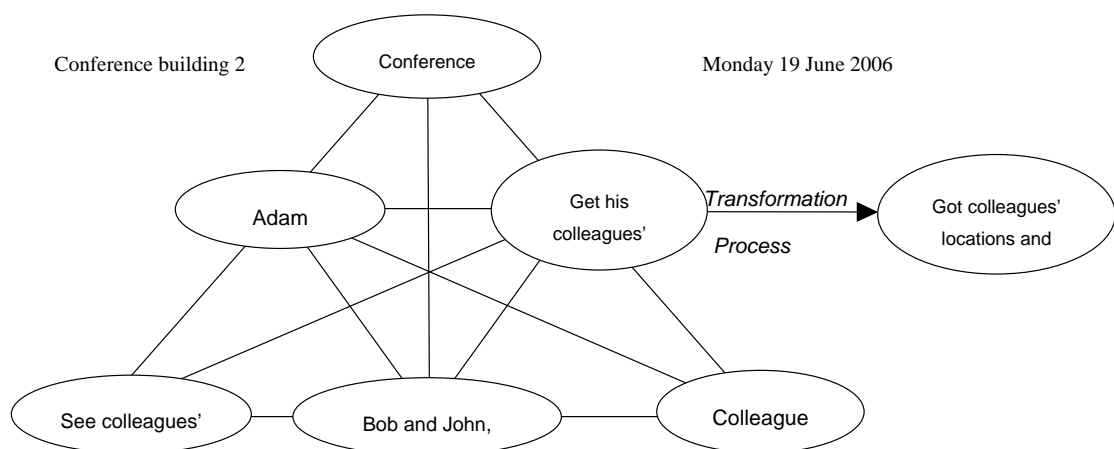
Figure 6-3 Context Model of Situation 3 - Adam meets up with colleagues

The system uses context including information about the location of Adam and his colleagues' information from the system in order to find the quickest route for Adam. The system provides *active supports* by automatically finding his location and his colleagues' locations on the map instead of using the user's explicit input of the location of himself. The system can get the information itself and find the

quickest route sufficiently. The system then provides *both active and passive support*s by automatically finding the quickest route to Adam's colleagues and presenting the route.

The object and outcome elements in Figure 6-3 suggest that the assistant should be able to show the location of conference attenders on the map if their existence can be detected in the same room. It also narrows down the people on show as there are many people at the conference who are not relevant to the user and whose details do not need to be displayed. The assistant should therefore take the user's information (user's profile of colleagues or friends list, current location) and community information (colleagues' locations) into account in order to show only information relevant to the user (locations of colleagues Bob and John).

6.1.2.4 Situation 4

He does not have much time before catching his flight back to UK but he wants to see a little bit of Hamburg.

Question 1:   What is the activity that the context-aware system is to support in this situation?

         - Visit nearby tourist places

Question 2:   What are some actions that a user may need to perform?

         - Narrow down to nearby tourist attractions in the area

         - Narrow down to relevant tourist attractions in the area

         - Find directions to the attractions

Question 3:   What are some operations that a user may need to perform?

    - Open the map

    - Get his current location on the Hamburg map

    - Look for the tourist attractions on the map

Question 4:   What operation level supports is the system going to provide?

    - Open the map *>> Passive*

    - Get his current location on the Hamburg map *>> Active*

    - Look for the tourist attractions *>> Active*

Question 5:   What action level supports is the system going to provide?

    - Narrow down to nearby tourist attractions in the area *>> Active*

    - Narrow down to relevant tourist attractions in the area *>> Active*

    - Find directions to the attractions *>> Active*

Question 6:   What activity level support is the system going to provide?

    - Visit tourist places *>> Both*

Similarly to Situation 1, the assistant provides *passive support* by showing a map of the area around Adam. Instead of showing the location of people around him inside the conference building, it shows an outdoor map of the local tourist attractions. It provides *active supports* by automatically narrowing down the features according to the user's interest and current location. Therefore the assistant automatically takes the user information (user's profile of tourist interests

and current location) into account in order to show only information relevant to the user (locations of interests that have a location close by to the user). The community in this case is just people around him. They do not have more substantive relationships. The user then selects the attraction that he wants to visit and the system automatically shows the directions on the map according to his current location and location of the selected tourist attraction.
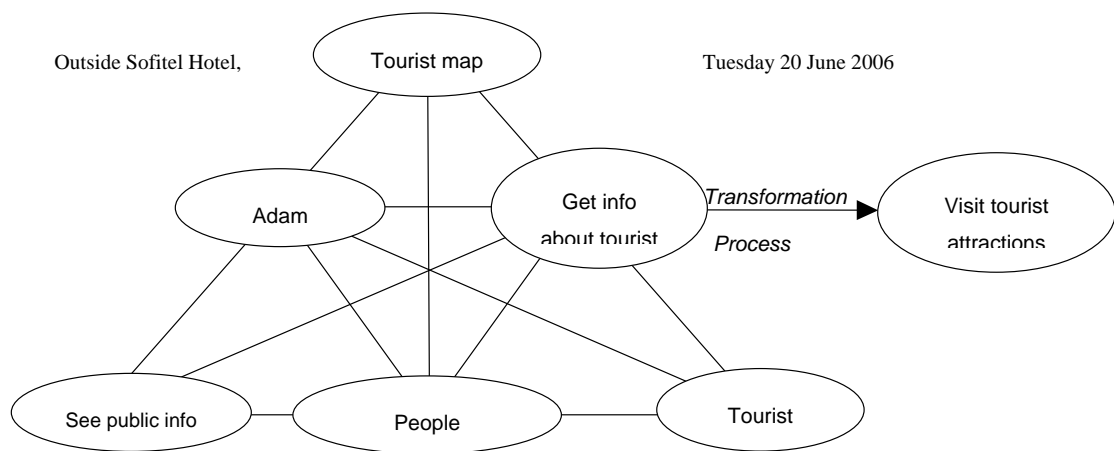
Figure 6-4 Context Model of Situation 4- Get directions to the selected attraction

For every situation, the designers now concentrate on modelling the context that influence a user's activity as shown in Step 3 below.

6.1.3 Step 3: From Situation to Elements in Context Model

Following on from the definitions of elements in the context model in Chapter 3, the values of each element are identified in more detail for each situation. This will support the designers to describe the possible values that are required to be detected for each situation to the developers uniformly and descriptively. The developers can also refer back to the simple models of the situations in Step 2 if

188

they want to study the relationship between the elements for a better understanding of the model for each situation.

The designers use the nine questions about the context element below to fill in Table 6-1 to Table 6-4:

1. User element: *Who is the user that the context-aware system supposes to support?*

For a particular user, other elements in the model are identified according to the user;

2. Environment element: *Where is the activity achieving both physical and virtual?*

3. Time element: *When is the activity achieving?*

4. Tools element: *What are the tools supporting the user to complete the activity?*

5. Community element: *Who are the people influencing the process of completing the activity?*

6. Role element: *What is the role of the user in society?*

7. Rules element: *What are the rules that restrict user in the current society?*

8. Object element: *What is the objective of the user to complete the activity?*

9. Outcome element: *What is the result from the activity that a user is performing?*

In this scenario, Adam is the user that the system is supposed to support. The user element includes information about Adam, for example his preferences, his interests and the current personal devices that he is carrying at that time in each situation. As a result of answering the questions above, the tables of context elements in each situation are extracted.

6.1.3.1   Situation 1

Context Elements	Values from Situation
ENVIRONMENT	The reception of the conference in front of the Building 2, University of Hamburg, Hamburg, Germany
TIME	Monday 19 June 2006 8.55
USER	Adam who has a PDA and iphone Research interests: Context-aware computing, mobile an ubiquitous applications, smart environment
TOOLS	Desktop 2 connected to the projector at the reception Wireless network Conference Map Conference timetable
COMMUNITY	Conference attendants including Bob and John
ROLES	Conference attendee
RULES	See the list of talks on the conference schedule Only access public devices and folders on the wireless network
OBJECTIVE	Decide which talk to attend quickly
OUTCOME	Attend the selected talk quickly

Table 6-1 Values are Identified for Context Elements in Situation 1

6.1.3.2   Situation 2

Context Elements	Values from Situation
Environment	Room W4 in the Conference Building 2, University of Hamburg, Hamburg, Germany
Time	Monday 19 June 2006 8:55
User	Adam who has a PDA and iphone Research interests: Context-aware computing, mobile an ubiquitous applications, smart environment
Tools	Wireless network Conference Timetable Conference Map
Community	Attendants
Roles	Attendee
Rules	See talk information See map
Objective	Find presentation room on time
Outcome	Got to presentation room on time

Table 6-2 Values are Identified for Context Elements in Situation 2

6.1.3.3 Situation 3

Context Elements	Values from Situation
Environment	Cafeteria in the Conference Building 2, University of Hamburg, Hamburg, Germany
Time	Monday 19 June 2006 13:50
User	Adam who has a PDA and iphone Research interests: Context-aware computing, mobile an ubiquitous applications, smart environment Colleague lists: Bob and John
Tools	Wireless network Conference timetable Conference Map
Community	Conference attendants and Bob and John
Roles	Colleague
Rules	See colleagues' location Not allowed to see attendants that do not have a relationship with or they do not register as a public user
Objective	Get his Colleagues' locations
Outcome	Got colleagues' locations on the map with reference to his current location

Table 6-3 Values are Identified for Context Elements in Situation 3

6.1.3.4   Situation 4

Context Elements	Values from Situation
Environment	Outside Sofitel Hotel, Alten Wall 40, 20457, Hamburg, Germany
Time	Tuesday 20 June 2006 9:00
User	Adam who has a PDA and iphone Tourist interests: Parks & Scenic attractions, Churches, Castle Return flight booking
Tools	Wireless network Tourist Map
Community	- or People
Roles	Tourist
Rules	See public information about the tourism
Objective	Get information about the tourist places
Outcome	Got information about tourist places near his current location and direction to get there

Table 6-4 Values are Identified for Context Elements in Situation 4

The tables provide a summary of information that designers consider as important for elements of context in each situation. The tables provide designers with a systematic group of information about the context for each situation. The next step describes the use of the table as a communication tool for designers and developers about the context in each situation.

6.1.4　　Step 4:　　From Context Elements to Sensors and Profiles

The designers use the detailed description of the values for each context element in Step 3 to discuss with the developers the possibilities for sensing the data or translating the data. The description of the values for each context element in different situations are grouped together in order to design the format for the sensing method and modelling the database for each context element. This will help the designers and developers agree on the selected sensors or creating values in the profiles better. For each value of the context element, the designers and developers decide a sensor or profile to use to capture the value as shown in Table 6-5. The values also generate the names of the attributes to store the values in the database for the context element. For example, the designers and developers agree that the value "cold" can be captured from the thermometer and the attribute name in the environment database for this value should be called or "*Condition*" (short for "*Weather Condition*").

Some values such as "*room*" can be captured from more than one type of sensor. Therefore the developers can assign more than one of the sensors for the value in case one sensor fails or is not in range.

Environment		
Values	**Sensor**	*Attribute in Database*
Cold	Thermometer	*Condition*
Reception of the conference in front of building	Bluetooth	*Room*
Room W4	Bluetooth	*Room*
Cafeteria	Bluetooth	*Room*
Outside Sofitel Hotel, Alten Wall 40	GPS	*Building*
Hamburg Airport,	GPS	*Building*
Conference Building 2	GPS	*Building*
University of Hamburg	GPS	*Area*
Hamburg	GPS	*Town*
Germany	GPS	*Country*

Table 6-5 Values of the Environment Element from Different Situations

After the names of attributes in the database are assigned, the attributes in the database can be created accordingly. The values from each situation are created as a set of information in the database (see for example the environment database in Table 6-6). Note that not all the values must be in the database as some values might not be necessary in the particular situation.

Environment Database						
Environment ID	**Room**	**Building**	**Area**	**Town**	**Country**	**Condition**
1	Reception	Building 2	University of Hamburg	Hamburg	Germany	-
2	W 4	Building 2	University of Hamburg	Hamburg	Germany	-
3	Cafeteria	Building 2	University of Hamburg	Hamburg	Germany	-
4	Outside	Sofitel Hotel	Alten Wall	Hamburg	Germany	Cold

Table 6-6 Environment Database Stores Sets of Values of Info in Different

Situations

Time		
Values	**Sensor**	*Attribute in Database*
Monday	System clock	*Day of the week*
Tuesday	System clock	*Day of the week*
19	System clock	*Date of the month*
20	System clock	*Date of the month*
June	System clock	*Month*
2006	System clock	*Year*
Morning	Interpretation	*Period of Day*
8	System clock	*Hour of the day*
9	System clock	*Hour of the day*
55	System clock	*Minute of the hour*
00	System clock	*Minutes of the hour*

Table 6-7 Values of the Time Element from Situations Modelled to the

Database

Time Database							
Time ID	**DOW**	**DOM**	**Month**	**Year**	**Period of Day**	**HH**	**MM**
1	Monday	19	June	2006	Morning	8	55
2	Monday	19	June	2006	Morning	9	00
3	Monday	19	June	2006	Lunch	13	50
4	Tuesday	20	June	2006	Morning	9	00
5	Tuesday	20	June	2006	Lunch	12	30
6	Monday	19	June	2006	-	-	-
…..	….	…	…	…	….	….	….

Table 6-8 Time Database Stores Sets of Values of Info According to the Attributes

For values that cannot be assigned a sensor, the developers have to design the interpretation methods. For example, for the value of a *period of day* attribute such as morning, the developers implement the mathematical calculation to group the *hour of day* into a different *period of the day*. In this case, a simple algorithm is created to group hour of day into 4 groups of morning (5-11am), lunch (12-13pm), afternoon (14-18pm), evening (19-22pm) and night (23, 0-4am). If the *hour of day* is between 5 and 11 then it classes the *period of day* value as morning. The time between 11pm and 4am classes the *period of day* value as night.

For other values that could not be assigned a sensor, to gather the data or translated data from the sensor data the developers have to create a profile if the

values are necessary, as shown in Table 6-9. At this stage, the developers decide what profiles to create. The profiles will be finalised after all the values in the element are assigned.

User		
Values	**Sensor**	*Attribute in Database*
Adam	Profile or Log in info	*Name*
PDA	Bluetooth or user profile	*Personal Device*
iphone	Bluetooth or user profile	*Personal Device*
Context-aware computing, mobile and ubiquitous applications, smart environment	User profile	*Research interest*
Parks & Scenic attractions, Churches, Castle	User profile	*Tourist interest*
Flight booking	User profile	*Schedule*
Perfume for wife	User profile	*Duty free shopping list*

Table 6-9 Values of the User Element are Modelled to the database

Table 6-9 shows that the user profile is required to hold a user's personal information or preferences such as research interests, tourist interests and duty free shopping list. The user profile is created for each user and should be easily accessible by the user so the values can be changed upon the user's needs. At this stage, if the information of each preference is too detailed, the developers can create another profile that holds a further description about the preference and the

user profile can refer to its reference point (see Table 6-10). By storing the descriptive values in the user profile and letting the user context element's attributes refer to them instead of building in the element itself, the user can easily edit and add the values and new attributes in the profile without changing all the values in the attributes in the user element in the database. For example, if the date of a flight booking is changed from the previous trip, the value in the Schedule attribute in the user element database does not change as it still refers to the same reference point in the user profile even though the value of the flight booking has changed.

Schedule Profile			
ID	**Name**	**date**	**time**
1	Flight Hamburg to London	18 June 2006	18.45
….	…	….	…

Table 6-10 Example of a Trip Booking Profile

User Profile					
ID	**User ID**	**Research interest**	**Tourist interest**	**Schedule**	**Duty free shopping list**
1	1	Context-aware computing, mobile and ubiquitous applications, smart environment	Parks & Scenic attractions, Churches, Castle	1	Perfume for wife
….	…	….	…	….	….

Table 6-11 Example of a User Profile

User Database			
User ID	**Name**	**Personal Devices**	**User Profile**
1	Adam	PDA, iphone	1
…..	….	…	….

Table 6-12 User Database Stores Sets of Values of Info

The tools context element is composed with information about different tools or devices. Each tool or device has its own descriptive information. To separate the descriptive information of each tool and the information of the tools available in each situation, the database for each tool is created with the reference identity that the tools context element can refer to.

Tools		
Values	**Sensor**	*Attribute in Database*
Public desktop 2 connected to projector	Bluetooth	*Tools list*
Printer 2	Wifi	*Tools list*
Wireless network	Wifi	*Wireless Types*
Conference map	Map Profile	*Maps list*
Conference timetable	Timetable Profile	*Timetable list*
Talks share folder in the server	Folder Profile	*Folder list*
Tourist map	Map Profile	*Map list*
Flight schedule	Timetable Profile	*Timetable list*
Airport map	Map Profile	*Map list*

Table 6-13 Values of the Tools Element for Modelling the Database

Tool – model database for each tool (desktop, printer, laptop, etc)		
Values	**Sensor**	*Attribute in Database*
Desktop 2	Wifi	*Name*
Printer 2	Wifi, Bluetooth	*Name*
Wireless network	Wifi	*Connection Types*
Connected to projector	Assign	*Function*
Public conference	Assign or Network	*Owner*
Reception at conference	Bluetooth	*Location Range*
Status is in used	Network	*Status*

Table 6-14 Values of Each Tool or Device Assigned Sensors and Attributes for Modelling the Database

Tool Database						
ID	**Name**	**Connection type**	**Owner**	**Location Range**	**Screen size**	**Status**
1	PDA	Wifi	Adam	100 meters	3.8 inches	Free
2	Nokia 6680	Bluetooth	Adam	50 meters	2.5 inches	Free
3	Desktop 2	Wifi	Public conference	100 meters	90 inches	Busy
4	Printer 2	Wifi, Bluetooth	Public conference	50 meters	-	Free
….	….	….	….	….	….	….

Table 6-15 The Tool Database Hold Info for Each Device

From Table 6-13, the developers can design the profiles in order to provide the information that cannot be sensed. The profiles are created to hold descriptive information about the values separately from the context element. By separating the descriptive information about the value of the attribute in database, it allows the value to be changed, updated, edited and added more easily without affecting the context model (see Table 6-16).

Map Profile				
Map ID	**Name**	**Source**	**Location Range**	**Period**
1	Map of Conference	www.mobile06.com/map.html	At Hamburg conference	19 June 2006
2	Tourist Map	www.hamburg.com/map.html	At Hamburg	forever
3	Hamburg Airport Map	www.HBAirport.de/map.html	At Hamburg	forever
….	….	….	….	….

Table 6-16 Example of the Map Profile

Timetable Profile				
Timetable ID	**Name**	**Source**	**Location Range**	**Period**
1	Conference Timetable	www.mobile06.com/timetable.xml	At Hamburg conference	19 June 2006
2	Flight Timetable	www.flights.de/timetable.xml	At Hamburg Airport	20 June 2006
….	….	….	….	….

Table 6-17 Example of the Timetable Profile

Folder Profile				
Folder ID	**Name**	**Source**	**Location Range**	**Period**
1	Conference Talks folder	www.mobile06.com/talks/	At Hamburg conference	19 June 2006
....	....	....	....	....

Table 6-18 Example of the Folder Profile

Tools Database					
ID	**Tool ID list**	**Map ID list**	**Timetable ID list**	**Folder ID list**	**Name list**
1	-	1	1	-	Map of conference, Conference timetable
2	3,4	-	1	1	Desktop2, printer2, Map of conference, Conference timetable
3	-	-	1	1	Map of conference, Conference timetable
4	-	-	-	2	Tourist Map
....	....	....	....	....	....

Table 6-19 Tools Database Stores Sets of Values of Information

Community		
Values	**Sensor**	***Attribute in Database***
Conference attendants	Bluetooth or NRFID	*Users list, numbers of people in community*
Talk presenter	Bluetooth or Schedule	*Users list*
Bob	Bluetooth	*Users list*
John	Bluetooth	*Users list*
People, more than 20 person	Bluetooth	*Users list, numbers of people in community*

Table 6-20 Values of the Community Element

Similar to the tools element, the community context element is composed with information of different users. The value of each community will therefore refer to different users' information. The information of the community can be general information about the community where the values in the community do not have a strong relationship with the user (such as passers-by or conference attendants). Instead of considering who is in the community, the number of people in the community shows the density of the community and can be more useful than information about unknown users. The number of people in the community is therefore added as an attribute name in the community database (see Table 6-21).

Community Database			
ID	User ID list	Name list	Number of users
1	2,5,12, 20, unknowns	John, Bob, Sam, Dan A, Sarah, unknowns	>10
2	41	Dan A, unknowns	-
3	2,5	John, Bob, unknowns	-
4	unknowns	Unknowns	>20
5	2, unknowns	John, unknowns	-
6	3, unknowns	Bob, unknowns	-
....	....	....	....

Table 6-21 Community Database Stores Sets of Values of Information

This step allows the developers to model and create a database for each context element. The key in modelling and storing the database is the identity key. It is created for every value of element. The identity key is used to refer to the set of information. Furthermore, the concept of identity key is used throughout in sensor databases and profile databases as well.

6.1.5 Step 5: From Context Elements to Reasoning

The elements such as rule, role, objective and outcome are normally difficult to infer from sensors in real time. Thus, the values for these elements are inferred from the history of situations. The basic history of situations is based on the situations identified from the scenarios in Step 2 (i.e. Situations 1-5 in Figure 6-1 to Figure 6-4). In order to create the database for the history of situations, first the profile for a role is created following the relationship between elements in the

context model. The role is influenced by the user element, community, environment and time element respectively. The role database is created so that for every role a value is associated with the reference points of the user, community, environment and time in their database. These elements then guide the designers and developers in assigning the attributes in the database as shown in Table 6-22. Each role value has its own reference point for the context model to refer to. The history of role values can then be created. The developers use the attributes in the database with the extracted situations (see Figure 6-1 to Figure 6-4) to create values in the role database. For example, when the user Adam *(User ID is 1)* is with the conference attendants *(Community ID is 1)* at the conference reception *(Environment ID 1)* at 8:55am Monday 19 June 2008 *(Time ID 1)*, these values will be stored in the database. In real time, the sensor data is processed to find the user ID in the user database and repeat the process in the community, environment and time context element databases. These IDs can then be used to find the best match in the role database in order to get the value of the user's current role. It is not necessary that all the values have to match the values in the database. The extracted situations can create a further set of values in the database, for example see Role ID 6 and 7 in Table 6-22. These are from Situation 3 where Bob and John are treated separately but they both hold the same role as a colleague to Adam during the conference.

Role Database					
Role ID	**Role Name**	**User ID**	**Community ID**	**Environment ID**	**Time ID**
1	Conference Attendant	1	1	1	1
2	Listener	1	2	2	2
3	Colleagues	1	3	3	3
4	Tourist	1	4	4	4
6	Colleagues	1	5	3	6
7	Colleagues	1	6	3	6
….	….	….	….	….	….

Table 6-22 Role Database Stores Sets of Reference Points to the Information

As mentioned in the design tool, the value of rule is referenced to the value of the role. The rule database is created as shown in Table 6-23.

Rules Database		
Rule ID	**Rule Name**	**Role ID**
1	See the lists of talks on timetable	1
2	See information but not presenter's note	2
3	See Colleagues' locations	3
4	See public information about Tourism includes map and tourist info	4
….	….	….

Table 6-23 Rules Database Stores Sets of Reference Points to the Information

The objective database is created as shown in Table 6-24 so that the value can be assigned for each context model in the context model database in Table 6-26.

Objective Database	
ID	**Objective Name**
1	Decide which talk to attend
2	Get info about talk
3	Get colleagues' locations on the map
4	Get info about nearby tourist places
….	….

Table 6-24 Objective Database of Possible Value of the Objective from the Situations

From the objective descriptions, the outcome is the result of the user's efforts to meet the objective. As described in Section 4.5, for every set of values in the outcome context element database, the objective value is paired with the ideal outcome. The database for the outcome element is shown in Table 6-25.

208

Outcome Database		
ID	**Outcome Name**	**Objective ID**
1	Attend the selected talk quickly	1
2	Got info about talk to make note efficiently	2
3	Got colleagues' locations and directions info on the map	3
4	Got info about nearby tourist places and direction to selected one	4
….	….	….

Table 6-25 Outcome Database Stores a Reference Point to the Objective Values

As described in Section 4.5, the value of the objective may be inferred from the history of the context model. The context model holds IDs of context elements. The IDs are the reference points to the set of values of the other elements in the context model (user, community, role, rule, tools, environment and time respectively) as shown in Table 6-26. The outcome ID of each situation is assigned in the Activity Theory database according to its objective value in the outcome database.

Activity Theory Database									
ID	User ID	Community ID	Role ID	Rule ID	Tools ID	Environment ID	Time ID	Objective ID	Outcome ID
1	1	1	1	1	1	1	1	1	1
2	1	2	2	2	2	2	2	2	2
3	1	3	3	3	3	3	3	3	3
4	1	4	4	4	4	4	4	4	4
5	1	4	5	5	5	5	5	5	5
....	....	....	....	....	....	....	....	....	....

Table 6-26 Activity Theory Context Model Database Stores Sets of Reference Points to the Information

This step shows the influence that the context model has on the reasoning method. By referring to the identity of a set of values instead referring to the value itself, the architecture provides flexibility in changing the value internally and in systematically referring to the data.

6.1.6   Step 6:   From Outcome Context to Selected Application and Required Context

From the defined situations in Step 2, the extracted features through which the context-aware system is supposed to support the user in each situation are analysed further in order to decide an appropriate application for each situation. The developers decide to use an existing application or implement a new one. For example, the existing application is the conference schedule and the new

application could be an extended version of the conference schedule that highlights an interesting talk. Furthermore, the context information that a user requires to complete the task is assigned to use with the application. For example, in Situation 1, as Adam's personal device is a PDA, the context-aware system is required to reduce information about the conference schedule. The developers assign the conference schedule application. In order to narrow down the information to be shown on the PDA screen, the context-aware system requires information about the current user's personal device, user interests, current time and conference schedule.

Application Database			
Application ID	Outcome ID	Application	Context
1	1	Conference Schedule with highlight	User (device, interest), Time (HH, MM) Tools (Conference schedule)
2	2	Talk Information	User (device), Environment (room), Time (HH, MM) Tools (Conference schedule, talks folder)
3	3	Users locator	User (device, interest), Community (Relevant users' location), Tools (Conference map)
4	4	Tourist Guide	User (device, flight booking, Tourist interest), Environment (Area), Tools (Tourist map)
5	5	Shopping reminder	User (Device, Duty free shopping list), Environment (Area), Tools (Airport map)
....	....	....	....

Table 6-27 Application Database Stores a Reference Point to the Outcome in Different Situations

The sets of values in the databases and profiles created from the extracted situations are used as a basic guide for the developers to generate further possible sets of values in order to be able to support the user in similar situations.

This section applied six design steps to the simple scenario where the user is not in a high pressure environment. Moreover, some elements of context do not need to be taken into account in the simple scenario. The design tool introduces systematic steps to the design process. Each step guides the designers with questions about the situation that the designers should concentrate on in order to meet the user requirements. The result of the context model for each situation is well structured and simple to follow. This facilitates the communication between the designers and developers.

A more complex scenario is introduced in the next section to illustrate how the design tool is applicable to more complex scenarios.

6.2   *Scenario 2: The Hospital A&E Department*

The scenario here is extracted from material collected during an 18 month study of observing a receptionist at the Accident and Emergency department of a hospital in London [O'Neill *et al.*, 2004]. It has been shortened for our purposes here.

"Sara is a receptionist at the Accident and Emergency (A&E) department of a hospital in London. Everyday the reception and waiting area is very busy. There are 2 printers at the reception desk that also serve other receptionists in the department. One printer is assigned to print the Case Card and another is assigned to print a set of sticky labels with the patient's details for sticking on to blood

samples, x-ray requests, appointment books, etc. There is a list of telephone numbers on a piece of paper next to the reception desk in the waiting area.

Sara has to handle all sorts of enquiries such as information about how to register with a GP, what to do next after check out, directions around the area, look for the beeper numbers to beep the doctors when their take-away arrives, and answer the phones that seem to be ringing all day.

Sara is required to book in the patients to the computer PAS system which holds the information about the patient and generates a print out case-note (file) to be filled in by the doctor. The first question is 'Have you been here before?' If yes, they will be on the system. However, sometimes there are lots of people who have the same name, or several entries with the surname spelt differently which may relate to the same person. Also, the address, phone number etc will often have changed. The date of birth is a key "demographic" enabling identification of an individual in ambiguous situations. If the patient has not visited the department before, a new entry will be made on the database. When a patient enters the department, the first person they see will be a triage nurse, who will ask them about their problem, and fill in a short form which they hand to the receptionist when booking in. The patient's name is already on this form, together with a short summary of their complaint. The reception therefore does not need to ask the name, if legible. If the patient is brought in by ambulance, the patient is booked in by the paramedic, using details from the pink form that they will have previously filled in. The receptionist then uses the pink form to book the patient into the PAS system instead of asking the patient who may be unconscious or too badly injured to answer questions. There will be a queue to book in with the receptionists because it is a time consuming process and the receptionist may be interrupted by a phone call, a language barrier or other enquiries.

When the patient is discharged from the department, either to home or admitted to a ward, the rest of the details from the case notes (as filled in by the doctor) are entered onto the PAS by the receptionist. This happens after the forms are collected from Majors by a receptionist. Not infreqeuently, the receptionist cannot admit a patient on the database because there is information missing on the Case Cards such as the admitting consultant's name."

The scenario shows the complexity of activity and the pressure that the user is under. The scenario is analysed further in the next step in order to extract the situation for each activity.

6.2.1 Step 1: Defining Scenarios in which the System will be Applied

The field study is complex and contains a 30 page description of the observations on different days. There is no real structure as the data is a story of what happens in a hectic hospital on different days. By keeping the context model in mind during analysis of the field study, for every situation where the designers consider that context awareness can support the user, they draw a simple context model next to the paragraph that describes the situation. The simple context models from different paragraphs can then be analysed and grouped according to their objectives. As the field study is long and describes certain objectives repetitively, the situation for each objective is created from combining information from different paragraphs in the field study that have been annotated with a simple context model with the same objective value. The designers use the situations with the different objectives to create a new descriptive scenario for the field study as shown in Section 6.2. This is to avoid a long unstructured and repetitive scenario that will be used to communicate between the designers, developers and users.

As the scenario provides non repetitive situations for each objective, it can then be used to extract situations for each activity. As mentioned in the previous section,

the situations visualise how context awareness can support the user in achieving their objectives. The new situations can then be modelled into different models following the elements in the context model. This will provide a simpler form for easier referencing with the implementers about the situation. This will also help designers analyse the situation and design the functions of the application to support each situation. The new situations can also be used to describe the use of context awareness to the users if a participatory design process is to be pursued.

6.2.2 Step 2: Define Situations where Context Awareness Can Support User

The user that the system is supporting in this scenario is Sara who works at a reception desk at the A&E department. The situations are extracted from the scenario for each activity where the system will support the user as shown below:

6.2.2.1 Situation 1

Sara works under a lot of pressure completing multiple simultaneous tasks. She was interrupted by the take-away delivery man. He asks her to get Dr Rach to come and get his food while she is filling information into the PAS on desktop1. Sara has a hard time looking for Dr Rach's beeper number from the list of telephone numbers on a piece of paper behind her. Instead, she turns to the PDA information assistant which holds a telephone book database. Based on the name of the doctor on the take-away receipt, it auto detects the beeper number and allows Sara to send a beeper message to the Dr Rach.
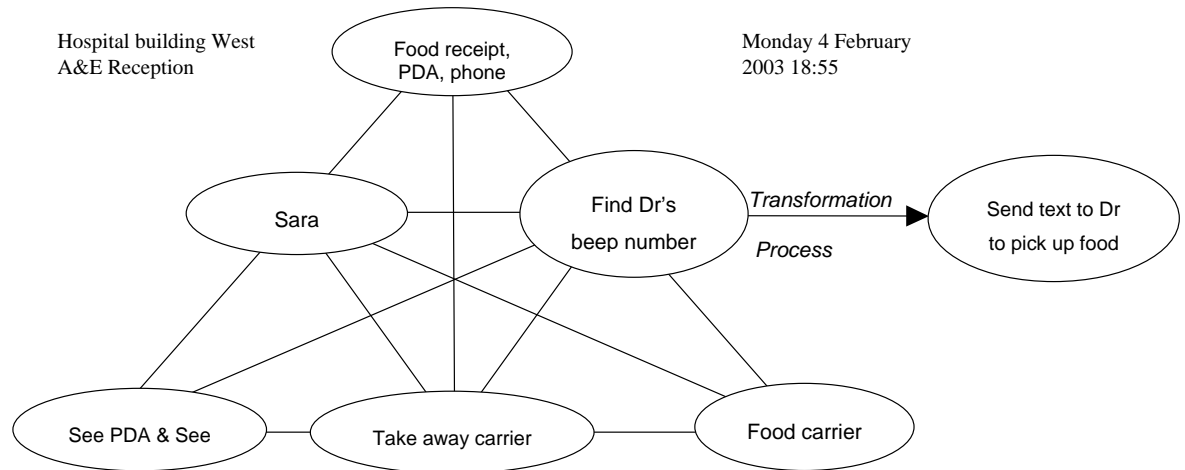
Figure 6-5 Context Model of Situation 1- Get doctor to pick up his food

The same process that has been applied to the simple scenario in the previous section is applied here. The designers follow the six questions in order to analyse the situation and decide on the context-aware supports for the user.

Question 1:   What is the activity for the context-aware system to support in this situation?

As the aim of this situation is for Sara to be able to find the doctor's beeper number, the activity in this situation is "sending the text to doctor's beeper to pick up food". This leads to the answer for question in step 2.

Question 2:   What are some actions that a user may need to perform?

In order to send a text to the doctor, the goal is to search for the doctor's beeper number that is in the long beeper number list. From this action, the operations for the next questions can be answered.

Question 3:   What are some operations that a user may need to perform?

The operations that the user need to complete in order to search the doctor's beeper number efficiently are first to get the doctor's name from the receipt, second to open the beeper number list, third to reorder the list by doctor's first name and last to look through the list to get the number from one that matches the doctor's name.

The designers assign a level of computing support that will be suitable for the activity, actions and operations from the previous questions:

Question 4:   What operation level supports is the system going to provide?

- Open the beeper number list >> *Passive*

- Reorder the list >> *Active*

- Get doctor's name >> Active

- Look through the beeper number list >> *Active*

Question 5:   What action level supports is the system going to provide?

- Search for the doctor beeper number >> *Active*

Question 6:   What activity level support is the system going to provide?

- Sending the text to doctor's beeper to pick up the food

The system provides active supports to the user by automatically getting the doctor's name from the receipt, for example by scanning a barcode or sensed RFID tag on the receipt, and searching for the best match from the beeper number list. The best results are shown on the screen for Sara to select the number to send the message for the doctor to pick up the food as a passive support from the system. The assistant should therefore take tools information (receipt with the customer's name, beeper number list on the system) into account in order to show only Dr Rach's beeper number on the screen allowing Sara to concentrate on making sure the number is correct and sending a text to tell Dr Rach that the food has arrived at reception.

The next situation is when Sara has to book a patient into the department to be treated.

6.2.2.2   Situation 2

Sara is asking questions in order to book a patient in. It is a time consuming process in order to get all the details about one patient. Behind the patient is a paramedic waiting to book another patient in. Instead, she turns to a booking in assistant. It detects information about the patient and fills in the fields in the form to reduce the questions that Sara has to ask each patient. But when the paramedic checks in a patient, the assistant detects the information from the pink form and automatically prints the Case Card for that patient.
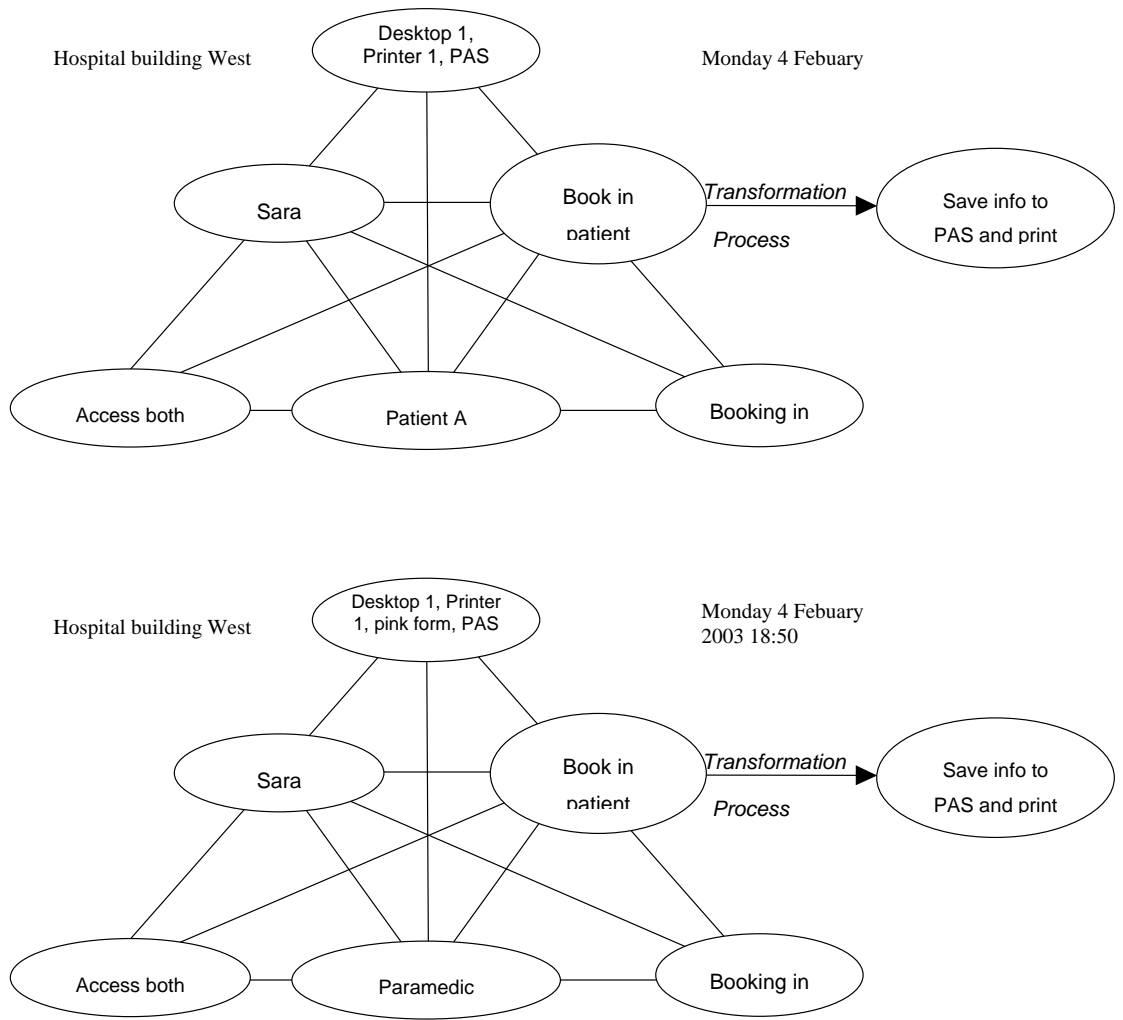
Figure 6-6 Context Model of Situation 2 - Booking in a patient

For this situation, there are two sets of context model that support the same activity of booking in. The first situation is when the patient is booking herself in. The second situation is when the paramedic is booking the patient in. Following the same questions as the previous situation, the answers are:

Question 1:   What is the activity for a context-aware system to support in this situation?

       - Register the patient on PAS database and print a Case Card

Question 2:   What are some actions that a user may need to perform?

       - Find out whether the patient is already registered or not

If the patient is not in the PAS database

       - Fill the patient detail onto the PAS database

       - Order a Case Card to be printed

Question 3:   What are some operations that a user may need to perform?

The operations that are required in order to find out whether the patient is already in the PAS or not, are

       - Open PAS system

       - Ask the patient or if the patient is unsure or unconscious:

         o Get patient date of birth and name from the patient or paramedic

         o Search if the date of birth and name are matched in the PAS database

If the patient is not in the PAS database, the operations that needed to be completed are:

- Type in the patient information such as date of birth, name, other relevant information

- Save data to the database

- Find the printer

- Select print a Case Card

Question 4:   What operation level supports is the system going to provide?

To find out if the patient is in the database or not, the operations to complete are

- Open PAS system >> *Active*

- Ask the patient or if the patient is unsure or unconscious:

o Get patient information such as date of birth and name >> *Active*

o Search if the date of birth and name are matched in the PAS database >> *Active*

If the patient is not in the PAS database, the operations that need to be completed are:

- Type in the patient information such as date of birth, name & other relevant information >> *Both*

- Save data to the database

- Select print a Case Card

- Find the printer >> *Active*

Question 5:   What action level supports is the system going to provide?

-        Find out whether the patient is already registered or not >> *Active*

If the patient is not in the PAS database

- Fill the patient detail onto the PAS database >> *Both*

- Order a Case Card to be printed

Question 6:   What activity level support is the system going to provide?

-        Register the patient on PAS database and print a Case Card >> *Both*

By using context awareness to support Sara in this situation, the system provides *active supports* by automatically getting the patient information and searching the database to see if the patient is already in the database or not. If the data of the patient is found in the database, it will provide passive support by showing the patient data on PAS so Sara can order a Case Card to be printed on the assigned printer that the system automatically found in the network. If the data of the patient is not found, the system provides *both active and passive* supports by automatically filling the patient details on the PAS and showing it on PAS so Sara can order a Case Card to be printed.

From the object and outcome elements in this situation, the information assistant should be able to detect information about the patient and automatically fill in the PAS form where the information is available. The receptionist will only have to ask for the information that is missing from the patient. Therefore when the assistant detects the patient, it should take community information (patient's name,

phone number, date of birth, address) into account in order to fill in the PAS. When the assistant detects a paramedic, it should take tools information (information on the discovered pink form – according to the name of the paramedic who created the pink form) into account in order to fill in the PAS. Obtaining information about the user can be time consuming process in the noisy and busy environment. Moreover, the patient may not speak fluent English or not understand English at all. For a system to obtain the information by using context awareness will help Sara to concentrate on other important activities.

6.2.2.3 Situation 3

Sara is trying to find the consultant's name on the Case Card in order to check out a patient. It is a time consuming process in order to find out information that is missing on the Case Card.
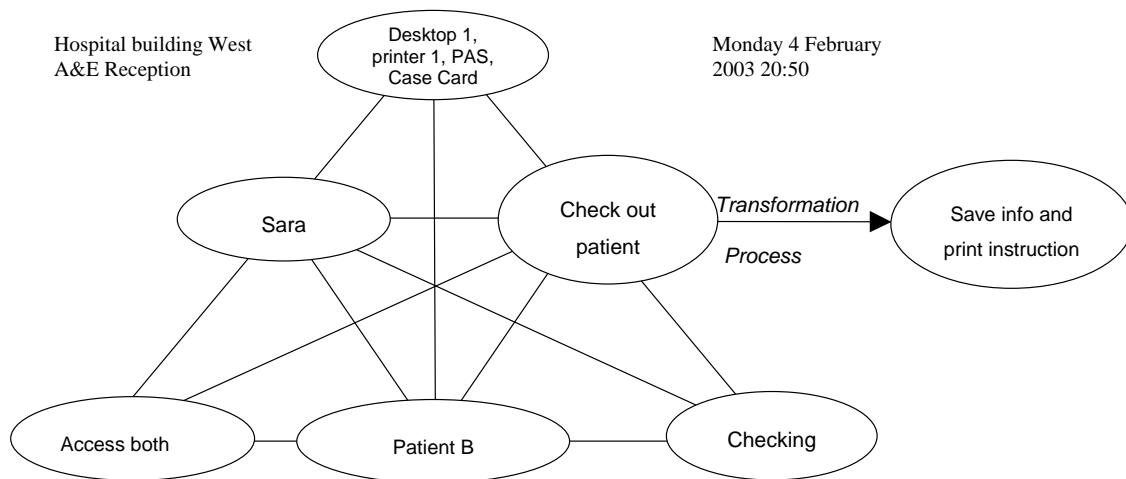
Figure 6-7 Context Model of Situation 3- Checking out patients

From this situation, the six levels of activity questions are answered below:

Question 1:   What is the activity for the context-aware system to support in this situation?

- Get patient checked out of A&E with relevant information

Question 2:   What are some actions that a user may need to perform?

- Update information about a user check out in PAS

- Get instruction for the patient to take home

Question 3:   What are some operations that a user may need to perform?

- Open the PAS

- Type information about user from the Case Card onto PAS

- Find the printer for printing instruction

- Print the required instruction for the patient

Question 4:   What operation level supports is the system going to provide?

- Open the PAS *>> Active*

- Type information about a user from the Case Card onto PAS *>> Active*

- Find the printer for printing instruction *>> Active*

- Print the required instruction for the patient

Question 5: What action level supports is the system going to provide?

- Update information about a user check out in PAS >> *Both*

- Get instructions for the patient to take home >> *Active*

Question 6: What activity level support is the system going to provide?

- Get patient checked out of A&E with relevant information

From the object and outcome elements in this situation, the information assistant should be able to find out the information on the Case Card of Patient B from the history of who edited the Case Card that was automatically recorded and the values from the sensor attached on the Case Card. By using context awareness to support Sara in this situation, the system provides *active supports* by automatically getting the patient information on the Case Card and filling the information on to PAS. The system will then provide *passive support* by showing the patient data on PAS so Sara can order an instruction to be printed on the assigned printer that the system automatically found in the network. As a result, the patient's instructions or prescription are printed for the patient to take home with them.

The situations are used in the next section for further analysis of information to take account of context elements in the context model of each situation.

6.2.3   Step 3:   From Situation to Elements in Context Model

The designers proceed to concentrate on expanding the list of information for each element of context based on the definitions of context elements in Section 4.3.

6.2.3.1   Situation 1

Context Elements	Values from Situation
Environment	A&E Reception, building West, Hospital, London, UK
Time	Monday 4 February 2003 18.55
User	Sara owns Nokia 6680
Tools	Food receipt, PDA, Phone number lists
Community	Take away carrier and People
Roles	Food carrier assistant
Rules	Use PDA to see the phone list
Objective	Find DR Rach's beeper number
Outcome	Send text to Dr Rach to pick up the take away

Table 6-28 Values are Identified for Context Elements in Situation 1

6.2.3.2   Situation 2

As mentioned, there are two context models that support the same activity in this situation. The first model is when the patient is checking herself in. The second model is when the paramedic is checking the patient in. The paramedic has already gathered information about the patient before the patient arrives at A&E. The information about the patient is filled in the pink form which refers to one of the tools available in this situation. Therefore with context awareness support, the information in the pink form can be transferred to the PAS system without explicit input from Sara.

Context Elements	Values from Situation
Environment	A&E Reception, building West, Hospital, London, UK
Time	Monday 4 February 2003 18.50
User	Sara owns Nokia 6680
Tools	Desktop 1, Printer 1, PDA, PAS
Community	Patient and People
Roles	Booking in Assistant
Rules	Access both desktop1, desktop2, printer 1, printer 2 and PAS
Objective	Fill information about patient into PAS
Outcome	Save information about patient in PAS and print a Case Card

Context Elements	Values from Situation
Environment	A&E Reception, building West, Hospital, London, UK
Time	Monday 4 February 2003 19.30
User	Sara owns Nokia 6680
Tools	Desktop 1, Printer 2, PDA, Pink form, PAS
Community	Paramedic and People
Roles	Booking in Assistant
Rules	Access both desktop1, desktop2, printer 1, printer 2, Pink form and PAS
Objective	Booking in patient
Outcome	Fill & save information about patient in PAS and print a Case Card

Table 6-29 Values are Identified for Context Elements in Situation 2

6.2.3.3 Situation 3

Context Elements	Values from Situation
Environment	A&E Reception, building West, Hospital, London, UK
Time	Monday 4 February 2003 20.50
User	Sara owns Nokia 6680
Tools	Desktop 1, Printer 1, PAS, Case Card
Community	Patient and People
Roles	Checking out Assistant
Rules	Access both desktop1, desktop2, printer 1, printer 2, Case Card
Objective	Discharge patient
Outcome	Fill & save information about patient from Case Card into PAS and print a instruction for patient (prescription, appointment)

Table 6-30 Values are Identified for Context Elements in Situation 3

At this stage, the designers concentrate on meeting the user requirements. In the next step the designers will bring the implementers into the process in order to determine the feasibility of gathering the information required for each context element listed in Table 6-28, Table 6-29 and Table 6-30.

6.2.4 Step 4: From Context Element to Sensors and Profiles

Following on from the information in the tables, the implementers inform the designers of what technology can be used to get information and to what level of

information. For example, GPS is used to get geographic coordinates, which can then be translated to information such as country, town etc. In this study, for occasions where more specific information and more accuracy is required such as at the reception desk, a RFID reader is adopted. As a result, we then get the context element databases and profiles below.

Environment		
Values	**Sensor**	***Attribute in Database***
	Thermometer	*Condition*
A&E Reception desk	RFID, Bluetooth	*Room*
Building West	GPS	*Building*
Hospital	GPS	*Area*
London	GPS	*Town*
UK	GPS	*Country*

Table 6-31 Values of the Environment Element from Different Situations

The level of information required from the tables in the previous step for each context element guides the developers to design the database for the element as shown in Table 6-32.

Environment Database						
Environment ID	**Room**	**Building**	**Area**	**Town**	**Country**	**Condition**
1	Reception	Building West	Hospital	London	UK	-

Table 6-32 Environment Database Stores Sets of Values of Information

The time element is designed similarly to the previous scenario. As a result, the level of time information is shown in Table 6-33. As a result of these, the database for the time element is as shown in Table 6-34. The attribute values in the time element can be null to give flexibility in storing time. For example the rest of the attributes can assigned to null value except *Period of Day* which is assigned to Morning to represent the situation that happens every morning. Each set of values of the time element is added to the database when the new scenario is found by the system during run time as mentioned in Section 3.6.1.

Time		
Values	**Sensor**	***Attribute in Database***
Monday	System clock	*Day of the week*
4	System clock	*Date of the month*
February	System clock	*Month*
2003	System clock	*Year*
Afternoon	Interpretation	*Period of Day*
18	System clock	*Hour of the day*
19	System clock	*Hour of the day*
20	System clock	*Hour of the day*
55	System clock	*Minute of the hour*
00	System clock	*Minutes of the hour*

Table 6-33 Values of the Time Element from Different Situations

Time Database							
Time ID	**DOW**	**DOM**	**Month**	**Year**	**Period of Day**	**HH**	**MM**
1	Monday	4	February	2003	Afternoon	18	55
2	Monday	4	February	2003	Afternoon	18	50
3	Monday	4	February	2003	Evening	19	30
3	Monday	4	February	2003	Evening	20	50
…..	….	…	…	…	….	….	….

Table 6-34 Time Database Stores Sets of Values of Information

In this case, Sara only has one personal device which is her mobile phone, information about which may be gathered by the system during run time through Bluetooth or from Sara's profile in the database.

User		
Values	**Sensor**	*Attribute in Database*
Sara	Profile or Log in info	*Name*
Nokia 6680	Bluetooth or user profile	*Personal Device*

Table 6-35 Values of the User Element from Different Situations

User Database		
User ID	**Name**	**Personal Devices**
1	Sara	Nokia 6680
…..	**….**	**…**

Table 6-36 User Database Stores Sets of Values of Information

As a hospital is a public space and has a variety of equipment to support several tasks for different people, the developers follow the information in the tables from the previous step to generate different values that are required for the tool element in the scenario. The values are assigned to the sources where the information can be gathered during run time as shown below.

Tools		
Values	**Sensor**	*Attribute in Database*
desktop 1	Bluetooth	*Tools list*
Printer 1	Wifi	*Tools list*
desktop 2	Bluetooth	*Tools list*
Printer 2	Wifi	*Tools list*
PDA	Wifi	*Tools list*
Wireless network	Wifi	*Wireless Types*
Food receipt	Barcode, NRFID	*Tools list*
Phone number list	Information Profile	*Information list*
Pink Form	Barcode, NRFID	*Tools list*
Hospital map	Map Profile	*Map list*
Registration Instruction	Information Profile	*Timetable list*
PAS	Information Profile	*Information list*
Case Card	Barcode, NRFID	*Tools list*

Table 6-37 Values of the Tools Element from Different Situations

As mentioned earlier, the profiles are created by the developers when they think there is no suitable sensor to gather the information about the context element directly. In this case, the developers assigned Information Profile to some tools' value. This is because these tools here are different sources of information, which in this case the developers could not find suitable sensor to sense this information. The developers created an information profile to store information about different

types of information that is used as a tool element in the context model from different scenarios. The Information Profile (see Table 6-41) is a database that holds information of what information is available, where the information can be accessed, who is the owner, at which location it is available, etc. This is done so that during run time, the profile can be referred to by the system in order to gather information about the available tool, which at that time cannot be directly gathered from sensor, in real time situations. For example, the sensor can gather information about user's current environment and who the user is during run time. These information is then used to refer to what types of Information Tool that are available from the Information Profile by matching the information about user with the Owner of the information in the profile and information about user's current environment with the Location Range of the information in the profile.

Tool – model database for each tool (desktop, printer, laptop, etc)		
Values	**Sensor**	*Attribute in Database*
Desktop 1	Wifi, Bluetooth, NRFID	*Name*
Printer 1	Wifi, Bluetooth	*Name*
Desktop 2	Wifi, Bluetooth, NRFID	*Name*
Printer 2	Wifi, Bluetooth	*Name*
Wireless network	Wifi	*Connection Types*
PDA	Wifi, Bluetooth, NRFID	*Name*
A&E Reception	Assign or Network	*Owner*
Reception at A&E	Bluetooth	*Location Range*
Status is in used	Network	*Status*

Table 6-38 Values of Each Tool or Device

Tool Database						
ID	Name	Connection type	Owner	Location Range	Screen size	Status
1	PDA	Wifi	A&E Reception	100 meters	3.8 inches	Free
2	Nokia 6680	Bluetooth	Sara	50 meters	2.5 inches	Free
3	Desktop 1	Wifi, Bluetooth	Public Hospital	100 meters	20 inches	Busy
4	Printer 1	Wifi	Public Hospital	50 meters	-	Free
5	Desktop 2	Wifi, Bluetooth	Public Hospital	100 meters	20 inches	Busy
6	Printer 2	Wifi	Public Hospital	50 meters	-	Busy
7	Food receipt	NRFID	Doctor N	1 meters	-	Free
….	….	…	….	….	…..	…

Table 6-39 Example of the Tool Database for Each Device

Map Profile					
Map ID	Name	Source	Owner	Location Range	Period
1	Hospital map	www.hospital.com/map.html	Public	At Hospital	forever
…..	….	…	….	….	…

Table 6-40 Example of the Map Profile for Each Map

In the hospital scenario, there are different types of information that are available for different users from patients to nurses to doctors. Different people have different levels of accessibility. This is because privacy and security are important issues in the hospital scenario. This assists the developers to design information as shown below.

Information Profile					
Information ID	**Name**	**Source**	**Owner**	**Location Range**	**Period**
1	Phone number list	www.hospital.com/contact.xml	Receptionist	At reception	4 February 2008
2	Registration Instruction	www.hospital.com/registration.html	Public	At hospital	4 February 2008
3	PAS system	www.hospital.com/PAS	Receptionist	At hospital	4 February 2008
…..	….	…	….	….	...

Table 6-41 Example of the Information Profile that Holds Descriptive Information for Each Information Tool

In the complex scenario, by following the user requirements, there is a possibility of developing more than one context model per situation/activity. For example, as shown in scenario 2, there are two context models that support the same activity. The first model is when the patient is checking herself in. The second model is when the paramedic is checking the patient in. As a result, these context models are both stored in the history of context database to be used to trigger support for the user in real time. Moreover, the history of context elements such as tools and community are also created as shown in Table 6-42 and Table 6-44 respectively.

Tools Database				
ID	Tool ID list	Map ID list	Information ID list	Name list
1	1, 7	-	1	PDA, Desktop1,Printer1, Food receipt, phone list
2	3, 4	-	3	Desktop1, printer1, PAS
3	3, 8	-	3	Desktop1, printer1, Pink form, PAS
4	3, 9	-	3	Desktop1, printer1, PAS, Case Card
….	….	…	….	…

Table 6-42 Tools Database - Stores sets of values of information about tools in different situations

Community		
Values	Sensor	*Attribute in Database*
Take away carrier	Bluetooth	*Users list*
Patient	RFID	*Users list*
Paramedic	RFID	*Users list*
…..	…..	…..

Table 6-43 Values of the Community Element

Community Database			
ID	User ID list	Name list	Number of users
1	5	Take away carrier	-
2	41	Patient A	-
3	50	Paramedic	-
4	34	Patient B	-
….	…..	…	…..

Table 6-44 Community Database - Stores sets of values of information about community

These tables help developers design the database for each sensor, profile and context element. All the values in the database are assigned with a unique identity (ID). This will allow the set of values to be referable in different databases or reasoning processes.

6.2.5   Step 5:   From Context Elements to Reasoning

Following on from the reasoning process in Section 4.5, the situations in steps 2 and 3 are used in order to assign the value in the role database for different situations from the user requirements as shown in Table 6-45. As a result of having two context models for Situation 2, the roles are created for each context model. At the same time, there are different sets of rules for the user in the context model in this situation because of the importance of being able to access the Pink Form from the ambulance if the paramedic is booking the patient in. When a patient walks in off the street and checks herself in, there is no Pink Form.

Role Database					
Role ID	Role Name	User ID	Community ID	Environment ID	Time ID
1	Food carrier assistant	1	1	1	1
2	Booking in assistant for patient	1	2	1	2
3	Booking in assistant for ambulance	1	3	1	2
4	Checking out assistant	1	4	1	3
…..	….	…	….	….	…..

Table 6-45 Role Database - Stores sets of reference points to the information that have influence on roles

As described in Section 4.5, each set of values of the rule is paired with the reference point to the value of the role; the rule database is created as shown in Table 6-23.

Rules Database		
Rule ID	Rule Name	Role ID
1	Access PDA, phone book and name on the receipt	1
2	Access both Desktops, printers and PDA and PAS	2
3	Access both Desktops, printers and PDA, Pink form and PAS	3
4	Access both Desktops, printers and PDA, Case Card and PAS and print instruction on printer 2 only	4
…..	….	...……

Table 6-46 Rules Database - Stores sets of reference points to the information that have influence on rules

Objective Database	
ID	**Objective Name**
1	Find Dr's beeper number
2	Check in patient
3	Check out patient
….	….

Table 6-47 Objective Database - Stores the objective values from different situations

From the objective descriptions, the developers can easily pair the reference points of objectives with the expected outcomes. The database for outcome element is shown in Table 6-25.

Outcome Database		
ID	**Outcome Name**	**Objective ID**
1	Send text to Dr to pick up food	1
2	Save info to PAS and print a Case Card	2
3	Save info and print instruction for patient	3
….	….	…

Table 6-48 Outcome Database - Stores reference points to objective values that have influence on outcome

The values of the elements in the situations are stored in the Activity Theory context model database to be used to infer the current user's objective as shown in Table 6-49.

Activity Theory Database									
ID	User ID	Community ID	Role ID	Rule ID	Tools ID	Environment ID	Time ID	Objective ID	Outcome ID
1	1	1	1	1	1	1	1	1	1
2	1	2	2	2	2	1	2	2	2
3	1	3	3	3	3	1	2	2	2
4	1	4	4	4	4	1	4	3	3
...	...	...	...	...	...	...	...	...	...

Table 6-49 Activity Theory Context Model Database - Stores sets of IDs of information that have influence on objectives

The set of reference points (IDs) of elements that have influence on the user's objectives in different situations can then be stored in the Activity Theory context model database as shown in Table 6-49. The information of these elements is to be used in real time in inferring about a user's current objective. The next step is to assign an application or service to each outcome so that when the situation occurs in real time the system can provide a suitable support to the user.

6.2.6 Step 6: From Outcome Context to Selected Application and Required Context

From the defined situations in Step 2, the extracted features that the context-aware system is supposed to support for the user in each situation are analysed further in order to decide an appropriate application for each situation. For Situation 1 (the take-away scenario), the assistant will send the text to the name on the receipt with the selected phone number from the phone list.

Application Database			
Application ID	**Outcome ID**	**Application**	**Context**
1	1	Take away called assistant	Tools (PDA, name on the receipt, telephone Book)
2	2	Checking in assistant	Community (Relevant user's information), Tools (Desktop1, printer 1, pink form), Time (HH, MM)
3	3	Checking out assistant	Community (Relevant user's information), Tools (Desktop1, printer 1, Case Card), Time (HH, MM)
…	…..	….	….

Table 6-50 Application Database - Stores a reference point to the outcome in different situations

This section shows how the design tool can be used with a more complex scenario. Moreover, Situation 2 gives an example of how the context tool can be used to create different context models for the situations in a scenario. The design tool guides designers to model the context based on user requirements. The designers can therefore identify as many context models for the situation as they wish. The context models support the same activity but in different circumstances such as

242

different tools available, different community, etc. These context models are then stored in the history of context models in order to recognise the events for which the system will provide services to the user in real time. The next section refers to the context-aware system design tool requirements described in Section 2.4.1 in order to evaluate the design tool.

6.3   *How Each Requirement is Met or Not Met in the Scenarios*

The requirements from Chapter 2 are discussed. Each individual requirement is referred to in order to show how it is met or not met by using the context model with the above scenarios:

6.3.1   To Provide Consistent Support for Shared Understanding amongst Researchers

By breaking down the scenario into several short situations for each user objective, the designers can refer to the defined situations in step 2 of the design steps in order to show the developers a simple structured model of context instead of showing them the description of the situation, which can be long and not well structured for implementation. The descriptive story can be too complicated and confusing to implement. Also, in the case of getting user involvement in the development process through some form of participatory design, the users can use both the description and a simple model of the situation in order to understand how the system is modelling and referring to context about the situation. The context model is used as a tool to help designers, developers and users in order for them to develop shared understandings about context, how the context is used in the system to infer the user's objective and how the system provides support for the user in different situations. For example, in Situation 1 of both scenarios, the context model includes the information of nine context elements that are related to each other in the same manner. The consistent context elements and their

relationships help designers, developers and users to build a structural understanding about the situations. The situations are formed in the same manner by identifying the value in 9 context elements that relate to each other in the same manner. Only the values in the elements are different to suit the situations.

6.3.2 To Identify Context Elements

The context model identifies key elements that have an influence on the user in achieving her objective. As seen in the results of the context models for the situations in step 2, the key elements that have an influence on the user's objective are identified consistently. For all the situations, suitable values from the situation are extracted to assign to the key elements in the context model in the same manner as shown in step 3. The information in the situation is broken into 9 groups of information for each element in the context model. This provides consistency in the context model used by the system. The consistency of the context model helps the users build their mental model of the system. Moreover, the context elements drive the design instead of it being driven by the technology that limits the design to what context is available. This allows the designers to concentrate on the user's requirements rather than on the availability of the technology.

6.3.3 To Demonstrate a Consistent Reasoning Method for the Interpretation about the Context

Similar to the previous requirement, the context models for the situations in step 2 show the consistent relationships between the elements. Step 5 of both scenarios demonstrates the use of the relationships in creating a profile uniformly in the same manner for both scenarios. For example, the role database is created according to the user, community, environment and time context elements respectively. Instead of assigning the role of the user directly to the inconsistent context model, the role is inferred by using consistent context elements in the same

manner. Similarly, the rule database is created separately from the context model database. For every database containing a set of rules, there is a reference point to the role of the user to whom those rules apply, instead of embedding the rules in a monolithic context model. The reasoning method that was introduced by the context model provides developers with a well structured reasoning method. With the well consistent structured reasoning method that supports both simple tour guide and hospital scenarios mentioned in Section 6.1 and 6.2 in the same way, the users build a mental model of the system successfully with the consistent structured reasoning method. If there is a mistake made by the system, the users can reduce the unexpected errors by easily correcting the system's decision through an appropriate interface of the recovery function where it provided by the system because the users understand the system. This is an opportunity for future work. This is because the studies of representation of context model to user should be done first in order to find a suitable representation format (For example, model, picture, sound, text, etc.) for the system to communicate with the user with minimal distraction from their main tasks.

6.3.4 To Show the Separation between Context and its Reasoning

Step 4 shows how the design tool provides the designers with guidance in grouping the context information into each element of context. The database of each element contains information about that context element (for example, Table 6-6, Table 6-8 and Table 6-19). The design tool helps the designers in grouping the information from the scenario before the information is used in the reasoning process. This step also supports the designers in separating the context information from its reasoning. This is because, by using the unique reference point (ID), only the references to each context element are used to reason about the situation as seen in the Objective Database (Table 6-24 and Table 6-47). Therefore the design tool supports the separation between the information about context and the reasoning method. The separation will reduce the time and effort required when a new situation is added to the system. For example, if Henry (a

new user) attends the same conference instead of Adam in scenario 1, the context model in the history can easily be reused. From scenario 1 of Adam attending conference situations in Table 6-24, first the reference point to Adam can be changed to a reference point to Henry. The reference point to community in Table 6-24 can be changed in order to relate to Henry's colleagues, if necessary. The information about Henry is created in the User database. A new profile for Henry is added to the user database, and the user profile (Table 6-11) in scenario 1 can be changed to suit Henry and added to the database of the system that supports Henry. The information about the context model of the objective in Table 6-24 can easily be reused. Instead of remodelling the whole context models, the existing values in the databases are reused and the reference points in the database guide the developers or users in editing the values. In conclusion, by separating the context information and context reasoning, it provides easier access to parts of the context information. It can then be reused in different scenarios or domains. Moreover, the reasoning about context can also be reused more easily as the new value of the context element can be changed and edited for a new user or new situations or domains.

6.3.5 To Represent the Usage of History and Time

The design tool supports using history through the time context element in the context model. The history of context is stored in the databases of sensors, context elements and context model. The context model database holds the history of context in different situations at different time. The time context element in the context model is used so that the history of context model can be stored in the database with the reference of time for each situation. The values from situations extracted from the scenarios are stored in the context model database (Table 6-26 and Table 6-49). The values in these databases can then be used to recognise the situations where the user requires support from the system. This means the values will be used in real time in order to infer the user's current objective from the

current sensed data. The context models that the system triggers in real time are also stored in the database in order to keep them in the history.

6.4   *Summary*

This chapter demonstrates the use of the context model and design tool introduced in Chapter 4. The context model and design tool are applied to two scenarios including a conference scenario commonly used in the literature and a study of a hospital. The demonstration shows that the context model and design tool are capable of aiding the designers during the design stage. The context model and design tool allow the designers to move away from a technology driven approach. Through consistently applied abstractions, they enable the designers to concentrate on the user's requirements rather than the availability of particular technology. The six design steps guide the designers in developing the consistent structure of context element and context model databases. The previous section discussed how the context model and design tool requirements mentioned in Section 2.4.1 are met or not met as demonstrated in the application to both scenarios.

Chapter 7

Implementation and Evaluation of the Architecture

This chapter describes the development of a context-aware system prototype based on the design output from the previous chapter. The prototype system is not intended to cover aspects outside our scope here, such as application GUIs or the matching algorithm. Rather, the development of the prototype is used to demonstrate the implementation of the architecture that supports the functionalities introduced by the context model, design tool and process introduced in this dissertation, and to further evaluate how well the requirements for that architecture have been met. The prototype implementation is based on the hospital scenario, which is the more complex of the two examples from the previous chapter.

The first section discusses the implementation of the prototype context-aware system for the hospital scenario used in the design processes of Chapter 6. The next section investigates how robust and reusable the implementation is by examining the feasibility of transferring the code implemented for the hospital scenario to the conference assistant scenario. In another perspective on reuse, the following section examines the use of the implemented architecture in the same domain that the architecture was implemented for in the first place – the hospital – but to support different situations. Finally, the prototype implementation is evaluated against the architecture requirements presented in Section 2.4.2.

7.1 *From Design to Implementation of the Architecture*

The architecture proposed by the context model was discussed in Chapter 5. This section demonstrates the uses of the architecture to support the context models developed for the situations of the hospital scenario through the design process in Chapter 6. The implementation of the prototype system in this section demonstrates how the architecture supports the design. The reason for implementing the prototype for the hospital scenario rather than the simple tour guide and conference scenario is to demonstrate that the architecture can support more complex context models that are influenced by several types of context. By supporting these context models, the functionalities that the context model and design process introduced to the architecture can be established. As a research prototype, this system is clearly does not intended to be used in a real hospital situation, and applications such as the PAS system are not available in this prototype. The applications that we use as examples here are "Book in patient" and "Check out patient" which are implemented to support situations 2 and 3 described in Section 6.2.

The prototype context-aware system is implemented using Java and XML. Java is designed for its cross-platform and object oriented capabilities. It is therefore alleged to have a number of advantages including the efficient reuse of code and

the elimination of undefined and architecture dependent constructs. XML is short for eXtensible Markup Language. It provides a foundation for creating documents and document systems. XML provides syntax for document markup. At the same time, it also provides the syntax for declaring the structures of documents [St.Laurent, 1998]. XML uses a set of basic nested structures to build XML documents. Since the structures can grow complex as layers and layers of detail are added, XML is readily extensible. The mechanisms for developing the structures are simple. Moreover, XML can be used on a wide variety of platforms and interpreted with a wide variety of tools. As the document structures behave consistently, parsers that interpret them can be built at a relatively low cost in a range of languages.

The prototype does not address issues of the technology of sensors, best matching algorithm and user interfaces in the context-aware system. The aim here is to demonstrate the implementation of the system architecture that supports the functionalities that the design tool introduces to the context-aware system. The consistency of the context elements in the context model introduced by Activity Theory provides a foundation for the databases in the architecture. The design output in which database structures were assigned for the context elements and profiles in Chapter 6 is used in order to implement the databases in the architecture. The development of the databases in the architecture is described in the next section.

7.1.1   Database

In this prototype, with the familiarity of the developer with XML, the databases are stored in XML. As mentioned this is a prototype, the architecture supports any forms of storage in the databases as long as it takes the concept of unique identity value. XML is used because of its set of basic nested structures, flexibility and accessibility. The design output from the previous chapter guides the design of the

XML structure for each database. The headers of the columns in the environment database in Table 6-32 are used as attributes for each set of environment data. By taking the unique reference point, every set of environment data is assigned a unique identity value – ID. In order to represent the set of values to the user, a short attribute name (nm in the XML in Appendix I) of the set of values is assigned to every set of data so that it can be presented to the user and makes sense to the user during run time. The database designed in the previous chapter is used in the same manner to produce XML files for all the databases required, for example the environment database (env.xml), time database (time.xml), user database (user.xml), tool database (tool.xml), and tools database (tools.xml – see example in appendix II). The databases are created for important elements of context that are used to reason about the user. The consistent of the databases allows easily reuse and extend the context. Therefore unlike most past projects mentioned in Section 2.2.3.3 such as CASS that model partial of context. The example of CASS context model shown in Table 2-2 can be used in our work as part of information for the environment database i.e. the result of their context model "*Goal*" can be used as a value for our condition attribute in our environment database. Even in the project such as Context Managing Framework that contains large context ontology, there is no consistency for the context model to be used in different applications. Each application requires subscribing to different part of the context ontology.

For every database, the object was created in Java for each context element or profile. Objects have two sections, fields (instance variables) and methods. Fields express what an object is. Methods show what an object does including method that allow it to edit, changed and access the values of the variable in the object. For example, object that deals with information about environment, *xmlEnv.java* – see example in appendix III.

The XML files are created for context elements and profiles and then the values from the situation can be stored in the database. Two Java programs are created in order to create, edit, remove and access data sets for the XML files. The first program is *writeXML.java*. It has methods to create XML files including editing or adding a set of values of an object for each database to the XML file. The second program is *readXML.java*. It has methods to read the XML file in order to access the values for the object in the database. It also has a method that searches through the database in the XML file and returns the object with a particular ID such as *getEnvAt().*
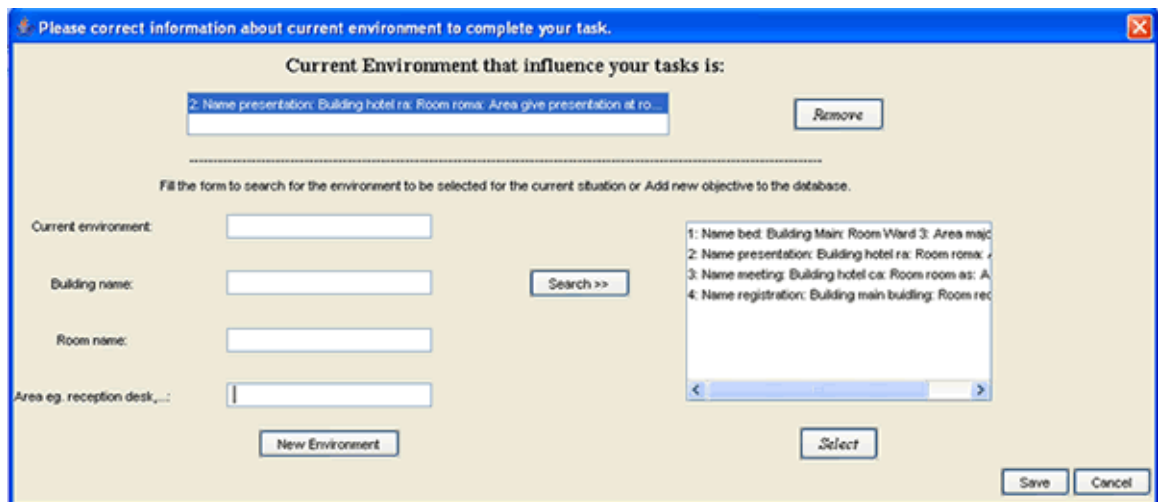

Figure 7-1 The GUI for Environment Element to Store in the Environment Object

As the profiles and databases are separate and have a consistent structure, a GUI is created for each object in each database so that it provides an ability to add or edit a set of values to the object database both during the design stage and in real time use of the context aware system. During design stage, a set of values in each context element that has been extracted from the user's requirement scenarios can be added by the developers through this GUI. At the same time, GUI demonstrates the possibility for the system to allow user to add a new set or edit existing set of values in the context element in the context model providing a good

presentation of context model is adopted in the system. Hence the user can add or edit datasets for each database or situation that relates context elements in the context model database (ATsXML). The GUI uses the methods in *readXML.java* and *writeXML.java* to update the values in the database where necessary. A GUI example of the environment object is shown in Figure 7-1. The new object is then added to the database XML file using the method in *writeXML.java*.

The design output from the previous chapter is used during implementation as a guide to assigning sensors for gathering information that will be used during real time. The design tool guides the designers in extracting and grouping the information required in the situation instead of using the availability of the sensors to limit the information that will be used in the system. For example, Table 6-31, Table 6-33, Table 6-35 and Table 6-37 demonstrate that the design tool guides the developers in implementing the sensor acquirers to gather information from sensors that will be used in the system. The next section describes how the sensors acquiring process is developed in the architecture.

7.1.2   Sensor Engine Layer

As described in Section 5.1.3, in order to support the separation between the raw sensor data from the application and the reasoning process, the sensor engine layer is divided into 3 elements – sensors, sensor translators and sensor engine. The implementation of these elements is discussed below:

7.1.2.1   Sensors

Each sensor requires different code to acquire data from the sensor. Bluetooth is used in the prototype as an example, even though in real life, Bluetooth might not be appropriate for many situations and other types of sensors would be used as appropriate. The code for acquiring raw data from the Bluetooth is implemented in Java. From now on the code for acquiring raw data from sensor is called

"acquiring code". The *javax.bluetooth* package is imported in order to take advantage of existing methods. The code is for the system to discover the Bluetooth devices that are in range. The data that is to be gathered is the ID of the device and the name of the device. Another acquiring code is to get the date and time. The code imports the *java.util.Date* to get the date object that hold information about date and time on the system. The code for gathering raw data from the Bluetooth and clock (*bluetooth.java* and *dateTime.java* respectively) are shown as ⬡ in Figure 7-11.

In this case, the user is required to register with the system in order to gather the information about the user in the user's profile. The GUI for log in and registration of a user is shown in Figure 7-2. Through this explicit input, the information from the registration form is stored in the user XML file.

For other information that cannot be gathered from the sensors, code for creating profiles is written. Following the design output from Section 6.2, the patient profile, which holds information about the patients in the scenario, is created. The patient profile stores the data about what information is available in the scenario (such as patient name, date of birth and address). The object code for each profile is created for dealing with acquiring data from the profile, i.e. *xmlPatient.java*. The object code is a class that deals with a object such as sensor and profile. The object code for each sensor and profile also provides a method for other code to access the data about the object.

The sensor acquirer is very common in the past frameworks mentioned in Section 2.2.3.3 as this is the main common aim of the frameworks to separate the sensor acquirer from the application. For example, the Sensory Capture and Context Provider in CORTEX and SOCAM project respectively. However, not many past projects prepare for the information that is not available by current technology.

The profile acquirers in our project are aiming at providing necessary information that will improve system efficiency in inferring about user's objective.

The information from the sensor or profile acquirer code is gathered and sent to the sensor or profile translator to be processed.

Figure 7-2 GUI for System's Log in and Registration

7.1.2.2 Sensor Translators

The raw data from the sensor is translated in order to get meaningful information from the raw data. As mentioned in Section 5.1.3, the first level of processing data is to reduce noise in the data. Then the second level of processing data is the interpretation. As we are not concentrating on accuracy of the data in this prototype, the first level of processing data, noise reduction, is not implemented. But it can be added to the sensor translator by, for example, instead of using the immediate set of detected Bluetooth devices, the sensor translator has ability to

monitor the detected devices over a period of time and the devices that were detected every time during the monitored period are sent to the next processing level. This process gets rid of devices that are not detected at all during the monitored period.   In   this   prototype,   the   Bluetooth   translator (*bluetoothTranslator.java)* gets the ID and name of the detected devices and processes them to get information such as Bluetooth ID, name of the device, owner's name and type of device, etc by referring to the MAC address and the registered device's profiles. The set of meaningful information for each Bluetooth device is stored in the database, following the design output from Chapter 6. The data from the system clock is also translated into a set of required values based on the design output, Time database as shown in Table 6-34 (i.e. day, date, month, year, hour and minute). The sets of values for these attributes in different situations are then stored in the database. The code that operates the translation of data is from now on called translation code. This is similar to the Context Interpreter, Interpreter in SOCAM, Context Toolkit respectively. Our Sensor and Profiles Translators have the attributes in the context element database as a guideline of what to translate the raw data into.

To avoid repetition in the database, the values are used by the translation code to ensure that the same set of values is not stored more than once in the database, by checking its ID. For example, if the Bluetooth ID is found in the database, the process of getting further information (such as Device type, Owner name and location) does not have to be performed. The set of values of the meaningful information from the existing database is used. The meaningful information in the database is accessible by using *readXML.java*. The set of values will then be used to build information for the context elements, as in the next section.

7.1.2.3　　Sensor Engine

In this prototype, the sensor engine (*sensorEngine.java*) contains code that manages the registry of the context providers – the startSensing() method. This method starts the sensor acquiring code. As a result, the sensor translator processes the data into meaningful data. The main duty of the sensor engine is to assign the meaningful data to the attributes in each context element. For example, in this case, the name of the device is assigned as name of tool in the tool context element, name of the Bluetooth owners were assigned as names of users in the situation, list of Bluetooth owners (users) detected is assigned as community, the user log in ID as for getting current user element etc. As a result, the information for user, tools, community, environment and time elements in the current context model is gathered from sensor data. These current elements are stored in the context element objects which are accessible by the context engine in order to reason about the current context model.

The sensor engine is similar to the Adaptor Layer in Hydrogen project mentioned in Section 2.2.3.3. In order to avoid multiple applications reading from the same sensor, it gathers meaningful information in the context element objects and sent them to another layer to manage the context. Hydrogen project does not have a consistent set of context element objects. It allows the application to query a specific context from the server. Our sensor engine gathers information from sensors into a consistent set of objects and sends them to the next layer to deal with inferring about user's objective rather than applications have access to inconsistent context information.

7.1.3　　Context Engine Layer

The context engine layer contains the context engine code (*contextEngine.java*). It takes available information about current context elements from the sensor layer (represented in the form of *user object, community object, tools object,*

258

environment object and time object) to infer the user's current role by comparing the ID of the current context elements from the sensor layer against the history of the context models in the database. The history of context models in the database is a set of ID values of the context elements in each slice of context model in the history (ATsXML). As a result of using our extension to Activity Theory in the context model, for every situation the context model which is represented as a set of IDs of context elements or *AT object* (ATXML) is stored in the history database (ATsXML). In this prototype, the code *findBestMatch.java* is developed for the inferring process. The matching algorithm that is adopted in this prototype is a simple process of matching the ID *String* for each element in the current context model with one in the existing context models in the history. The *equalsIgnoreCase()* function available in Java is adopted here but a more sophisticated algorithm can and should be used to improve the accuracy of the system in the *findBestMatch.java*. The current *role object*'s ID is extracted from the matched context model in the history. If all or almost all available current IDs are matched to the values in the context model in the history database, the *role object*'s ID is extracted from the matched context model in the history to assign values for the *role object* for the current situation. If there are no matches at all, a new role has to be created and stored in the database via the GUI, as shown in Figure 7-3. The GUI can be used by the user in real time to create a new role and assign the values for the context elements (user and community) in the role database. The required information on the GUI such as current user ID, community ID and role name is based on the design output as shown in Table 6-45. If the current user ID and the current community ID can be gathered from sensor or profile databases for the situation, they will be pre-filled and will leave the user just to assign the new role name.
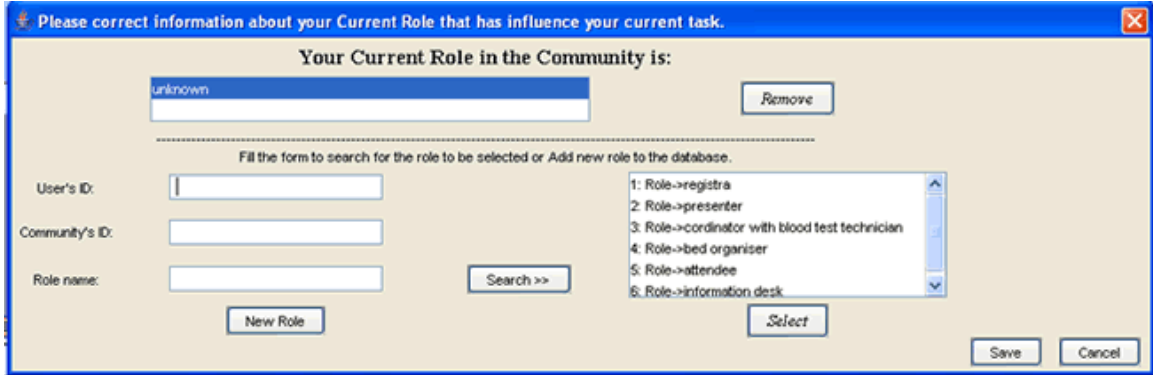
Figure 7-3 GUI for Gathering Information about Role Element

Based on the design output from Chapter 6, the rule database holds the information about the rules that each role is liable to. The *role object*'s ID is then used by the context engine to get the *rule object* from the rules database. Similarly, if the set of values of rule element is not found, the new rule can be created by the user in real time via the GUI. The GUI is implemented based on the attributes in the rule database as a result of the design output shown in Table 6-46.

As a result, the set of IDs of the current context model (*user object, tools object, community object, environment object, time object, role object* and *rule object*) is compared against the history of the context models to get the ID of the *objective object* using the same matching algorithm. If there is no best match found, a new objective is added to the objective database by the user via the GUI, in order to create the *objective object*. The ID of the *objective object* is then assigned for the object element in the current context model or *AT object* (ATXML).

The ID of the *objective object* is used to find the *outcome object* from the outcome database, again following on from the results of the design process in Chapter 6 (see Table 6-25 and Table 6-48). If the ID of the *outcome object* is not found in

260

the outcome database, a new set of data for the *outcome object* is created via the *outcome object* GUI. Once the new *outcome object* is created, its ID is assigned for the outcome element in the current context model or *AT object*. The current context model or AT object are then stored in the history of the context model database (ATsXML), bases on the design output shown in Table 6-49.

The context engine only supports the design output based on our design tool. Therefore different context models that have been created in different projects will need to be adapted. The consistency in the context model to be used by the applications allows the context engine to infer about user's objective uniformly. Unlike the past projects that have an application that subscribes to different context elements. For example Aggregator and ContextClient in Context Toolkit and Hydrogen that inconsistently subscribe and reason about context for different applications. For every new application, the Aggregator and Context Client are required to be developed in order to support the new application. This process is including making decision about context to be subscribed and how to reason about them which can be a time consuming process.

The code for the context engine not only provides the inference methods (in this prototype, a matching algorithm is used to match the IDs of the context elements) above but it also provides the code that allowed other code to access the current context model or *AT object* (ATXML) via the *getAT()* method. This method allowed other code to get further information about the context elements in the current context model through the set of IDs of the context elements in the current context model that the method provided. This was done by using the ID of the element in the current context model to refer to the set of values in the database of that element. The set of values from the context element database that has the same ID value is used to present the information about the context element.

This function will be useful for the application engine as shown in the next section.

7.1.4 Application Engine Layer

Users can have difficulty in understanding the complex reasoning methods behind a context-aware system. This lack of understanding can lead to breakdowns and frustration when the system makes mistakes. Humans often make mistakes in their interpretations of *other humans'* intentions but we are still able to achieve our objectives by recovering from such misunderstandings. But in many cases, users stop using a computer application because they do not understand what the application is doing, how it is trying to do it, or why it repeatedly comes to a wrong decision. Even though the application may allow the user to correct errors, a lack of understanding between the user and application in how the errors occurred can result in users quickly becoming disenchanted with the application. Improved communication between the user and the context-aware system is important, as this will increase the user's knowledge of how to control the application. For a successful context aware system, it is as important for the user to have an accurate model of the system's intentions as it is for the system to have an accurate model of the user's intentions.

Following on from the previous section, the application engine layer accessed the current context model or AT object through the *getAT()* method in the context engine. The ID of the outcome object is used to find the best matched *application object* in the application database which stores information such as application ID, Outcome ID, description of application and required context information as shown in the design output, Table 6-50. If the outcome ID is not found in the application database, the application object GUI is used by the user in real time to create a new set of values for the application database. The GUI is implemented based on the attributes in the application database. Once the user selects an application

from the list of available application that the GUI provides, the value is stored in the application database with the current outcome ID. The new *application object* for the outcome ID is then stored in the application database.

The application engine then uses the *application object* to process the information further. The process is divided into 2 main steps.

In the first step, the context information that the application requires from the current *AT object* can be identified from the discovered *application object* as shown in Table 6-50. The application engine transforms the IDs from the *AT object* into the information that the application requires. This is done by finding the context element object in each context element database that matches the same ID of that element in the current *AT object*. For example, the Tools element is required by the application ID 3 "Checking In assistant". The ID of the tool element in the *AT object* is used to get further information about the list of tools available in the current environment from the *tools object* in the tools database that has the same ID as the one from the current *AT object*. Each ID of each tool in the list of *tools object* is then used to get information for each tool available in the environment. The information of each tool such as Desktop1, Printer1 and Pink Form is then available for the application. This is done in the same manner by using the ID of each tool to refer to the set of values of the tool in the tool database, as shown in Table 6-39. The application engine gathers information requires by the application in the same manner following the values in the context attribute in Table 6-50. For example, in the check out patient situation, the context elements that are required by the application are Community, Tools and Time context elements. The ID values of these context elements are taken from the current context model in order to gather further information about each context element from their databases.

In the second step, following the design tool, for each value in the application attribute in the application database, the developers decide to implement a new application or use an existing application. The new applications will be implemented following the requirements developed during the design stage for each situation. Moreover, the developers may adapt the existing applications to meet the requirements.

In this case, the prototype applications of Checking In (*GUIbookInDialog.java*) and Checking Out (*GUIcheckOutDialog.java*) are implemented in Java. In addition, the applications are registered to the context-aware system. This is done by assigning the applications to the outcome in the application database for the relevant situations, as shown in Table 6-50. The application implementation follows the user requirements and the levels of activity produced during the design stage.

The flowchart in Figure 7-4 shows how the system supports users in real time. When the current context model is found in the history database, the system provides the application with information relevant to the user or just provides context information to the user. In this case, according to the information required by Sara recorded in the user requirements, the GUI interfaces for Check In patient and Check Out patient are shown in Figure 7-5 and Figure 7-6 respectively. As a result, instead of the traditional Check In and Check Out forms that require explicit input from Sara, the assistants add methods to automatically open the Check In or Check Out form and fill it with the patient information where the information can be gathered from the user in the community context element in the current context model.

In this case, the community element in the current context model or AT object is used to get the information about the community in the current situation. The ID of the community element is referred to the database to determine who is in the

current situation based on the set of values that has the same ID in the database. The application engine extracts the list of users in the community in order to get users' information. The information from the discovered user object is then transformed to the patient information as a *patient object*. The *patient object* is then passed to a method in the application so that the form can be filled with the available information to save Sara from explicitly typing in all the information about the patient. Sara only has to check if the information is correct and then submit the information to the system in order to confirm the check in and check out status of the patient.

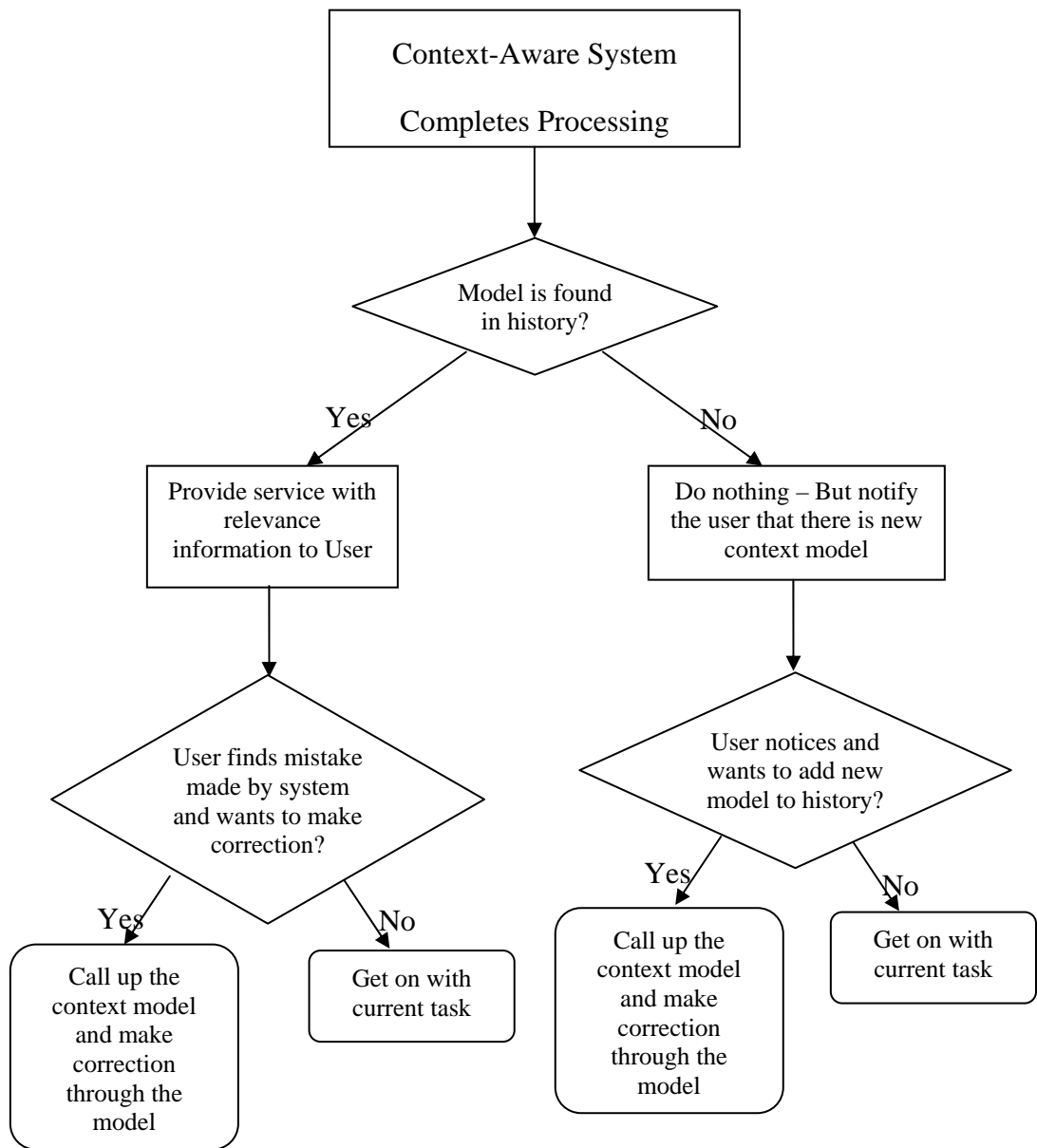**Figure 7-4 Flowchart of how the System Supports the User in the Prototype**

For this prototype, after the receptionist (Sara) logs in to the context-aware system, the GUI interface in Figure 7-7 shows the receptionist is logged in, with a statement reading "Hello Sara A" where Sara A is the user's name. The GUI also provides buttons for available applications so she is able to access the applications

explicitly when needed. This is done in order to maintain the user's sense of being in control (see Section 2.2.4). In addition, the context model button is presented so that the user has an option to see the current context model which represents the underlying reasoning model used by the system. If the user finds the service or information that the system provides is inappropriate, the user can click on the context model button in order to see the current context model. The user can make changes to the context model when she thinks the values in the model are not appropriate. As the values are then stored in the databases, the changes will take effect on subsequent inference processes.

Figure 7-5 GUI for Check In Patient Application
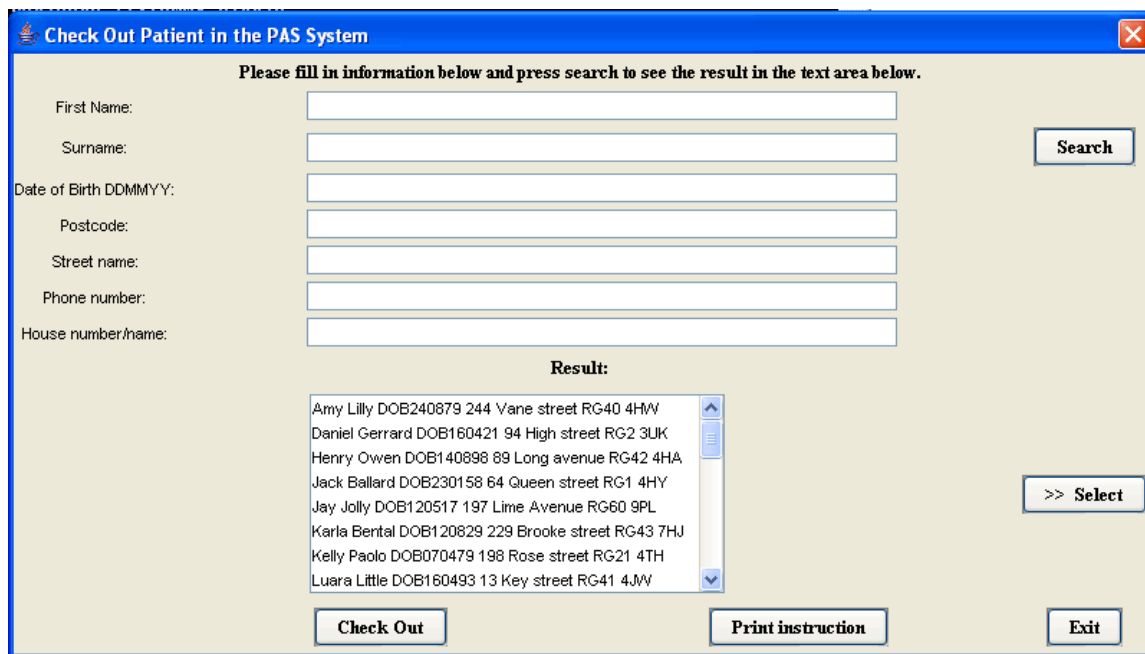
Figure 7-6 GUI for Check Out Patient Application
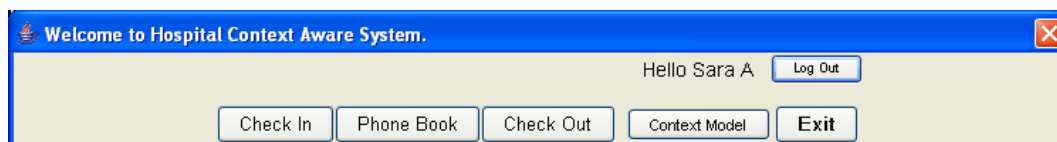
Figure 7-7 GUI of the Hospital Context-aware System

From the flowchart in Figure 7-4, when the current context model is not found in the history database, the system does not make any context-driven interventions but discreetly notifies the user that it found a new situation that might be of interest to the user. This gives the option for the user to add the new context model to the history database or ignore it and continue with her current task.

268

When the user decides to add the new context model, the context model button is clicked, the current context model is represented to the user via the GUI shown in Figure 7-8. The GUI is represented in the form of each slice of the extended Activity Theory model. The buttons are used to represent the values of elements in the context model of context. If the user thinks the value that is shown on a given button is inappropriate or missing, the user can press the button to open the GUI for that context element in order to change or add the values of the attributes in that context element. The GUI for context elements are the same as the ones described in Section 7.1.1 including code such as GUIenvXML.java, GUIroleXML.java, GUIuserXML.java and GUIruleXML.java where the implementation is based on the design result tables. Once the values are updated, the database for the context element is also updated. Once the user completes editing the values in the context model, the user selects the save button to update the context model in the history of context model. The changes will therefore influence subsequent inference processes[1].
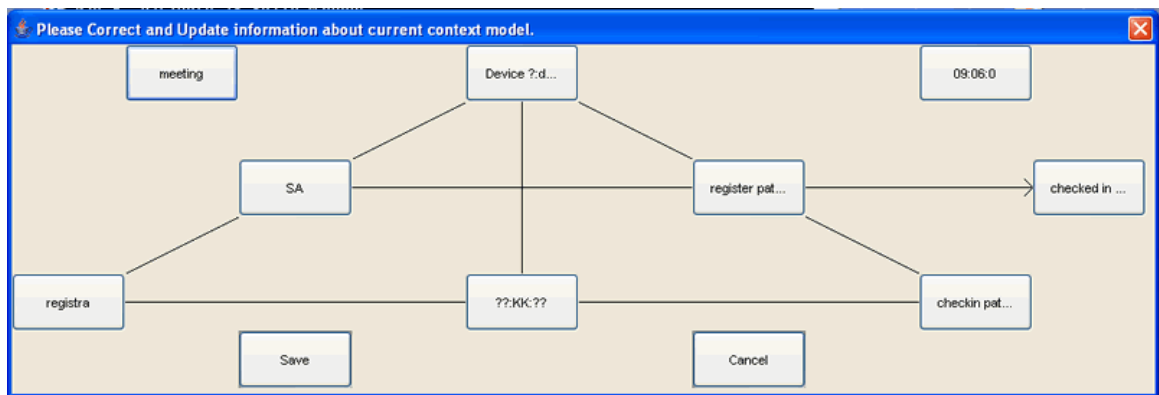
Figure 7-8 GUI Shows Current Context Model - allows users to update the model if required

---

[1] In a somewhat similar approach, in November 2008 Google.com introduced "Promote" and "Remove" functions for the user to edit search results in a way that influences subsequent search processes.

Moreover, as noted in Section 5.1.5, the application engine requires a command to execute the chosen application through a suitable device and pass the selected parameters accordingly. The device information from the user element (personal device) or tool element (tools availability) will allow the application to decide the format of the information e.g. for a PDA (personal small screen and less powerful) or a projector (more public large screen). In this situation, the Desktop is chosen as a device to show the support to the user as the receptionist (Sara) sits at the registration desk where the desktop computer is available to her.

The application engine acts like an interface for the application to access the context information. Unlike Aggregator, ContextClient and Context service in Context Toolkit, Hydrogen and Gaia project respectively, application is not subscribed to certain context information. It has access to the uniform context model. The application engine refer to the current context model and application database to decide what application or what context information to be presented to the user. The application itself does not directly deal with the context reasoning. It only has access to the databases in order to get information about required context elements.

7.1.5 Conclusion

XML and the Java language were used to implement the hospital prototype in order to demonstrate the implementation of the architecture that supports the functionalities introduced by our design tool. As a result of the implementation of the context-aware system based on the design output, the architecture is shown in Figure 7-9. The architecture contains three layers: sensor engine layer, context engine layer and application engine layer. These layers have access to the databases as described in Chapter 5.

The implementation of the prototype described in this chapter has illustrated how the architecture provides the separation of context elements and its reasoning process. The architecture introduces advantages such as efficiencies in reusing code and existing context data in the databases. The process of gathering the data from the situations during the design stage in order to be stored in the database to be used to infer about user's current objective is a time consuming process. Once the current context model is found in the history database, the current context model, which hold a consistent set of IDs of context elements, is passed to the application engine layer. Then the application engine, which has information about the available applications, first uses the ID of the outcome element in the current context model to access application database to get information about the require application and context elements. Then the application engine translates information of the required context elements and passes it to the selected application according to the application database. The application is then activated by the application engine. The application engine acts as a translator for the application. The application can be redesigned or changed to different application for the situation as long as the developers notify the application engine and update the application database.

Moreover, the process is an ongoing process as the user should be able to add and edit the set of values in real time. Therefore it is important that the existing context data can efficiently be reused and edited through the consistent context elements that are separate from each other and the context model. The possibility of manual in-use-adjustment (adaptability) by an end user is demonstrated through the simple context model GUI shown in Figure 7-8.

In order to evaluate the architecture, the next sections will discuss how the implementation can be adapted and extended to support other scenarios discussed

in Chapter 6. How well the implemented architecture meets the requirements described in Section 2.4.2 is then discussed.
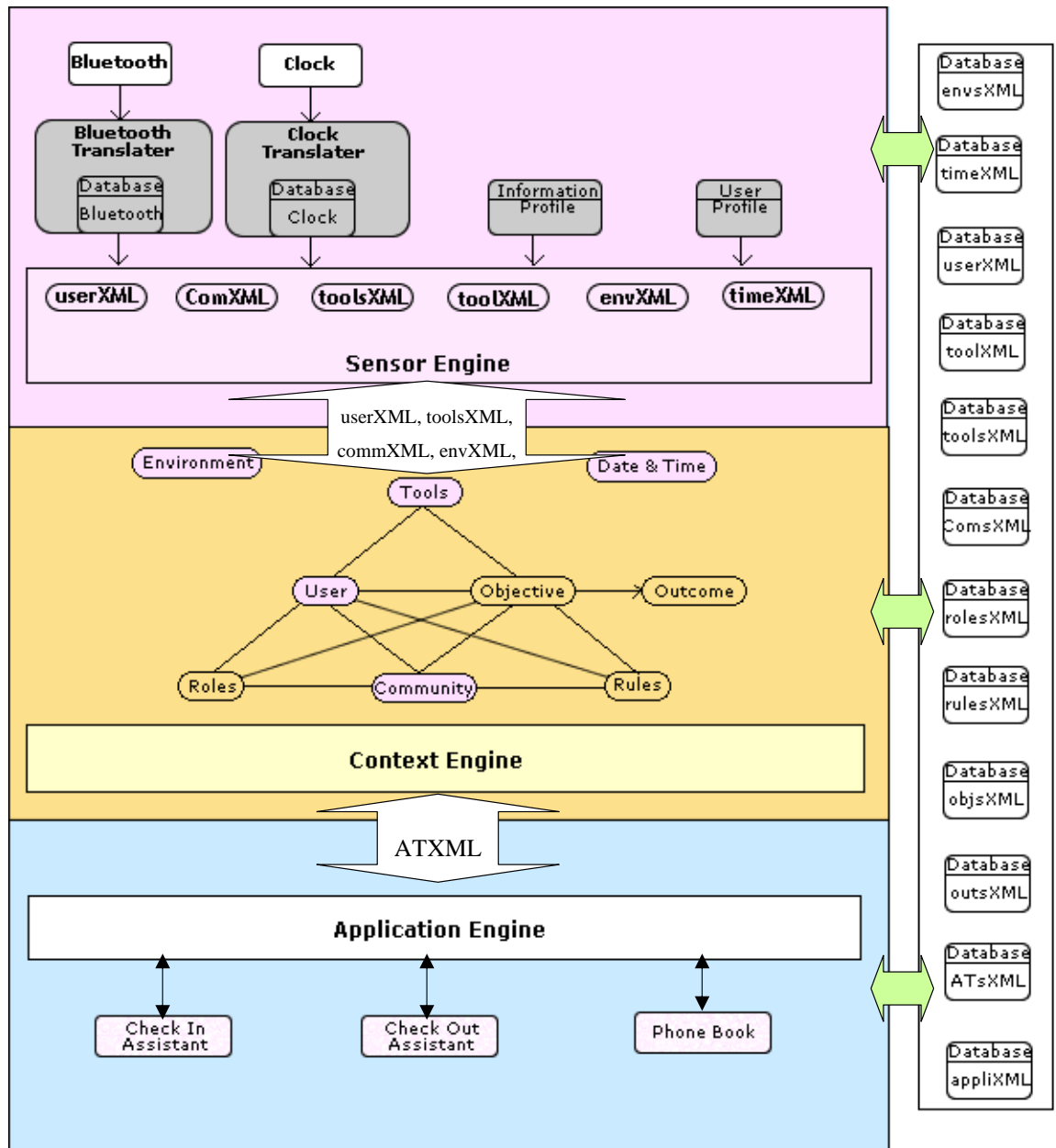
Figure 7-9 Architecture of Context-aware System Based on Results from the Design Tool

7.2   *Applying the Prototype to the Scenarios Design*

The architecture implemented in the previous section shows that implementation of the architecture presented in Chapter 5, which is introduced by the design tool as a result of adapting Activity Theory in the context model, is feasible.   The architecture supports the database structure as a result of the design process.   The next section will show how the simple and more complex applications mentioned in Chapter 6 take advantage of this architecture in a ubiquitous computing environment.   This section is intended to demonstrate how the existing functionalities in the prototype can be applied to the simple and more complex applications in order to evaluate the system architecture.

7.2.1   Simple Tourist Guide and Conference Applications

The "tourist guide and conference assistant" scenario in Chapter 6 is used to discuss the functionalities of the existing architecture implementation.   This section demonstrates the usability of the existing architecture in a different domain, i.e. moving from the hospital domain to the conference domain. The next section describes the potential for reusing or extending the context-aware system from the existing architecture within the same domain.

Database

The Bluetooth, clock, context element databases and user profile can be reused in the tourist guide and conference assistant application.   This is because the same level of data is required for each database. Even if the database requires extra attributes in the database, it can easily be extended by adding XML code in the database to represent each new attribute required in each context element via simple extra lines of code in readXML.java and writeXML.java. The code is added to allow the java class to be able to access (read/white) the new attributes in the database. The existing sets of values will then be updated with the value, such as

"*unknown*", for new attribute unless the value is known. , . At the same time, the Java code for that object is updated in order to add the new variable (i.e. add new attribute value to the context element) to the object. Additional profiles required in this scenario are map, folder and timetable profiles in order to hold information about maps, folders and timetables available in the scenario. The XML (such as maps.xml, folders.xml and timetables.xml) and Java (such as *xmlMaps.java, xmlFolders.java and xmlTimetables.java*) code for these objects are created in order to allow other code to access a set of values for each object. Similarly, the variables in the object are chosen according to the design output tables (Table 6-16, Table 6-17 and Table 6-18).

Sensor Engine Layer

From the existing databases developed in the previous section, the existing architecture has a well structured concept of what sensors and profiles are to be used in the system based on the design outputs. The sensor engine does not rely on any sensors or profiles in particular as long as the information about context elements available from any available sensors and profiles are received by sensor engine in this layer. Sensor engine then passes the current information about context elements to the context engine layer to get full current context model for the application engine layer. The sensor engine layer does not directly interact with application engine layer. The application does not know what sensors are available to them and does not need to know. The databases are consistent and separated from each other. This section describes how this architecture enables the reuse and extension of the existing sensor engine layer for the conference assistant scenario.

Sensors

From applying the design tool to the conference assistant scenario, there are seven types of sensor, including the thermometer, Bluetooth,

274

GPS, system clock and WiFi, to be used in the system according to the design output in Section 6.1. In order to be able to acquire data from new sensors that the existing implementation does not support, such as thermometer, GPS and WiFi, the implementation of the code to request raw data from each sensor has to be done separately in the appropriate languages. The codes for the Bluetooth and Clock can easily be reused. The codes for other sensors must be implemented with the functions that allow any subscribed object to access the data similarly to the existing method get() in the Bluetooth and Clock code.

For the data that cannot be sensed from sensors, new profiles such as maps, timetables and folders profiles are created in XML in order to store the values of the attributes in the profile following the design output (see Table 6-16, Table 6-17 and Table 6-18). The database structure of the context elements and context model in the existing architecture can be reused as the information from sensors and profiles are grouped in the same manner as in the prototype as a result of following the context model and design tool.

Sensor Translators

The existing translators for clock and Bluetooth can be reused. Other translators will have to be implemented. As mentioned previously, for every new raw data that is received from a *sensor*, the data is processed in order to get meaningful information. The first level of processing data can often be to reduce noise in the data. For example, instead of using one piece of raw data from the GPS, the developers can design the code to collect a set of raw GPS data over some period, say one minute, and use the average value. The second level of processing data is the interpretation. The code is implemented to

process the data for different sensors according to the design output tables. For example, the processed GPS value is used to get information about the building, area, town and country following the design output table. As in the previous section, for other objects to access the data, a method to provide communication between the sensor or profile translator and other objects is implemented. In this case, it is getGPS(), used to send the data about the current values of the sensor to the subscribed object, where GPS is the name of the sensor or profile.

Sensor Engine

In the sensor engine, the developers implement the code to assign the values from the attributes from the new sensors and profiles to the attributes in the different context elements, according to the design outputs. New code to add new information from the new sensor is required to assign the value to the variables in different context elements. Following the design outputs, the values for the *environment object* in Figure 7-10 are combined with the values from the assigned sensors (Thermometer, Bluetooth and GPS) in Table 6-5. For example, the processed GPS value is used to get information to assign to the building, area, town and country attributes in the environment database.

The method of deciding whether the information already exists in the database can be reused. By using the best match algorithm, a set of values is then added or updated in the environment database in order to get the ID of the *environment object*.

The code for translating information from the system clock can be reused. In order to get value for the new attribute of Period of Day in the time database, the code for transforming the time to the Period of Day is added as suggested during the design step.

Identical to the existing implementation, for every unique tool detected by the WiFi, Bluetooth and NRFID as discussed in Table 6-13, the *tool object* is assigned the values for the attributes in the *tool object* for each detected device. The discovered *tool objects* are then added or updated in the tool database in order to get the ID of each *tool object*. The additional code is implemented for this scenario in order for the sensor engine to be able to detect the availability of the map, timetable and folder from the profiles. The discovered object is then added to the *tools object*. In this case, following the design output, the value of the environment element in the map profile is used to get the relevant *map object* for the situation in order to get the ID for the *map object*.

The existing code can be reused to create the *user objects* for every unique user detected by every detected tool such as WiFi, Bluetooth and NRFID. Similarly to the tool, the user objects are then added or updated in the user database in order to get the ID of each *user object*. The code for extracting the user object is therefore extended by adding the information from other sensors aside from the Bluetooth in the implemented architecture.

Similarly, the existing code combines the unique tools into the information for the tool list, map list and name list attribute in the *tools object* that will be stored in the tools database. Similarly for the *community object*, the detected *user objects* are combined to obtain information for the attributes in the community database. As a result of using the IDs to refer to object, the existing code for detecting unique IDs from the *tool* and *user objects* can be reused. Also the code for getting the ID for

both the *tools object* and *community object* from the tools database and community database respectively can also be reused.

Other codes can then access the context element objects via the existing getUser() method – where the User is the name of the element in the context model where the sensor engine can gather its information.
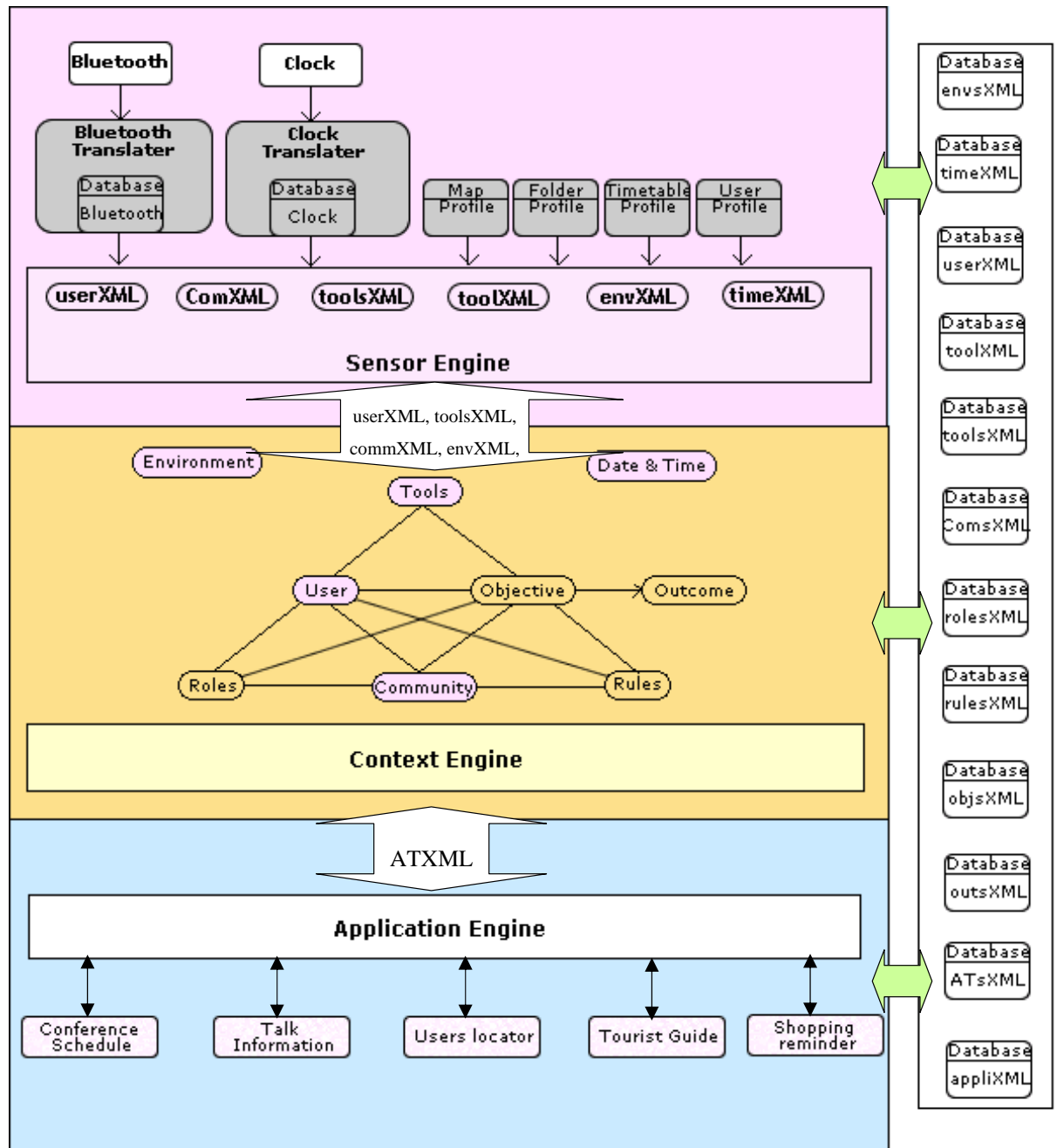
Figure 7-10 Diagram of the Architecture Supporting Scenario 1

Context Engine Layer

The reasoning process for finding the missing elements in the current context model is done in a consistent manner by starting to find the role, rule, object and outcome elements respectively. There is therefore no change required in the code for the reasoning process about the context model. As a result, the existing *contextEngine.java* can be reused in this scenario, and others.

Application Engine Layer:

The application engine used only the ID of the outcome element rather than directly embedding the information about the current context model and reasoning process in the application engine. As only the ID of the outcome in the *AT object* is used, rather than "proprietary" information and reasoning process, the same code can be reused to find the best match in the application database for the new scenario. The existing code in the application engine for accessing the *AT object* to gather the information required by the application can also be reused. This is done, as mentioned previously, by using the getUserAt() method from readXML.java. It passes the ID of the set of values in the context model; in this case it obtains the *user object* of that reference user ID in the *AT object*. The application engine gathers information required by the application in the same manner following the context attribute from Table 6-27. Then the application engine either provides the service to the user or selects information to be provided to the user in this application engine layer.

Following the design outputs, new and existing applications can be used to support the user in this scenario. For example, by using an existing application such as Google Maps, the web browser opens the Google map with the user's current location on the left of the screen, this covers 50% of the screen and on the right of the web browser is some small detail showing tourist information with a reference point on the map. After the application engine decides what context-aware

support to offer the user, it shows the application, for example a tourist attraction where the map is opened using the web browser.

Thus, the implementation transfers well from supporting the features of the hospital scenario for which it was originally created to supporting the relatively simple conference assistant and tourist guide scenario. The existing code can be reused on many occasions such as in existing sensor translators, sensor engine, context model reasoning in context engine and accessing data from context model in application engine. As a result of the designed architecture's consistent and well separated structure of the context elements and context model. In the next section we look at extending the implementation to support further features within the more complex hospital scenario.

7.2.2 Complex Hospital Scenario

The previous section suggested that on many occasions the existing code of different methods in the implemented three layered architecture could be reused for the conference assistant scenario. This is because of the consistent structure of the databases and the systematic separation of dealing with different levels of information in each layer in the architecture. Moreover, the predefined values in the context elements and context model databases in the previous scenario can also be reused. This section discusses the use of the implemented architecture in the same domain for which the architecture was implemented in the first place but to support different situations such as a Phone Book assistant. As the architecture was implemented according to the design output for the complex hospital scenario, the existing databases can be reused.

Sensor Engine Layer

Sensors

According to the design output for the complex hospital scenario, the chosen sensors include Bluetooth, GPS, clock and RFID. This is similar to the previous scenario so the code for acquiring raw data for Bluetooth, GPS and clock can be reused. In addition, code for acquiring data from the RFID is required. The RFID is used to sense information about the community because the hospital can be a busy space. A near range sensor such as RFID or finger print scan is required in order to know the right community that has an influence on the user (Sara) in different situations. A busy place like the hospital fills up with people who have different reasons for being there (i.e. patient, visitor, relative, etc). In order for the system to provide appropriate support for the receptionist, more precise information about who she is currently dealing with is required. This may be achieved with data from a short range sensor. The implementation of the code for acquiring the data from the RFID is therefore required including the get() method to provide the data accessibility to the subscribed object.

A new profile is required for the tools element in order to capture information about the information available, as shown in Figure 7-11. As mentioned previously, at the hospital access to information or tools is categorised into many levels according to the responsibility of the users ranging from the patient, receptionist, nurse to the doctor. As it is a complex environment, sensitive information such as the patient's database and x-ray results are not available to everyone in the environment. The sensitive information limits access to certain groups of people or just one particular person. For example, when the value of the owner attribute is Private, it means only certain people are allowed to access the information; e.g. the Pink Form that was created by the paramedic can be accessed by the receptionist only when the paramedic arrives at the reception of A&E. The attribute Owner is therefore required to be added to the Map profile and

Information profile. By adding the attribute to the profiles, a few lines of code are to be added to the readXML.java, writeXML.java and the code for *profile objects* in order to represent the value of the new attribute in the database and *object*. These codes can be used to update the database of the object to add the values for the new attribute to the existing object in the database.

Sensor Translators

The translators for Bluetooth, GPS and clock can be reused. A new translator for the RFID is required in order to transform the RFID data to the information for the context elements. The raw data from the RFID can be taken as a reference ID to find the information from the registered data in the RFID database. For example, the RFID database stores the ID and name of the registered patients and paramedics. The information is then assigned for the attributes in the *user objects*. Similar to the previous scenario, if the values are new, they are then stored as a set of values for the new RFID in the RFID database that is created and stored in XML.

Sensor Engine

The sensor engine does not require large changes, apart from allowing the RFID translator and information profile to be detected. New code for assigning the values from the attributes in the *RFID object* to the attributes in the *user object* and (if required) *tool object* is implemented. The codes for assigning the values from Bluetooth and clock can be reused.
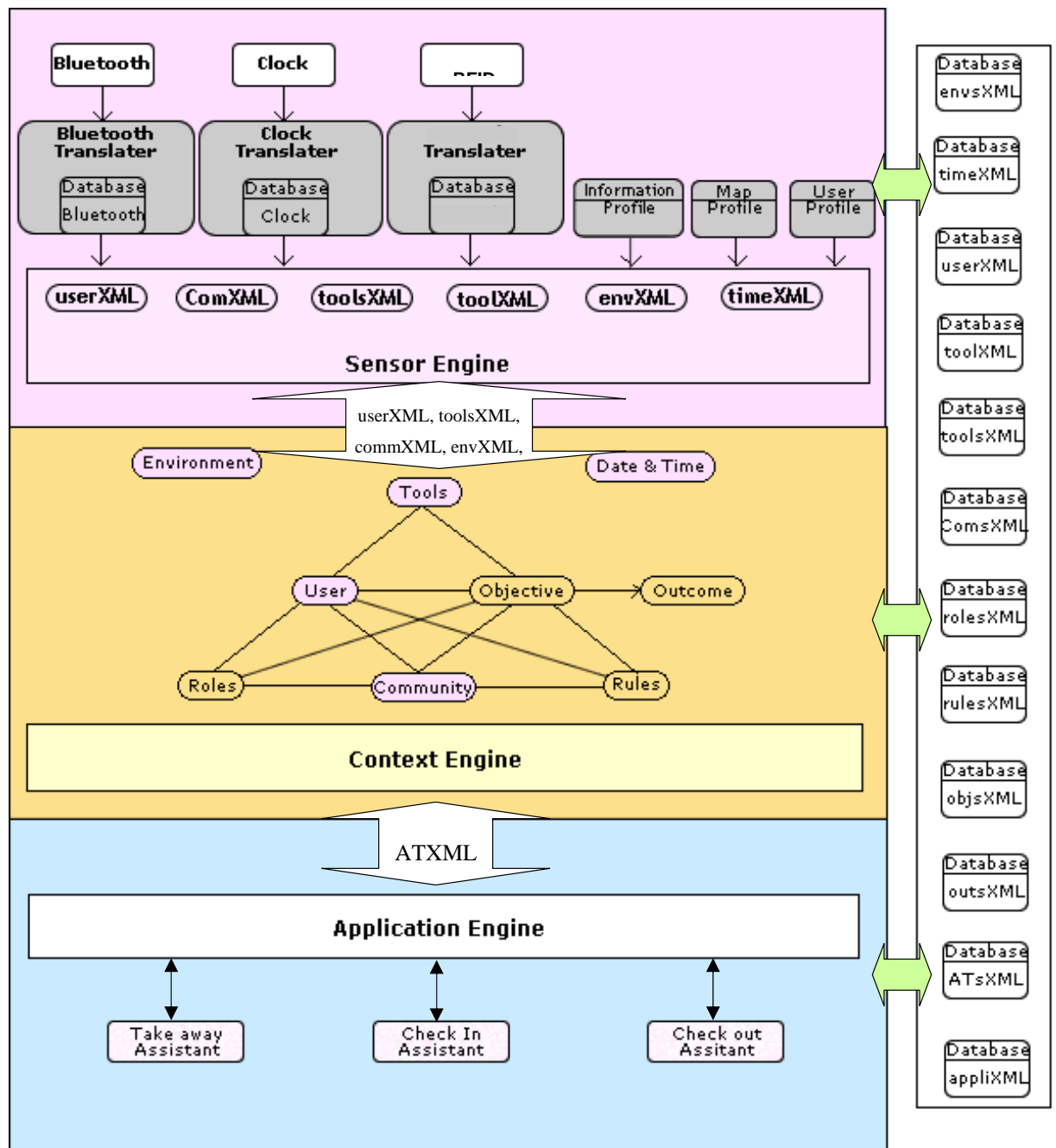
Figure 7-11 Overview Architecture Supporting the Hospital Scenario

284

Context Engine Layer

With the consistent reasoning process of using the IDs of available context elements to infer other elements in the current context model, there is no change required in the code of the context engine. This is because the context engine only deals with the values of the set of IDs and processes in the same manner to get the values of the missing elements in the current context model. The reasoning process is not directly embedded within an inconsistent set of information. There are always 9 uniform elements in the context model where the context reasoning process is occurred in order to infer user's objective. The additional attributes in the database do not affect the reasoning process code even though the new values will be taken into account. Therefore contextEngine.java can be reused.

Application Engine Layer

Following the design output, the application engine takes the ID of the current outcome element to infer about services to offer support to the user. The first function of transforming the *AT object* to the information that the application requires is the same as before.

The new situations to be supported in the hospital scenario require new applications. The application requirements extracted for each situation during the design stage are followed in order to implement the application. For example, the Phone Book assistant can be extended from the previous check in patient in our prototype PAS system by adding a method to pass the values to automatically fill the form where possible. In this case, the tools element in the context model can be used to get the name on the receipt. The name is then used to narrow down the list shown in the phone book. Then the list can be shown on the GUI screen for the user to check the phone number before she sends the message.

This chapter so far has demonstrated the implementation of the Activity Theory based architecture for context-aware systems and how it may be extended to support new situations and scenarios. The next section discusses the architecture functionalities with reference to the requirements of the context-aware system architecture presented in Section 2.4.2.

7.2.3   How Each Requirement is Met or Not Met in the Scenarios

The requirements from Chapter 2 are discussed. Each individual requirement is referred to in order to show how the architecture developed to support the use of the context model is met or not met in:

- Supporting the separation of concerns

First of all, in the sensor engine, each sensor has its own code for acquiring raw data. It is separated from the code for translating the raw data into a meaningful set of information – *sensor translator*. Therefore when the new sensor such as a RFID reader was added to the system for scenario 2, the code for acquiring the data from the RFID and the RFID translator was added to the system without having to change the other translation code which could be time consuming. Moreover, the existing code for acquiring data from clock, GPS and Bluetooth and the code for translating the data can easily be reused without rewriting the code – this provides a plug and play ability.

Furthermore, the architecture supports the separation between the sensor database and the context elements. As shown previously, each sensor has its own database that holds information that can be translated from the sensor. Therefore the information from the sensors or profiles can be reused in different scenarios or domains without the process of remodelling or gathering. For example, the map profile in Scenario 1 can easily be reused in a new scenario involving Sara, our hospital receptionist, on a 3 day trip to Hamburg.

286

Moreover, the clear separation between context elements in the context model allows the developers efficiently to reuse the context elements in different scenarios or domains. For example, information about the user (Henry) can be reused in Scenario 1 to represent Henry attending the conference instead of Adam. The history of context model in Scenario 1 can easily be reused for new user Henry by replacing Adam's ID with Henry's ID in the user element of the history of context model. As history of context model is used to infer user's current objective, by having the history of context model for Henry, it will save time for new user like Henry. It will be even more useful when the information about the element becomes more complex and time consuming to rewrite, such as the user database which can contain further details such as a current action and the user's stability (moving fast, slow, still), etc. In addition, when there is a need to edit or add a new attribute to the database, this can easily be done to the particular sensor database, profile and context element without affecting other databases, e.g. adding Owner attributes to the map profile in Scenario 2 from the map profile in Scenario 1 or adding current actions in the user element, etc. This makes it easier to reuse or extend information from the existing sensor database or context element database.

- ▪ Providing a consistent structure of the context interpretation

Following the structured reasoning method about the situation based on the context model and the ID referral approach, the code for reasoning about context elements shows a consistent structure of the interpretation about context elements and inference about the user's current objective. Therefore the code for interpretation about context elements in the context engine can easily be reused with the databases created according to the context model during the design stage. As shown before, the code in the *context engine* does not require any changes in order to be used with new scenarios or domains – e.g. the hospital domain in

Scenario 2. The architecture supports the consistent structure of the context interpretation, therefore the code in the context engine can easily be reused. Moreover, the database for the context model is separated from the database of context elements. Therefore the context model can easily be reused. For example the history of context model of Scenario 1 can be reused by adapting the information about the user (Adam) to refer to a different user such as Sara from Scenario 2 to represent Sara attending the conference. It allows easy access to edit, add or remove items to the database without affecting other information or without having the hassle of remodelling the context model.

- Providing a constant availability of context acquisition

As each sensor has its own acquiring code that has method get() that allows any subscribed object to get the raw data from the sensor, the architecture allows multiple objects to access the raw data from the sensors when they require. Moreover, similarly to the method get(), the method getXXX() in the sensor translator, sensor engine and context model allows any subscribe objects to access the set of data of the sensor database (bluetoothXML), sensor engine (current context information such as toolXML) and context element database (such as userXML, commXML, toolsXML, etc.) respectively.

- Providing a consistent structure for the context storage and history

By separating the databases for the sensor, context elements and context model, the architecture provides storage for each sensor, context element and context model. The history of the sets of values can be stored uniformly in the database following the attributes name. Moreover, the reasoning about the context is represented uniformly in the context model database (ATsXML) as a history of context about the situations. As the attributes in the context model database are consistent following the elements in the context model, the values in the attributes are the points of reference to the set of information about the context elements.

288

The history of the context model is a consistent set of values of context elements in the context model. As a result, together with the ID referral approach, the modelling of information about the context elements or sensors can easily be transformed without affecting the context model as long as it contains an ID attribute. For example, for each context element, the XML database holds different values in the attributes about the context object in the database.

The information can be transformed into OWL Web Ontology Language [2004], which provides additional vocabulary along with formal semantics, in order to add more relationships between attributes by following the attribute names in the database. The values for the attributes in the context model database can then refer to different points of values in the OWL ontology. By using the context model in defining the template in OWL, it will provide the separation between elements and context reasoning. The OWL templates will therefore provide a uniform storage for context that can be reused in different projects instead of them having their own templates. Similarly, the sensor data can be translated into information that is described in OWL. The values in the attributes of the context element can be referred to the point of information in OWL. The context model separates the high level interpretation (inferring the user's objective) from the lower level interpretation so that the OWL templates can easily be extended, reused and edited without affecting the other templates about the sensor, context elements and context model. Furthermore, as shown previously, a value in the attribute can be more meaningful, for example the number of people and time of day do not have to be an exact number or time that is represented as a value of the attribute. They can be referred to as "there are more than 20 people in the community and it is lunch time" instead of "24 people in the community" and "12.30pm".

- Providing a mechanism of resource discovery

In this architecture, the sensor engine layer contains code, startSensing(), that detects all available sensors and profiles in the situation. It refers to a simple list of context providers and subscribes to the ones available in the situation. The code of startSensing() allows the system to gather information from different sensors. Each sensor translator individually deals with translating the sensor data into information of each value in the attribute in the context element. Therefore when the information cannot be gathered from one sensor, other sensors can be used. For example, WiFi and Bluetooth can be used to get information about Printer 2 as shown in Table 6-14.

- Providing a mechanism of security and privacy.

The architecture here does not provide real mechanisms for security and privacy as they are beyond the scope of this dissertation. However, there is a possibility of using the rule element to add a security mechanism to the architecture. The rule can be used to limit the access of the user to certain information or tools – like a policy driven rule [Keeney and Cahill, 2003]. For example, only the user who has a role as a nurse is able to access the PAS system. This provides security to the PAS and the information that cannot be seen or accessed by others. Moreover, by separating the information about a user in a user profile (Table 6-11, Figure 7-12 and Figure 7-13), it can limit the access by, for example, saving the user profile in the user's personal device or allowing only the user who has the same ID as the one on the user profile to access it.

7.3   *Context Frameworks Comparison*

By using two scenarios to evaluate the architecture proposed by this research, the previous section illustrates that the architecture meets the majority of the requirements. Other requirements noted in chapter 2, such as security and privacy, are beyond the scope of this dissertation. The architecture introduces the separation between sensors, context, context reasoning and application. Furthermore, the unique identity concept allows easy access, update and reuse of values in the databases. With the well separated databases structure introduced by the context model and the three layered architecture, the set of values in the database can easily be edited without changes in other databases that refer to the set of values. This is a huge advantage in cases such as new sensors, new types of sensor, new applications or a new domain as the process of gathering context information and predefining the context models can be a time consuming process.

Table 7-1 compares the capabilities of architectures based on different frameworks. The chosen conditions to be used in comparing the frameworks are based on the normal situations in context aware systems where there are often changes in sensors, types of sensor, applications or domains.

Chapter 2 discussed different context aware frameworks including their ability to cope with new sensors, new types of sensor and changes in rules of context reasoning. Chapter 2 mentioned that previous frameworks cope well when a new version of the same type of sensor is introduced to the system. However, the previous frameworks require changes in the predefined context models in the database whereas in this work it only affects lower levels of context in sensor engine layer not the context model itself. So the context engine layer and application engine layer are dealing with consistency of the context model. Moreover, there are requirements in rewriting or recreating some part of the architecture such as the aggregator and application in the context Toolkit project.

This can be a time consuming and complicated process. Also, when there are new applications or domains, the predefined context models in the database for the system to be recognised are normally required to be changed or added. This means there will be changes in rules within the context information or adding a new rule for reasoning about context. But in this work the rules for reasoning about context are only changed in lower levels of the context. The reasoning process of context model is consistent according to the relationships between context elements.

To create a set of predefined context models for the system to recognise the situations to support the user, the designers have to spend a lot of time extracting the models from the user requirements, and may not always get it right. Ideally, the user should be able to edit and add a set of predefined context models in real time while using the system. This process could take a long time. For example, in the smart home, the user stores different settings such as curtains, heating and lighting, for the context-aware system to recognise in different situations at different time of the year. This means it could take the user a year before she completes defining context models for the system. Therefore it will be difficult if the user has to turn to developers every time there are new rules, applications or domains. In this work, we hope the consistency in the context model will help user build mental model about the system more easily. Through this understanding and its uniform separation of concern capability, the user should be able to extend the context values in the context model through a sufficient interaction provided by the system. For example, through the GUIs that show the current context model and information about each context element. This requires intensive studies about such as representation of context for user and levels of willingness from user to make correction etc.

The context model and architecture proposed in this research introduce a clear separation between sensor, context and its reasoning. The system then requires minimum changes in the architecture. Therefore it is easier for the user herself to deal with the changes in rules, applications or domain. This shows that the architecture that supports the results from the design tool meets the requirements quite well. Moreover, the architecture provides reusability in context history in all four conditions that are used to evaluate the architecture in Table 7-1 (including changes in sensor, add new sensor, changes in rules and add new set of rules). Our architecture is referred as AT context framework in Table 7-1. Moreover, the changes required to the system are minimal compared to the other frameworks. However, the process of gathering information of context models from user's requirement and designing the profiles and databases can be complicated and time consuming. This will be worthwhile in a long run because of being able to easily expand and reuse these context models, profiles and database in the future. The consistency of the context model introduced by Activity Theory helps the developers and users in building a mental model of the system. The next chapter summarises the research reported here and discusses future research that may be conducted to further the field of context awareness based on this dissertation work.

Conditions	Changes in sensor			Add new type sensor			Changes in rules			Add new set of rule		
Context Frameworks	*Context rep/ User context conception*	*Context process/ Designer or developercontext conception*	*Reuse history*	*Context rep/ User context conception*	*Context process/ Designer or developercontext conception*	*Reuse history*	*Context rep/ User context conception*	*Context process/ Designer or developercontext conception*	*Reuse history*	*Context rep/ User context conception*	*Context process/ Designer or developercontext conception*	*Reuse history*
Gaia 2002	No changes	change in cotext provider	yes	change in context model	change in context model and hierachy	no	change in context model	change in context model and hierachy	no	change in context model	change in context model and hierachy	no
CASS 2004	No changes	change in sensor node	yes	change in knowledge base in ruleEngine	change in knowledge base in ruleEngine and in each client devices	no	change in knowledge base in ruleEngine	change in knowledge base in ruleEngine and in each client devices	no	change in knowledge base in ruleEngine	change in knowledge base in ruleEngine and in each client devices	no
Context Toolkit 2000	No changes	change in sensor widget	yes	change in condition model	change in condition model and intepreter & aggregator for relevant applications	no	change in condition model	change in condition model and intepreter & aggregator for relevant applications	no	change in condition model	change in condition model and intepreter & aggregator for relevant applications	no
Hydrogen 2003	No changes	change in adapter	n.a.	change in contextObjects	change in contextObject in contextClient for relevant applications	n.a.	change in contextObjects	change in contextObject in contextClient for relevant applications	n.a.	change in contextObjects	change in contextObject in contextClient for relevant applications	n.a.
CORTEX	No changes	change in sensor component	yes	change in context hierachy	change in context hierachy and production rules	no	change in context hierachy	change in context hierachy and production rules	no	change in context hierachy	change in context hierachy and production rules	no
Context Managing Framework	No changes	change in sensor resource	n.a.	change in context vocabulary	change in context vocabulary and labelling process	n.a.	change in context vocabulary	change in context vocabulary and labelling process	n.a.	change in context vocabulary	change in context vocabulary and labelling process	n.a.
SOCAM	No changes	change in sensor provider	yes	change in context model	change in knowledge base and logic reasoning rules in interpretater	no	change in context model	change in knowledge base and logic reasoning rules in interpretater	no	change in context model	change in knowledge base and logic reasoning rules in interpretater	no
CoBrA	No changes	change in sensor module	yes	change in context model	change in knowledge base and logic reasoning rules in context reasoning engine	no	change in context model	change in knowledge base and logic reasoning rules in context reasoning engine	no	change in context model	change in knowledge base and logic reasoning rules in context reasoning engine	no
STU21	No changes	change in sensor agent	yes	change in context model	change in knowledge base and logic reasoning rules in context reasoning engine	no	No changes	change in knowledge base	no	change in context model	change in knowledge base	no
AT Context Framework	No changes	change in OldsensorEngine	yes	No changes	change in Sensor Engine	yes for AT context history & unaffected history	No changes	change in sensorEngine or Sensor Engine (Depend on the type of rule)	yes for AT context history & unaffected history	No changes	change in sensorEngine or Sensor Engine (Depend on the type of rule)	yes for AT context history & unaffected history

Table 7-1 Frameworks Comparison

Chapter 8

Conclusion and Future Work

This chapter concludes the dissertation. It briefly reviews the three main aims of the dissertation which include offering a new context model, design tool and architecture for context-aware system design. Activity Theory was applied and extended in developing our proposed context model because of a number of useful features provided by its standard modelling. We developed a new design tool based on the combination of our proposed context model and concepts introduced by Activity Theory, such as its three levels of activity concept, in order to steer context-aware system design away from a technology-driven approach and towards a more generalisable yet concrete approach to context aware system design. To support the functionalities that the context model and design tool introduced to context-aware systems, we proposed a three layered implementation architecture. Towards the end of this chapter, we suggest future research that might further inform the field of context awareness.

8.1 *Dissertation Summary*

8.1.1 Supporting Research and Practice in Context Awareness

This dissertation has discussed previous context definitions and classifications (Chapter 2). It has shown that researchers in the field have different views about context, what elements it should include and the relationships amongst those elements. Moreover, from a review of previous context-aware projects, it has been argued that developers often design and implement context aware systems with greater reference to the features of specific technologies that are available to them, rather than to user requirements. Users themselves often reject attempts at context aware applications when they struggle to make sense of the system's model of user context, especially when the system gets it wrong. In order to support research and practice in developing and using context-aware systems, we need to support shared understandings and provide tools that allow designers, implementers and users to understand, communicate about and represent context effectively, efficiently and easily.

8.1.2 Aims

From the issues in context awareness and the requirements for a context model and context-aware system architecture discussed in Chapter 2, we arrived at three main aims that had to be achieved in order to develop better context-aware systems. The aims addressed in this dissertation were:

- Providing a context model that is consistent and simple;

- Providing a new design tool to support context aware system designers in focusing on the user's requirements;

- Providing an architecture that complements the new design tool and supports the implementation of consistent, reusable context-aware system components.

296

Example scenarios from different domains were used to investigate how the new design tool and architecture meet their requirements. In the next section, we review our contribution and the success of our approach in the areas mentioned above. Then potential future improvements are discussed.

8.1.3 Context Model

In Chapter 3 we proposed Activity Theory for use in context modelling. Activity Theory may be considered as a descriptive conceptual framework rather than strictly a theory, proposing a simple triangular structure of human activity that relates the seven elements (subject, mediating tools, community, rules, division of labour, object and outcome) which it claims have an influence on a person's activity. Our use of Activity Theory introduces a consistent separation of context elements and at the same time maintains a consistent set of relationships between them. Such a treatment of context elements has not been proposed in the context awareness field before.

We reviewed the features of our use of Activity Theory against the problems in previous context definitions and classifications. At the end of Chapter 3 we proposed a new context model. In our context model, a temporal dimension, "a timeline", is added to represent history, present and future in the Activity Theory model. For each Activity Theory model on the timeline, it also includes information on the environment and time that have an impact on the user's activity. By exploiting Activity Theory in our context model, it not only provides a consistent set of context elements in the context model but also provides systematic relationships between them. Therefore the history of each context element can be stored separately from each other and from the history of context models. The history of context elements and context model can be used in inferring about the user in a consistent manner according to the consistent

297

relationships in the context model rather than, for example, inconsistent embedded rules in the context databases.

From the new context model, context is defined as: "Related sets of attributes of information about the user, community, role, rule, tools, environment and time that have an influence on the user in achieving her current objective or goal".

8.1.4 Design Tool

Based on the proposed context model and Activity Theory concept, we suggested a new design tool that is composed of six steps. These six steps are:

1. *Nature of User Requirements*

As stories and descriptions in scenarios, often themselves drawn from field studies, are a main source of user requirements in ubiquitous computing , we suggested that the designers concentrate on the requirements for the user in different scenario-driven situations. A scenario was then created to show how context awareness can support the user in each situation and particular activity.

2. *Define Situations that Context Awareness Can Support*

The context model was used by the designers to generate a simple structure of each situation with possible functionalities of applications that support the user in the situation. The simple structure of each situation which was represented as a context model and the concept of three levels of activity in Activity Theory can be used by the developers during implementation. The context model can be used together with the descriptive scenario for the designers, implementers and users to develop better shared understandings about the situation.

3.   From Situation to Elements in the Context Model

For each extracted situation, the definition of each element in the context model guides the designers to make a decision about the information that should be used for that element in order to infer about the user's objective.

4.   From Context Element to Sensors and Profiles

The results from step 3 may be used by the designers to discuss with the implementers the possibilities for collecting and processing values from various sensor technologies. The developers assign a sensor where possible and create profiles for necessary information.

5.   From Context Elements to Reasoning

The sets of values for context elements and context model are extracted from the scenario for different situations. The values are stored in the databases for the system to use during the inference process in real time against the current context values. The relationships in the context model provide the developers with a well structured reasoning process about undiscovered context elements (such as role, rule, outcome context elements) and about the inference process for the user's current objective.

6.   From Outcome Context to Selected Application and Context

From the defined outcome and functionalities of the application, the designers can guide the implementers in developing an application that supports the

functionalities. The functionalities can be passive or active computing that takes information from the current context model where appropriate.

Chapter 6 discussed how our context model meets the requirements for a design tool introduced in Chapter 2. It then described the details of the design tool and described how the context model is used and itself evolves as part of the process of use, with a simple example of each step.

8.1.5   Architecture

Based on the design tool, we proposed a three layered context-aware system architecture to support the facilities that were introduced by Activity Theory in the context model. The architecture was described in Chapter 5. It included:

- The Sensor Engine Layer containing

    o   Sensors which provide the facility of acquiring raw data from a particular sensor.

    o   Sensor translators that provide an interpretation of the raw data or a set of raw data into less noisy and more meaningful information, and store it in the sensor database.

    o   A Sensor engine that identifies what sensors and profiles are available in different situations. It combines meaningful information from different sensors and profiles to get values for the attributes in different context elements.

- The Context Engine Layer which provides a reasoning process in order to combine information from the available context elements to get missing elements via the use of history of the context elements and context models.

- The Application Engine Layer which uses the IDs of the identified context elements in the application database to gather relevant information. The information is then used to determine the representation of the application to the user. The application engine then executes the chosen application with the relevant contextual information for supporting the user, and also facilitates user interrogation and refinement of the underlying context model.

Chapter 7 discussed how our architecture meets the requirements introduced in Chapter 2. It described the details of the architecture including how the data flows in the architecture.

8.1.6   Scenarios

Chapter 6 demonstrated step by step how we used the six steps of the design tool to transform a descriptive scenario into well structured sets of information – the database structure to be used in the architecture. A simple conference assistant scenario, which is a common scenario and has been used in previous context awareness projects, and a more complex hospital reception assistant scenario were used. In the more complex hospital scenario, the user is working under pressure and various multiple tasks are associated with different roles of the user. The use of the design tool was evaluated against the requirements proposed in Chapter 2. The results from applying the design tool to two different scenarios illustrated how the design tool could be used to simplify and systematise the design process in both domains so that the designers could concentrate more on meeting the user's requirements.

Furthermore, the results from the design tool were used to implement a prototype context-aware system including the architecture for the exemplar scenarios. This demonstrated how the architecture implemented from the context model simplified the process of adding support for a new situation or domain to the architecture.

The clear separations between the context elements and the reasoning processes allowed an easier process of reusing or remodelling context. The three layered architecture also supported the separation between sensor acquisition and the applications.

8.2   *Applicability to Other Applications*

Two scenarios from different domains were demonstrated. In general the design tool and architecture can be used to support designers in any domains. The design tool allows the designers and developers to communicate through a uniform context model. The context model provides a clear separation between context elements and consistent relationship between them. The designers can use it to analyse the user requirement systematically. The system can them meet the user's requirement and extract context information accordingly. Although by avoiding the technology driven approach, the developers require further effort and time to design and develop profiles and databases to provide information that the system requires where the technology is not available yet. This will be worthwhile when the system can be expanded and reused more easily in the future.

In order to apply the design tool output, the developers have to use our architecture in order to take full advantages of the design output. For example, the architecture provides a separation of concerns. The clear separation between sensor data and the context elements allows developers to change the methods of processing or interpreting the data from the sensors. Therefore developers who use different algorithms in interpreting the raw data will be assisted by the use of the design tool as they can plug in their algorithm into our sensor translator without affecting the other parts. For example, instead of using raw data from GPS in the sensor object, an algorithm for getting a mean value of the GPS data over 10 seconds could be added to the sensor object. There will be no consequent change in other parts of

the architecture. Similarly, instead of using attribute-value tuples to reason about the context model, the developer can use new algorithms or methods for reasoning about the context model such as fuzzy logic, forward and backward chaining rule engine or 4-ary predicates.

8.3 *Future Work*

8.3.1 Improve the Reasoning Algorithm

As noted previously, context has different properties, some more persistent than others. Activity Theory introduces a separation between consistent elements that influence the user's activity. Consequently, the context model separates the context into these consistent context elements with different properties. There is potential for using such properties in the reasoning process. In this work we use a simple best match function which compare characters of the attribute values. The more advanced techniques such as fuzzy logic, forward and backward chaining rule engine should be studied in order to improve the efficiency in reasoning process. For example, for the rule based logical inference, a weighting system of different types of context could be used, with different weights being applied for context elements that have different properties. Further analysis could be done to investigate if this approach improves the efficiency of the system. If it does, more research should be conducted to determine how each property should be applied.

8.3.2 Security and Privacy

The architecture here does not provide a real mechanism for security and privacy. However, as Activity Theory emphasises the division of labour and role concepts, there are possibilities of using rules and role elements to add security and privacy mechanisms to the architecture. Rules can be used to limit the access of the user to certain information or tools – cf. policy driven rules [Keeney and Cahill, 2003].

In addition, as the context element and profile databases are clearly separated, the user can choose to store private information such as the user's profile where they think it will be safe, as long as they notify the system where it is. For example, the user may not want to share her medical record with the public so she could move it to a trusted device or space (such as secure file servers). The user then notifies the system (i.e. updates the user's profile database) where the resource of the medical record is now stored.

8.3.3   Context Model Representation

The user's mental model [Johnson-Laird, 1983; Payne, 2003] of the system is important for the user to be able to interact with the system efficiently. The user interface and the representation to the user of the current context model that is influencing the decisions of the system should be further analysed. These are potentially very useful supports for the user's model and the system's model to be consistent, or at least for the user and the system to have some understanding of where their models differ. There are many possible ways of representing information about the system's context model. For example, the context model diagram based on Activity Theory's triangular model, photos, cartoons, tables or lists. The different methods should be evaluated and compared in order to investigate how each method influences the user's understanding of the system.

8.3.4   Real Time Efficiency Improvements Involving Users

As described in Chapter 6 and Chapter 7, the design tool and architecture support the possibility of real time efficiency improvements that involve users. Further investigation could be carried out into this. First of all, as suggested by Section 8.3.3, different methods of context representation should be studied in order to

support the user in giving feedback or correcting mistakes made by the context-aware system. Secondly, the impact on user experience of improving the context reasoning in real time should be studied. Thirdly, the possibility of allowing users to add new events to the context history should be studied. By allowing users to add their own events, they can extend the use of the system to meet their needs.

8.3.5 Coping with Limited Storage Space

As there is, however large, a limit to storage space, it is not feasible to store all available context. Therefore, methods of prioritising context for storage should be considered. For example, the frequency and recency of context data should be taken into account so that old and least used data can be downgraded or removed from the database. Concepts such as LRFU (Least recently/Frequently Used) policy [Lee, et al., 2001] can be added as a new attribute in each structured database in our context architecture. The LRFU value can be stored for each dataset in the database. For every new dataset, the LRFU values of existing datasets are compared and the dataset with the least frequency and least recently used is replaced with the new dataset. Further research should be done in order to find a range of suitable methods for increasing context storage efficiency.

8.3.6 Integrating Our Design Tool with Other Design Mechanisms

Our design tool can be divided into two parts. First, design tool (step 1-2 and 6) provides consistent steps for designers to model the system to support the user in a context awareness manner. Second, design tool (step 3-5) provides a uniform model for transforming user's requirement into context model. It will advance the system design process if the design tool is studied further in order to be applied to existing language for software modelling and designing such as Unified Modeling Language (UML) and object-modelling technique (OMT). As discussed in Section 3.3, Activity Modelling extends Usage-Centred Design by introducing new notations which are related to the Unified Modeling Language (UML) used in

software engineering [Fowler and Scott, 1997]. It demonstrates the possibility of using first part of our design tool with UML for software modelling and designing. This could leads to the further studies of multiusers system.

8.3.7 Relating Our Framework with Others Approaches

Our context-aware system architecture provides the separation of concern between 3 layers and databases. Each layer has its own different objects to handle different tasks. The sufficient in dividing tasks in the architecture should be studied further. The uniform separation should be able to apply to different approaches. The relevant of different approaches such as Aspect-oriented programming (AOP), multi-agent system (MAS) and service-oriented architecture (SOA)) should be investigated in order to further the efficient of the architecture.

8.4 *Conclusion*

This dissertation presents a context model and design tool that adopt the consistent simple triangular structure of human activity proposed by Activity Theory, in order to facilitate shared understandings about context and context reasoning amongst designers, developers and users. The standard model provided by Activity Theory was extended to include a temporal dimension in order to model the history of context. The resulting context model not only offers a consistent set of context elements. It also provides consistent context reasoning through the model. In order to support the facilities introduced by the context model and design tool a three layered implementation architecture was introduced. It provides separation between objects that deal with different types of context, with different properties, and the context reasoning. As a result, the context can be reused or remodelled for different domains with relative ease. Through two example scenarios, the context model, design tool and implementation architecture

have been demonstrated and evaluated against the requirements which we have proposed for them.

References

Abowd, G., Atkeson, C., Aust, D., and Long, S. (1996). Cyberguide: Prototyping Context-Aware Mobile Applications. In Proceedings of the Human Factors in Computing Systems (CHI), ACM Press, Vancouver, BC Canada. 293-294.

Abowd, G. D., and Dey, A. K. (2000). CybreMinder: A Context-Aware System for Supporting Reminders. In Proceedings of the Second Int. Symposium on Handheld and Ubiquitous Computing (HUC), Springer Verlag, Bristol, UK. 172-186.

Agarawala, A., Greenberg, S., and Ho, G. (2004). The Context-Aware Pill Bottle and Medication Monitor. In Proceedings of the Sixth International Conference on Ubiquitous Computing, Nottingham, England.

Aizawa, K., Hori, T., Kawasaki, S., and Ishikawa, T. (2004). Capture and Efficient Retrieval of Life Log. In Proceedings of the Pervasive Workshop on Memory and Sharing Experiences, Vienna, Austria. 15-20.

Antifakos, S., Schiele, B. and Holmquist, L. E. (2003) Grouping Mechanisms for Smart Objects Based On Implicit Interaction and Context Proximity. In Proceedings of UBICOMP 2003 Interactive, 207-208.

Aoki, P. M., Grinter, R. E., Hurst, A., Szymanski, M. H., Thornton, J. D., and Woodruff, A. (2002). Sotto Voce: Exploring the Interplay of Conversation

and Mobile Audio Spaces. In Proceedings of the Human Factors in Computing Systems (CHI), ACM Press, Minneapolis, MN. 431-438.

Bai, Y., Ji, H., Han, Q., Huang, J., and Qian, D. (2007). MidCASE : A Service Oriented Middleware Enabling Context Awareness for Smart Environment. In Proceedings of the International Conference on Multimedia and Ubiquitous Engineering (MUE), Seoul, Korea. 946-951.

Baldauf, M., Dustdar, S., and Rosenberg, F. (2006). A Survey on Context-Aware systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2, 263-277.

Baldwin, D. A. and Baird, J. A. (2001). Discerning intentions in dynamic human action. *Trends in Cognitive Sciences*, 5(4), 171-178. Becker, C., and Nicklas, D. (2004). Where do spatial context-models end and where do ontologies start? A proposal of a combined approach. In Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management in conjunction with UbiComp, Nottingham, England.

Barkhuus, L. and Dey, A. (2003). Is context-aware computing taking control away from the user? Three levels of interactivity examined. In Proceedings of Ubicomp 2003, 149-156.

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

Benerecetti, M., Bonifacio, M., and Bouquet, P. (2001). Distributed Context-Aware Systems. *Human-Computer Interaction* 16, 213-228.

Benford, S., Anastasi, R., Flintham, M., Drozd, A., Crabtree, A., and Greenhalgh, C. (2003). Coping with uncertainty in a location-based game. *IEEE Pervasive Computing* 2, 34-41.

Biegel, G., and Cahill, V. (2004). A Framework for Developing Mobile, Context-aware Applications. In Proceedings of the Second IEEE Annual Conference on Pervasive Computing (PerCom). 361 - 365.

Brown, B., and Randell, R. (2002). Building a context sensitive telephone: Some hopes and pitfalls for context sensitive computing. *Glasgow Context Group 1st Symposium, Building Bridges: Interdisciplinary Context-Sensitive Computing*

Brown, P. J. (1996). The Stick-e Document: A Framework for Creating Context-Aware Applications. In Proceedings of the Electronic Publishing. 259-272.

Bucur, O., Beaune, P., and Boissier, O. (2005). Representing context in an agent architecture for context-based decision making. In Proceedings of the Workshop on Context Representation and Reasoning (CRR'05), Paris, France.

Capra, L., Emmerich, W., Mascolo, C., and Zachariadis, S. (2001). Towards a Mobile Computing Middleware: A Synergy of Reflection and Mobile Code Techniques. In Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems, Bologna, Italy. 148-154.

Carroll, J. M. (1999). Five Reasons for Scenario-Based Design. In Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences, IEEE Computer Society, Maui, HI, USA.

Chalmers, D., and Sloman, M. (1999). QoS and Context Awareness for Mobile Computing. In Proceedings of the Handheld and Ubiquitous Computing, First International Symposium (HUC), Springer, Karlsruhe, Germany. 380-382.

Chalmers, D., Dulay, N., and Sloman, M. (2004). A framework for contextual mediation in mobile and ubiquitous computing applied to the context-aware adaptation of maps. *Personal and Ubiquitous Computing* 8, 1-18.

Chalmers, M. (2004). A Historical View of Context. *The Journal of Collaborative Computing: Computer Supported Cooperative Work* 13, 223-247.

Chen, G., and Kotz, D. (2000). *A Survey of Context-Aware Mobile Computing Research* (Report Number TR2000-381). Dartmouth College, Department of Computer Science, UK.

Chen, H., Finin, T., and Joshi, A. (2003). *An Ontology for Context Aware Pervasive Computing Environments*, Acapulco, Mexico.

Connolly, D., van Harmelen, F., Horrocks, I., McGuinness,D. L., Patel-Schneider, P. F. and Stein, L. A. (2001) "DAML+OIL (March 2001) Reference Description," http://www.daml.org/2001/03/daml+oil-index

Constantine, L. L. (1995). Essential Modeling: Use Cases for User Interfaces. *Interactions* 2, 34-46.

Constantine, L. L. (1998). Abstract Prototyping. *Software Development* 6,

Constantine, L. L. (2006). *Activity Modelling: Toward a Pragmatic Integration of Activity Theory with Usage-Centered Design*. Laboratory for Usage-Centered Software Engineering.

Conway, A. S. M. (2006). *Survey of Context aware Frameworks. – Analysis and Criticism*. The University of North Carolina, Chapel Hill.

Crowley, J. (2006). Situation Models for Observing Human Activity. *ACM*

Crystal, A., and Ellington, B. (2004). Task analysis and human-computer interaction: approaches, techniques, and levels of analysis. In Proceedings of the Tenth Americas Conference on Information Systems (AMCIS), New York, NY.

Davies, N., Cheverst, K., Mitchell, K., and Efrat, A. (2001). Using and Determining Location in a Context-Sensitive Tour Guide. *IEEE Computer* 34, 35-41.

Derntl, M., and Hummel, K. A. (2005). Modeling context-aware e-learning scenarios. In Proceedings of the Pervasive Computing and Communications Workshops, PerCom 2005. 337 - 342.

Desmet, B., Vallejos, J., Costanza, P., and Hirschfeld, R. (2007). Layered design approach for context-aware systems. In Proceedings of the First International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2007), Limerick, Ireland.

Dey, A. K., and Abowd, G. D. (1999). *Toward a Better Understanding of Context and Context-Awareness* (Report Number GITGVU -99-22). Georgia Institute of Technology, Atlanta, GA, USA.

Dey, A. K., Salber, D., Futakawa, M., and Abowd, G. D. (1999). *An Architecture to Support Context-Aware Applications*. Georgia Institute of Technology.

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

Dey, A. K. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing* 5, 4-7.

Dey, A. K., Abowd, G. D., and Salber, D. (2001). A Conceptual Framework and A Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction* 16, 97-166.

DirectGov. (2005). *Percentage of households with selected consumer durables,* Expenditure and Food Survey, Office for National Statistics*, UK*. http://www.statistics.gov.uk/cci/nugget.asp?id=868Dix,

A., Rodden, T., Davies, N., Trevor, J., Friday, A., and Palfreyman, K. (2000). Exploiting Space and Location as a Design Framework for Interactive Mobile Systems. *ACM Transactions on Computer-Human Interaction* 7, 285–321.

Dong, J. S., Feng, Y., Sun, J., and Sun, J. (2006). Context Awareness Systems Design and Reasoning. In Proceedings of the Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), Paphos, Cyprus. 335-340.

Dourish. P. (2004) What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1), 19–30.

Dreyfus, H. (2001). *On the Internet: Thinking in Action*. Routledge, London, UK.

Engeström, Y. (1987). Learning by expanding: An activity-theoretical approach to developmental research (Helsinki, Orienta-Konsultit).

Engeström, Y., Miettinen, R., and Punamäki, R. L., eds. (1999). *Perspectives on Activity Theory*. Cambridge University Press.

Fahy, P., and Clarke, S. (2004). CASS – Middleware for Mobile Context-Aware Applications. In Proceedings of the Mobisys, Boston, USA.

Flintham, M., Anastasi, R., Benford, S., Hemmings, T., Crabtree, A., Greenhalgh, C., Rodden, T., Tandavanitj, N., Adams, M., and Row-Farr, J. (2003). Where on-line meets on-the-streets: experiences with mobile mixed reality games. In Proceedings of the Human Factors in Computing Systems (CHI), ACM Press, Florida. 569-576.

Fowler, M., and Scott, K. (1997). *UML Distilled.* Addison-Wesley, Reading, MA.

Gay, G., and Hembrooke, H. (2004). *Activity - Centered Design : an ecolgical approach to designing smart tools and usable systems*. MIT Press, Cambridge, Massachusetts.

Gellersen, H.-W., Schmidt, A., and Beigl, M. (2002). Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts. *Mobile Networks and Applications* 7, 341-351.

Giarratano, J., Riley, G., and Riley, G. D. (2004). *Expert Systems: Principles and Programming*. Course Technology Ptr.

Gonzalez, A. J., and Ahlers, R. (1999). Context-Based Representation

of Intelligent Behaviour in Training Simulations. *Transactions of the Society for Computer Simulation International* 15,

Goodwin, C., and Duranti, A. (1992). *Rethinking context: an introduction.* Cambridge University Press.

315

Grant, S. (1992). A context model needed for complex tasks. *Mental Models and Everyday Activities: Proceedings of the Second Interdisciplinary Workshop on Mental Models,* Cambridge, England, 94-102.

Greenberg, S. (2001). Context As A Dynamic Construct. *Human Computer Interaction* 16, 257-268.

Gu, T., Pung, H. K., and Zhang, D. Q. (2004). A middleware for building context-aware mobile services. In Proceedings of the IEEE 59th Vehicular Technology Conference (VTC), Milan, Italy. 2656 - 2660.

Haghighat, A., Lopes, C. V., Givargis, T., and Mandal, A. (2004). Location-Aware Web System. In Proceedings of the OOPSLA, Vancouver.

Hariharan, R., and Toyama, K. (2004). Project Lachesis: Parsing and Modeling Location Histories. *Geographic Information Science* 106-124.

Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., and Jansen, E. (2005). The Gator Tech Smart House: A Programmable Pervasive Space. *Computer* 38, 50-60.

Helander, J. (2005). Exploiting Context Histories in Setting up an e-Home. In Proceedings of the 1st International Workshop on Exploiting Context Histories in Smart Environments, Munich, Germany.

Henricksen, K., Indulska, J., and Rakotonirainy, A. (2002). Modeling Context Information in Pervasive Compututing Systems. In Proceedings of the Pervasive Computing, Springer, Zurich, Switzerland,. 167-180.

Henrickson, K. (2003). A framework for context-aware pervasive computing applications, School of Information Technology and Electrical Engineering,, The University of Queensland.

Hertzog, P., and Torrens, M. (2004). Context-aware mobile assistants for optimal interaction: a prototype for supporting the business traveler. In Proceedings of the 9th international conference on Intelligent user interface, ACM Press, Funchal, Madeira, Portugal. 256-258.

Hinze, A., and Viosard, A. (2003). Location- and time-based information delivery in tourist. In Proceedings of the Advances in spatial and temporal databases (SSTD), Springer. 489-507.

Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., and Altmann, J. (2003). Context-Awareness on Mobile Devices - the Hydrogen Approach. In Proceedings of the 36th Hawaii International Conference on System Science (HICSS), IEEE Computer Society, Waikoloa, Big Island, Hawaii, USA. 292-301.

Hopper, A., Jones, A., and Ward, A. (1997). A New Location Technique for the Active Office. *IEEE Personnel Communications* 4, 42-47.

Hsu, J.-M., Wu, W.-J., and Chang, I.-R. (2007). Ubiquitous Multimedia Information Delivering Service for Smart Home. In Proceedings of the Multimedia and Ubiquitous Engineering (MUE). 341-346.

Hull, R., Bedford-Roberts, J., and Neaves, P. (1997). Towards Situated Computing. In Proceedings of the First Int. Symposium on Wearable Computers, IEEE Computer Society Press, Cambridge, Massachusetts. 146-153.

Ipiña, D. L. d. (2001). An ECA Rule-Matching Service for Simpler Development of Reactive Applications. In Proceedings of the Middleware 2001.

Jameson, A., and Klöckner, K. (2005). User Multitasking with Mobile Multimodal Systems. In *Spoken Multimodal Human-Computer Dialogue in Mobile Environments* (Minker, W., Bühler, D., and Dybkjær, L., eds.), pp. 349-377. Springer, Netherlands.

Jiang, X., Chen, N. Y., Hong, J. I., Wang, K., Takayama, L., and Landay, J. A. (2004). Siren: Context-aware Computing for Firefighting. In Proceedings of the Second International Conference on Pervasive Computing (Pervasive), Vienna, Austria. 87-105.

Johnson-Laird, P. N. (1983). *Mental models*. Cambridge University Press, Cambridge.

Kaenampornpan, M., and O'Neill, E. (2004a). Modelling context: an Activity Theory approach. In Proceedings of the Ambient Intelligence: Second European Symposium (EUSAI), Springer, Eindhoven, The Netherlands. 367-374.

Kaenampornpan, M., and O'Neill, E. (2004b). An Integrated Context Model: Bringing Activity to Context. In Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management. In conjunction with The sixth international UbiComp Conference., University of Southampton, Nottingham, England. 17-21.

Kaptelinin, V., and Nardi, B. (1997). Activity Theory: Basic Concepts and Application. In Proceedings of the CHI, Los Angeles.

318

Keeney, J., and Cahill, V. (2003). Chisel: A policy-driven, Context-aware, dynamic Adaptation framework. In Proceedings of the Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY), Lake Como, Italy. 3-14.

Kim, E., Plummer, M., Hiltz, S. R., and Jones, Q. (2007). Perceived Benefits and Concerns of Prospective Users of the SmartCampus Location-Aware Community System Test-bed. In Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS), Waikoloa, HI. 19.

Kim, S. W., Kim, M. C., Park, S. H., Jin, Y. K., and Choi, W. S. (2004). Gate reminder: a design case of a smart reminder. In Proceedings of the Designing interactive systems: processes, practices, methods, and techniques, ACM Press, Cambridge, MA, USA.

Korpipää, P., Mäntyjärvi, J., Kela, J., Keränen, H., and Malm, E.-J. (2003). Managing Context Information in Mobile Devices. *Pervasive Computing* 42-51.

Kostakos, V., and O'Neill, E. (2003). A directional stroke recognition technique for mobile interaction in a pervasive computing world, People and Computers XVII. In Proceedings of the HCI: Designing for Society, Bath, UK. 197-206.

Kröner, A., Heckmann, D., and Wahlster, W. (2006). SPECTER: Building, Exploiting, and Sharing Augmented Memories. In Proceedings of the Knowledge Sharing for Everyday Life (KSEL), Kyoto, Japan.

Kuutti, K. (1995). Activity Theory as a potential framework for human-computer interaction research. In *Context and conciousness: Activity Theory and*

human computer interaction (Nardi, B., ed, pp. 17-44. MIT Press, Cambridge.

Lee, D., Choi, J., Kim, J.-H., Noh, S. H., Min, S. L., Cho, Y., and Kim, C. S. (2001). LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies. *IEEE Transctions on Computer* 50, 1352-1361.

Lee, D., and Meier, R. (2007). Primary-Context Model and Ontology: A Combined Approach for Pervasive Transportation Services. In Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom), White Plains, NY.

Leontiev, A. N., ed. (1979). *The problem of activity in psychology*. Sharpe, New York.

Lieberman, H., and Selker, T. (2000). Out of context: Computer Systems that Adapt to, and Learn from, Context. *IBM Systems Journal* 39, 617-632.

Little, L., Briggs, P., and Cventry, L. (2003). An Activity Theory Approach to Technology Use in Public Areas: The case of the ATM. In Proceedings of the 17th British HCI Annual Conference, Bath. 45-48.

Lucas, P. (2001). Mobile Devices and Mobile Data-Issues of Identity and Reference. *Human-Computer Interaction* 16, 323-336.

Lueg, C. (2001). On context-aware artifacts and socially responsible design. In Proceedings of the Annual Conference of the Computer Human Interaction Special Interest Group of the Ergonomics Society of Australia. 84-89.

320

Lueg, C. (2002). On the Gap between vision and feasibility. In Proceedings of the Pervasive computing, Springer Lecture Note in Computer science (LNCS 141), Zurich, Switzerland. 45-57.

Lukkari, J., Korhonen, J., and Ojala, T. (2004). SmartRestaurant: mobile payments in context-aware environment. In Proceedings of the the 6th international conference on Electronic commerce, ACM Press, Delft, The Netherlands. 575-582.

Luyten, K., and Coninx, K. (2004). ImogI: Take Control over a Context-Aware Electronic Mobile Guide for Museums. In Proceedings of the HCI in Mobile Guides, in conjunction with Mobile HCI, Glasgow, Scotland.

Mappin, D. A. (2000). *Edpy 597: Advanced Instructional Design - Diving Deeper*. University of Alberta.

Mayrhofer, R. (2005). Context Prediction based on Context Histories: Expected Benefits, Issues and Current State-of-the-Art. In Proceedings of the 1st International Workshop on Exploiting Context Histories in Smart Environments, Munich, Germany.

McCrickard, D. S., Chewar, C. M., Somervell, J. P., and Ali, N. (2003). A model for notification systems evaluation—assessing user goals for multitasking activity. In Proceedings of the TOCHI, ACM Transactions on Computer-Human Interaction. 312-338.

McGuinness, D. L., and Harmelen, F. v. (2004). *OWL Web Ontology Language Overview*. W3C.

Meier, R. e., and Cahill, V. (2003). Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications. In Proceedings of the

4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), Springer-Verlag Heidelberg, Germany.

Minker, W., Dybkjaer, L., and Buhler, D. (2005). *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*. Kluwer Academic Publishers.

Mobile Data Association. (2006). *Press Releases: Latest Statistics*. MDA Press Office.

Mohr, P., Timmis, J., and Ryan, N. (2005). Immune Inspired Context Memory. In Proceedings of the 1st International Workshop on Exploiting Context Histories in Smart Environments, Munich, Germany.

Mori, G., PaternoÁ, F., and Santoro, C. (2002). CTTE: Support for Developing and Analysing Task Models for Interactive System Design. *IEEE Transactions on Software Engineering* 28, 797-813.

Morse, D., Pascoe, J., and Ryan, N. (1997). *Enhanced Reality Fieldwork: The Context-Aware Archeological assistant*. *Computer Applications in Archeology*

Muñoz, M. A., Rodríguez, M., Favela, J., Martinez-Garcia, A. I., and González, V. M. (2003). Context-Aware Mobile Communication in Hospitals. *Computer* 36, 38-46.

Norman, D. A., ed. (1983). *Some Observations on mental models*. L. Erlbaum Associates.

O'Hara, K., Glancy, M., and Robertshaw, S. (2008) Understanding collective play in an urban screen game. *CSCW '08: Proceedings of the ACM conference on Computer supported cooperative work*, CA, USA. 67-76

O'Hara, K., Perry, M., Churchill, E. and Russell, D. (2003) Public and Situated Displays: Social and interactional aspects of shared display technologies. Kluwer.

O'Neill, E., and Johnson, P. (2004). Participatory Task Modelling: users and developers modelling users' tasks and domains. In Proceedings of the TAMODIA, 3rd International Workshop on task models and diagrams for user interface design, ACM Press, Prague, Czech Republic.

O'Neill, E., Kaenampornpan, M., Kostakos, V., Warr, A., and Woodgate, D. (2006). Can we do without GUIs? Gesture and speech interaction with a patient information system. *Personal and Ubiquitous Computing* 10, 269-283.

O'Neill, E., Woodgate, D., and Kostakos, V. (2004). Easing the wait in the emergency room: building a theory of public information systems. In Proceedings of the Designing interactive systems: processes, practices, methods, and techniques, ACM Press, Cambridge, MA, USA. 17 - 25.

Oh, Y. S., Yoon, H. S., and Woo, W. T. (2006). Simulating Context-Aware Systems based on Personal Devices. In Proceedings of the International Symposium on Ubiquitous VR (ISUVR). 49-52.

Oviatt, S., Coulston, R., and Lunsford, R. (2004). When Do We Interact Multimodally? Cognitive Load and Multimodal Communication Patterns. In Proceedings of the ICMI. 129-136.

Paganelli, F., and Giuli, D. (2007). An Ontology-Based Context Model for Home Health Monitoring and Alerting in Chronic Patient Care Networks. In Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops, IEEE Computer Society. 838-845.

Park, D. G., Kim, J. K., Sung, J. B., Hwang, J. H., Hyung, C. H., and Kang, S. W. (2006). TAP: touch-and-play. In Proceedings of the Human Factors in Computing Systems, ACM Press, Montréal, Québec, Canada. 677-680.

Pascoe, J. (1998). Adding Generic Contextual Capabilities to Wearable Computers. In Proceedings of the 2nd International Symposium on Wearable Computers, IEEE Computer Society, Pittsburgh, PA. 92-99.

Paternò, F. (1999). *Model-based design and evaluation of interactive applications*. Springer-Verlag, London.

Payne, S. J. (2003). Users' Mental Models: The Very Ideas. In *HCI Models, Theories, and Frameworks Toward a Multidisciplinary Science* (Carroll, J. M., ed, pp. 27-54. Morgan Kaufmann, San Francisco.

Pfeifer, R., and Rademakers, P. (1991). Situated adaptive design: Toward a methodology forknowledge systems development. In Proceedings of the Verteilte Kunstliche Intelligenz und kooper-atives Arbeiten: 4. Internationaler GI-Kongress Wissensbasierte Systeme, Springer-Verlag, Berlin.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. (1999). *Human-Computer Interaction*. Addison-Wesley.

Pylyshyn, Z. W. (1987). *The robot's Dilemma: the frame problem in artificial intelligence*. Publishing Corporation, Norwood, NJ.

Pyramid Research. (2005). *Wi-Fi Gaining Traction*. businessweek.com.

Roberts, G., Meng, X., and Dodson, A. (2005). Using adaptive filtering to detect multipath and cycle slips in GPS/Accelerometer bridge deflection monitoring data. In Proceedings of the ISGPAP??

Rodriguez, H. (1998). *Activity theory and Cognitive Sciences*.

Rogers, Y., and Scaife, M. (1997). *Activity Theory*. COGS, University of Sussex.

Román, M., Hess, C. K., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K. (2002). Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing* 74-83.

Saternus, M., Weis, T., Knoll, M., and Durr, F. (2007). A Middleware for Context-Aware Applications and Services Based on Messenger Protocols. In Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom). 467-471.

Schilit, B., Adams, N., and Want, R. (1994). Context-Aware Computing Applications. In Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, US. 85-90.

Schilit, B., and Theimer, M. (1994). Disseminating Active Map Information to Mobile Hosts. *IEEE Network* 8, 22-32.

Schmidt, A., Adoo, K. A., Takaluoma, A., Tuomela, U., Velde, W. V. d., and Laerhoven, K. V. (1999a). Advanced Interaction in Context. *Lecture Notes in Computer Science* 1707, 89-101.

Schmidt, A., Beigl, M., and Gellersen, H. W. (1999b). There is More to Context Than Location. *Computers and Graphics* 23, 893-901.

Schmidt, A. (2000). Implicit Human Computer Interaction Through Context. *Personal Technologies* 4, 191-199.

Schumacher, E. H., SeyMour, T. L., Glass, J. M., Fencsik, D. E., Lauber, E. J., Kieras, D. E., and Meyer, D. E. (2001). Virtually perfect time sharing in dual-task performance: uncorking the central cognitive bottleneck. *Psychological Science* 12, 101-108.

Shepherd, A. (1989). Analysis and training in information technology tasks. In *Task Analysis for Human-Computer Interaction* (Diaper, D., ed, pp. 15-55. Chichester: Ellis Horwood.

Shepherd, A. (1998). *Hierarchical task analysis*. Taylor & Francis.

Siewiorek, D. P. e. a. (2003). SenSay: A context-aware mobile phone. In Proceedings of the International Symposium on wearable computers (ISWC), White Plains, New York, USA.

Smith, M, K., Welty, C., McGuinness, D. L. (2003). *OWL Web Ontology Language Guide*, W3C http://www.w3.org/TR/2003/CR-owl-guide-20030818/

St.Laurent, S. (1998). *Why XML?* www.simonstl.com/articles/whyxml.htm

Stephanidis, C. (2001). *Ambient Intelligence in the Context of Universal Access.* ERCIM.

Sumi, Y., and Mase, K. (2001). Digital Assistant for Supporting Conference Participants: An Attempt to Combine Mobile, Ubiquitous and Web Computing. *Lecture Notes In Computer Science* 2201, 156-175.

Thomson, G., Nixon, P., and Terzis, S. (2005). Situation Determination with Distributed context histories. In Proceedings of the 1st Int workshop on exploiting context histories in smart environments, Pervasive, Munich, Germany.

Tonnis, M., Klinker, G., and Jan-Gregor, F. (2007). Ontology-Based Pervasive Spatial Knowledge for Car Driver Assistance. In Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom). 401-406.

Van Der Veer, G. C., Lenting, B. F., and Bergevoet, B. A. J. (1996). GTA: Groupware Task Analysis - Modeling Complexity. *Acta Psychologica* 91, 297-322.

Weiser, M (1999). The Computer for the 21st Century. *ACM SIGMOBILE Mobile Computing and Communications Review 3(3) 3-11.*

Winograd, T. (2001). Architectures for Context. *Human-Computer Interaction* 16, 401-419.

Appendix

I. Example of XML File for Environment data

In order to represent the set of values to the user, a short name (nm in the XML in Figure I) of the set of values is assigned to every set of data so that it can be presented to the user and makes sense to the user during run time. For example, the Reception Desk is assigned as a name of the set of values (This is represented as *nm* in XML file) from Table 6-32. Any value that is not available in the table for each attribute in the element is assigned as "unknown".

```
<environments>
 <environment>
   <ID>1</ID>
   <nm>Reception Desk</nm>
   <building>West</building>
   <room>Reception</room>
   <area>Hospital</area>
   <town>London</town>
   <country>UK</country>
   <condition>unknown</condition>
 </environment>

<tools>
  <tool>
<ID>2</ID>
<dname>Reception Phone</dname>
<dtype>mobile</dtype>
<owner>hospital</owner>
<bt>y</bt>
  </tool>
<tools>
```

Figure I XML Codes for Environment Database and Tool Database

II. Example of XML File for Tools.xml

The tools database holds the list of IDs of the public tools that are available in the situation as shown in Figure II. The IDs are the list of identities of the tools in the tools database where the further information about each tool can be acquired from the ID of each tool in Tool Database which is similar manner to Environment Database in Figure I.

```
<tools>
    <tool>
        <ID>1</ID>
        <TIDs>2</TIDs>
    </tool>
<tools>
```

Figure II XML Codes for Tools Database

III. Example of Object File for xmlEnv.java

Objects have two sections, fields (instance variables) and methods. Fields for the object for accessing environment information has variables of ID, nm, building, room, area, town, country and condition as a set of values in the environment object following the XML structure. *xmlEnv.java* has methods (such as *getNM()* and *setNM()*) that allow it to edit, change and access the value of the variable in the object as shown in Figure III. This process is followed for the other databases such as *xmlUser.java*, *xmlTool.java* and *xmlTools.java*. The methods in these programs allow the architecture to access these objects. Moreover, they allow access between objects themselves.

330

```java
public class xmlEnv {

    String ID, nm, building, room, area, time;

public xmlEnv(String id, String out1, String bd, String rm, String ar,String co)

{

    this.ID = id;
    this.nm = out1;
    this.building = bd;
    this.room = rm;
    this.area = ar;
    this.time = co;

}

...

  public String getID()

   {return ID;}

   public String getNM()

   {return nm;}

  public void setID(String id)

   { ID=id;}

   public void setNM(String ob)

   { nm=ob;}

   ....

}
```

Figure III Object Code for Dealing with Environment Information